Pràctica 4

Objectius:

a) Començar amb l'escriptura d'un programa en assemblador que emula una altra màquina.

b) Maneig de dades a nivell de bits.

Emulació d'una màquina elemental amb l'assemblador del 68K

Un programa emulador permet que un ordinador pugui executar un programa escrit per a una altra màquina. Durant les pròximes sessions (*P4*, *P5* i *P6*) ens dedicarem a escriure un programa que ens permetrà executar damunt l'emulador del *68K* qualsevol programa escrit per a una versió reduïda de la màquina elemental que es veu a classe de teoria. Aquesta nova màquina s'anomena *PEPAr* (*PEPA* **r**educed).

Per fer aquesta emulació, el nostre programa haurà de ser capaç de llegir de la memòria del 68K una seqüència de paraules codificades d'acord amb la codificació de les instruccions de la *PEPAr*, interpretar de quina instrucció es tracta i emular la seva execució. A més de la memoria per al programa i les dades, l'emulador també reservarà una sèrie de posicions de memòria per representar els registres de la màquina elemental.

Abans de descriure amb detall la feina concreta que s'ha de desenvolupar a aquesta pràctica, anem a introduir els conceptes necessaris de la *PEPAr*. A la taula següent s'indica la informació més rellevant del conjunt d'instruccions de la *PEPAr* a implementar. Noteu que aquestes instruccions representen un subconjunt no complet de la *PEPA* original vista a classe.

Id	Mnemònic	Codificació	Acció	Flags
0	JMI M	0000mmmmmmmm	PC ← M	n.s'a.
1	JMZ M	0001mmmmmmmm	si Z=1, PC ← M	n.s'a.
2	JMN M	0010mmmmmmmm	si N=1, PC ← M	n.s'a.
3	STORE M	010xmmmmmmmm	M ← [R1]	n.s'a.
4	LOAD M	011xmmmmmmmm	R1 ← [M]	n.s'a.
5	CMP	10xxxx000100	[R0] − [R1]	C, Z i N=s.v.r.
6	ADD	10xxxx010000	R0 ← [R0] + [R1]	C, Z i N=s.v.r.
7	SUB	10xxxx010100	R0 ← [R0] − [R1]	C, Z i N=s.v.r.
8	R1→R0	10xxxx011000	R0 ← [R1]	n.s'a.
9	R0→R1	10xxxx100000	R1 ← [R0]	n.s'a.
10	HALT	11xxxxxxxxxx	Atura la màquina	n.s'a.
n.s'a.: No s'actualitzen. s.v.r.: Segons el valor del resultat de l'operació.				

Normalment, quan s'escriu en assemblador s'utilitzen noms simbòlics per denotar variables i adreces de bot. Els programes de la màquina elemental que emularem hauran de dur adreces absolutes (numèriques) tant per als bots com per als operands. Així, a la figura següent el

programa de l'esquerra no es podrà executar a l'emulador que escriureu. Haureu d'escriure els programes com el de la dreta de la figura perquè el vostre emulador els pugui entendre. Tal com es fa a aquest exemple, per passar a adreces absolutes suposarem sempre que **el programa comença a la posició 0 de la memòria** de la màquina elemental (després del segond HALT es guarden les dades). Noteu que el programa carrega dues dades des de memòria, fa la resta i emmagatzema el resultat a una altra posició de memòria.

Etiqueta	Assemblador amb etiquetes	Adreça	Assemblador sense etiquetes	Instruccions codificades
	LOAD A	0:	LOAD 10	011x00001010
	R1→R0	1:	R1→R0	10xxxx011000
	LOAD B	2:	LOAD 11	011x00001011
	SUB	3:	SUB	10xxxx010100
	JMN ET1	4:	JMN 7	001000000111
	STORE C	5:	STORE 12	010x00001100
	HALT	6:	HALT	11xxxxxxxxxx
ET1:	R0→R1	7:	R0→R1	10xxxx100000
	STORE C	8:	STORE 12	010x00001100
	HALT	9:	HALT	11xxxxxxxxxx
A:	1	10:	1	000000000001
B:	2	11:	2	000000000010
C:	0	12:	0	000000000000

NOTA : per facilitar la distinció entre tot el relatiu a la màquina elemental i el relatiu al 68K s'afegirà a partir d'ara el prefix "e" a tot el que pertany a la primera. Així el registre R0 de la màquina elemental el denotarem ER0, els programes de la màquina elemental es diran eprogrames, etc.

Tenint en compte el que s'ha dit fins ara, el programa que heu de dissenyar i escriure començarà amb una capçalera com aquesta.

```
        ORG $1000
EPROG:  DC.W $060A, $0818, $060B, $0814, $0207, $040C, $0C00
        DC.W $0820, $040C, $0C00, $0001, $0002, $0000
EIR:    DC.W 0       ;eregistre d'instrucció
EPC:    DC.W 0       ;ecomptador de programa
ER0:    DC.W 0       ;eregistre R0 (acumulador)
ER1:    DC.W 0       ;eregistre R1 (buffer de memòria)
ESR:    DC.W 0       ;eregistre de flags (00000000 00000ZNC)
```

L'eprograma que es vulgui emular en cada cas s'introduirà com un vector de *words* a partir de l'etiqueta `EPROG`. Les **einstruccions** es codificaran en els **12 bits menys significatius** de cada *word*, afegint zeros davant per completar els 16 bits i posant 0's a les posicions *don't care*. Comproveu que els *words* a partir de l'etiqueta `EPROG` de la capçalera es corresponen amb les einstruccions de l'eprograma d'exemple que hem posat abans tant amb etiquetes com sense.

El vostre programa ha de ser capaç d'emular l'execució de qualsevol eprograma que s'escrigui codificat dins el vector `EPROG`.

A més del vector que conté l'eprograma i les eposicions de memòria reservades per a dades i resultats, a la capçalera es reservaran una sèrie de *words* per emular els registres de la màquina elemental, que al final de l'execució de cada einstrucció de l'eprograma contindran als seus 12 bits menys significatius el valor dels eregistres (la resta de bits han de ser 0). Així aquestes posicions s'anomenaran `EIR` (eregistre d'einstrucció), `EPC` (ecomptador d'eprograma), `ER0` (eregistre 0), `ER1` (eregistre 1) i `ESR` (eregistre d'status). Aquest darrer tindrà als seus 3 bits menys significatius els eflags amb l'ordre indicat (`ZNC`).

El programa emulador que heu d'escriure haurà de ser una iteració que faci les següents passes per a cada einstrucció de l'eprograma:

1. Realitzar el *fetch* de la següent einstrucció a executar.
2. Descodificar l'einstrucció.
3. Executar l'einstrucció.

Per tal de fer l'emulació correctament, a cada passa (1,2,3) els eregistres **s'han d'actualitzar amb el valor que prendrien a la màquina elemental**.

Especificació del programa que s'ha d'escriure durant aquesta sessió

Naturalment a aquesta sessió no heu d'escriure el programa emulador de la *PEPAr* complet. Ens centrarem a les dues primeres passes de la interacció que acabam d'esmentar, el *fetch* de la següent einstrucció i la descodificació de la mateixa. A més no programarem aquestes passes tal com s'haurà de fer per l'emulador definitiu, sino d'una manera simplificada.

El *fetch* de l'einstrucció hauria de consistir en tres subpasses:

1. Fer servir el valor contingut a `EPC` per accedir a la següent einstrucció.
2. Emmagatzemar l'einstrucció a l'eregistre `EIR`.
3. Incrementar l'eregistre `EPC`.

En canvi a aquesta pràctica ens conformarem amb accedir a les einstruccions una darrera l'altra com si fossin simplement components d'un vector identificat per l'etiqueta `EPROG`. En concret, el programa que s'ha d'escriure consistirà en un bucle que en cada iteració accedirà a la següent component del vector `EPROG` i procedirà a descodificar-la. Per fer la descodificació anirà analitzant els bits de l'einstrucció un darrera l'altre (començant pel més significatiu) fent servir qualsevol dels mètodes explicats al final d'aquest enunciat.

Cada vegada que s'ha analitzat el valor d'un bit es bota a una altra etiqueta i s'analitza el valor del bit següent. Al final de cada cadena de bots sabrem de quina einstrucció es tracta

i escriurem el seu identificador a la component corresponent d'un altre vector de *words*, que començarà a l'adreça `CODE` i que tindrà tant de components com `EPROG`. Aquest identificador es pot veure a la taula d'einstruccions: un 0 si es tractava de la primera einstrucció de la *PEPAr* (JMI), un 1 si es tractava de la segona (JMZ), i així successivament fins al 10 que escriuria en cas de que fos la darrera einstrucció de la llista (HALT). Comproveu que totes les einstruccions es descodifiquen correctament.

Maneig de bits

Fins ara, en els programes desenvolupats a les pràctiques anteriors, les dades manipulades sempre han estat *bytes*, *words* o *long words*. Un dels avantatges de l'ús de llenguatges assembladors és que es poden manipular directament els bits amb relativa facilitat. En aquesta pràctica haureu de realitzar un programa en el qual serà necessari conèixer el valor dels diferents bits d'una paraula.

Al 68K, aquesta informació es pot aconseguir de diverses maneres:

- Amb la instrucció `and mask,dst`: Aquesta instrucció realitza una operació AND bit a bit entre dos operands, emmagatzema el resultat i modifica els codis de condició d'acord amb ell. L'operand `dst` serà aquell on hi ha el bit (o bits) del que volem conèixer el seu valor (0 o 1), mentre que `mask` realitzarà les funcions d'allò que s'anomena una *màscara*, és a dir, un valor que presenta tots els seus bits a 0, excepte aquell (o aquells) del que volem saber el seu valor. Si el bit (o bits) a investigar és 0, el resultat serà tot 0's i el bit `Z` dels codis de condició passarà a valer 1 mentre que, si el resultat és diferent de zero, el bit `Z` valdrà 0, i significarà que el bit (o algun dels bits) investigat val 1.
- Amb la instrucció `btst n,dst`: Aquesta instrucció dóna directament accés al bit `n` de `dst` mitjançant l'operació `Z` ← `not(bit n de b)`. Noteu que el comportament d'aquesta instrucció es diferent si l'operand de destí es una posició de memòria o un registre.

A CampusExtens teniu un **codi d'exemple** de com fer servir aquestes instruccions. Mirau-vos també el detall dels modes d'adreçament permesos d'aquestes dues instruccions al document *Appendix C. Instruction set for Motorola 68000 microprocessor* que es troba disponible a CampusExtens, o bé al *help* de l'emulador del 68K.