Pràctica 5

Objectius:

a) Programació de subrutines.

b) Maneig de la pila.

c) Continuar amb l'escriptura de l'emulador de la màquina elemental.

Subrutines

En aquesta pràctica començareu a fer feina amb una estructura bàsica de la programació en assemblador: la *subrutina.* L'equivalent a les subrutines en els llenguatges d'alt nivell són els procediments i les funcions. Una subrutina és un fragment de programa al qual es bota en ser *cridada*, però de tal manera que en finalitzar la seva execució es torna a la instrucció següent a la que provocà el bot.

En el 68K la instrucció per cridar una subrutina és `jsr` (*Jump to Subroutine*) i la que implementa el retorn de la subrutina és `rts` (*Return from Subroutine*). Dins una subrutina pot haver-hi més d'un `rts` ja que pot acabar a diferents llocs depenent de les condicions.

La sintaxi completa de la instrucció de cridada és `jsr label`, on `label` és l'etiqueta del començament de la subrutina. D'una altra banda, `rts` no fa servir cap argument.

La Pila del sistema

Una *pila* (en anglès *stack*) és una forma d'organitzar dades prou important. En ella les dades es posen una darrera l'altra a mode de cua, però amb la particularitat que aquesta cua funciona com un caramull de plats: els elements es lleven en l'ordre invers a com han arribat, el darrer que s'ha introduït és el primer que es lleva. Aquest darrer element s'anomena *cap* de la pila. Tots els processadors moderns gestionen una pila en memòria, on poden guardar diferents tipus d'informació.

En el cas del 68K, la pila creix des del final de la memòria $00FF FFFF cap a la posició 0. Quan es crida una subrutina, el processador fa créixer la pila per poder guardar una adreça de memòria (4 bytes), guarda l'adreça de tornada a l'espai reservat (és a dir, l'adreça de la instrucció que s'ha d'executar quan acabi la subrutina), i després bota a la subrutina [1]. Quan es retorna d'una subrutina, el processador treu l'adreça de retorn del cap de la pila i allibera l'espai corresponent.

El registre A7 del 68K és el punter de pila (*stack pointer*); també vos podeu referir a aquest registre com SP dins de l'emulador. El seu contingut és, sempre, l'adreça del cap de la pila; quan la pila està buida pren el valor $0100 0000.

[1] Com veurem més endavant, la pila també se pot fer servir pel pas de paràmetres i resultats.

L'usuari pot introduir i treure dades de la pila (per seguretat useu sempre *words* o *long words* i mai *bytes*). El 68K no té instruccions PUSH i POP, sino que fa servir instruccions MOVE amb adreçaments específics:

```
PUSH x      MOVE.S x, -(SP)

POP x       MOVE.S (SP)+, x
```

Noteu que la definició de pila només indica com es fiquen i treuen dades d'ella, però no com s'hi ha d'accedir. No podeu introduir ni eliminar cap dada nova del mig de la pila, però podeu llegir-les o modificar-ne qualsevol.

Subrutines de llibreria

A l'hora de dissenyar una subrutina els plantejaments possibles són dos: dissenyar la subrutina per ser emprada dins un programa en particular, fent ús de dades del programa situades dins registres o en posicions específiques de memòria, és el que direm una *subrutina d'usuari*; l'altra opció dissenya la subrutina per ser utilitzada amb qualsevol programa, l'anomenarem *subrutina de llibreria.*

D'una subrutina de llibreria l'usuari sols necessita saber *que fa*, i no *com ho fa*. Evidentment, també cal conèixer com passar-hi els paràmetres necessaris i com extreure'n els resultats (l'interfície de la subrutina). Per exemple, si es disposa d'una subrutina que calcula el producte escalar de dos vectors, cal saber com indicar quins són els vectors a utilitzar i on apareixerà el resultat, mentres que el procediment usat per calcular el producte escalar és totalment irrellevant.

Si la subrutina ha de poder-se usar dins qualsevol programa, s'ha d'implementar de manera que el seu funcionament no es vegi afectat per la situació de l'entorn en el moment de l'execució, o que al manco les restriccions que imposi siguin mínimes. És a dir que una subrutina de llibreria no pot programar-se suposant que, per exemple, el registre `D3` o la posició de memòria $100A estan buits. Això no significa que no pugui usar-se qualsevol dels registres o determinades posicions de memòria, però com que s'ha de conservar el seu valor en finalitzar la subrutina, abans d'utilitzar-los s'haurà de salvar el seu contingut, de manera que es puguin recuperar abans de retornar al programa principal. Per aconseguir aquesta independència, les subrutines de llibreria, normalment, utilitzen la *pila del sistema.*

Per exemple, si durant l'execució d'una subrutina de llibreria es volen utilitzar els registres `D0, D2` i `D3`, i la subrutina necessita 4 *words* de memòria per guardar variables internes, el principi i final de la subrutina seran:

```
SUBR: MOVE.L D0,-(SP)
      MOVE.L D2,-(SP)
      MOVE.L D3,-(SP) ; salva el contingut dels registres
      SUB.L #8,SP     ; reserva 4 words
             .
             .
             .
      ADD.L #8,SP     ; allibera l'espai per variables internes
      MOVE.L (SP)+,D3
      MOVE.L (SP)+,D2
      MOVE.L (SP)+,D0 ; recupera el valor del registres
      RTS
```

En aquesta assignatura, les subrutines de llibreria també faran servir la pila per fer el pas de paràmetres, és a dir que el programa principal ficarà a la pila els paràmetres per a la subrutina i aquesta hi dipositarà posteriorment els resultats. Suposem que, continuant amb l'exemple del producte escalar, les especificacions indiquen que els paràmetres necessaris, que es passen a través de la pila, són per aquest ordre, les dues adreces d'inici dels dos vectors a multiplicar i la longitud del vector. Després de l'execució, i també segons les especificacions, en el cap de la pila s'hi haurà guardat el valor del producte escalar; aleshores, la part de programa corresponent a la cridada de la subrutina i obtenció de resultats podria ser:

```
LEA A,A0
MOVE.L A0,-(SP)     ; introdueix inici del primer vector.
LEA B,A0
MOVE.L A0,-(SP)     ; introdueix inici del segon vector.
MOVE.W #N,-(SP)     ; introdueix la longitud dels vectors.
JSR PRODESC         ; crida la subrutina.
MOVE.W (SP)+,RESULT ; recupera el resultat.
ADD.L #8,SP         ; ajusta valor del punter de pila.
```

Fixau-vos en aquest exemple que, just en acabar la subrutina, el resultat ocupa la posició que abans ocupava `N`, i que les adreces d'`A` i `B` encara es mantenen dins la pila. Després d'extreure el resultat l'*stack pointer* apunta a la posició ocupada per l'adreça de `B`. La darrera instrucció serveix per deixar el punter de pila apuntant al valor que tenia abans de la primera instrucció de l'exemple. Si no es controla, per part del programa principal, aquest desajust entre l'espai ocupat pels paràmetres d'entrada i resultats, la pila es va omplint d'informació inútil i es desbalancetja.

Com es pot observar en els exemples anteriors, el mode d'adreçament indexat, presentat a la pràctica 4, és especialment útil pel maneig de la pila. Quan s'escriu una subrutina de llibreria no es coneix l'estat inicial de la pila, però els paràmetres sempre ocuparan la mateixa posició relativa respecte el cap de la pila. Així, i usant el mode indexat, `0(SP)` sempre correspon al cap de la pila, `2(SP)` és la segona paraula de la pila, `4(SP)` la tercera i així successivament. Així serà aquest el tipus d'adreçament que farà servir la subrutina per llegir els paràmetres de la pila i per escriure-hi el resultat.

El següent programa es correspondria amb l'exemple del càlcul del producte escalar:

```
            ORG $1000
N           EQU 3
A:          DC.W 4, 3, 1
B:          DC.W 3, -2, 4
RESULT:     DS.W 1

START:
            LEA A,A0
            MOVE.L A0,-(SP)
            LEA B,A0
            MOVE.L A0,-(SP)
            MOVE.W #N,-(SP)
            JSR PRODESC
            MOVE.W (SP)+,RESULT
            ADD.L #8,SP

            MOVE.B #9,D0
            TRAP #15

PRODESC:
            MOVE.L D0,-(SP)
            MOVE.L A0,-(SP)
            MOVE.L A1,-(SP)
            SUB.L #8,SP

            MOVE.W 24(SP),D0
            MOVEA.L 26(SP),A0
            MOVEA.L 30(SP),A1

            ; calcul del producte escalar

            ADD.L #8,SP
            MOVE.L (SP)+,A1
            MOVE.L (SP)+,A0
            MOVE.L (SP)+,D0
            RTS

            END START
```

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

Durant aquesta sessió continuarem amb la escriptura del programa emulador de la màquina elemental que es va començar a la pràctica anterior. Concretament:

1. Heu de completar la implementació de la fase de *fetch*. El *fetch* de l'einstrucció hauria de consistir en tres subpasses:

   a) Fer servir el valor contingut a `EPC` per accedir a la següent einstrucció.

   b) Emmagatzemar l'einstrucció a l'eregistre `EIR`.

   c) Incrementar l'eregistre `EPC`.

   Cal notar que l'eregistre `EPC` s'ha d'incrementar d'un en un, tot i que esteim fent servir un vector de *words* del *68K* per introduir el nostre eprograma a l'emulador. El programa aturarà la seva execució quan trobi l'einstrucció HALT.

2. Heu de convertir la part del programa que fa la descodificació d'una einstrucció en una subrutina de llibreria que tindrà per a la seva primera instrucció l'etiqueta `DESCO`. El programa principal, després de fer el *fetch* de la següent einstrucció, invocarà a la subrutina amb el contingut de l'eregistre `EIR`. La subrutina tornarà el codi numèric entre 0 i 10 que es va indicar a la pràctica anterior i el programa principal l'escriurà a la component del vector `CODE` que correspongui.

   El pas de paràmetres concret s'ha de fer de la següent manera. En primer lloc, el programa principal reservarà un *word* a la pila perquè la subrutina pugui escriure el seu resultat posteriorment. A continuació el programa principal escriurà a la pila el codi de l'einstrucció contingut a `EIR` i invocarà la subrutina.

   La subrutina no haurà de fer servir cap altra informació externa que el paràmetre que se li passa. En cas de necessitar variables per fer càlculs (si no basten els registres) la pròpia subrutina haurà de reservar les posicions necessàries a la pila. No ha d'accedir a la memòria, només a la pila.

   Recordeu que, pel fet de ser de llibreria, la subrutina salvarà a la pila els registres que hagi de fer servir. Després d'escriure el resultat a la posició reservada a la pila, la subrutina recuperarà el valor dels registres salvats a la pila i retornarà al programa principal. Allà s'eliminarà de la pila el paràmetre passat a la subrutina (incrementant el valor de l'SP) i a continuació es recuperarà de la pila el resultat retornat per la subrutina. Com podeu veure és el programa principal qui s'ha d'encarregar de deixar l'*stack pointer* amb el mateix valor que abans de botar a la subrutina.

   No oblideu tornar a comprovar que totes les einstruccions es descodifiquen correctament.