Quadern de laboratori
Estructura de computadors 1

Javier Alonso
Carlos Álvarez
Pere Barlet
Montse Fernández
Daniel Jiménez
Joan Manuel Parcerisa
Rubèn Tous
Jordi Tubella

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Gener 2008

Licencia Creative Commons

Sessió 0: Introducció

Objectiu: En aquesta sessió coneixereu quin és l'entorn de treball a les sessions de laboratori. Editareu un programa senzill en assemblador, generareu un programa equivalent en llenguatge màquina utilitzant un assemblador, i utilitzareu un simulador per executar-lo i/o depurar-lo.

Lectura prèvia

Desenvolupament de software

Els passos habituals per fer un programa en assemblador són els següents: primer es crea el programa escrivint-lo en aquest llenguatge assemblador per mitjà d'un editor de programes. El resultat és un *fitxer font* en un llenguatge que pot entendre l'usuari, però no la màquina. Per traduïr-lo a llenguatge màquina es fa servir un programa traductor, també anomenat *assemblador*. Aquest genera un *fitxer objecte* amb la traducció del vostre programa. Molts programes estan formats per múltiples fitxers font que es poden traduir per separat en múltiples fitxers objecte. Per tal de crear el *fitxer executable* amb el programa final cal encara un darrer pas. S'anomena enllaçat o muntatge (link) i es realitza amb un programa enllaçador o *muntador* (linker).

Durant el procés de creació d'un programa se solen produir errors. Hi ha dos tipus d'errors: els sintàctics o detectables en temps de traducció, i els semàntics o detectables en temps d'execució. Els errors sintàctics són, per exemple, escriure malament una instrucció o fer una operació entre dos tipus de dades incompatibles. Aquests errors són detectats pel traductor i els hem de solucionar abans de poder generar un executable.

Un cop tenim un programa sintàcticament correcte el podem executar, però això no implica que el programa sigui correcte. Totes les instruccions poden ser correctes, però ens podem haver oblidat de posar la condició de final d'un bucle (i que aquest no acabi mai) o que, senzillament, el programa no faci el que nosaltres volem. Aquests errors només es poden detectar en temps d'execució, i per tal d'eliminar-los fem servir un *depurador* de programes (debugger). Els depuradors ens permeten executar el programa instrucció a instrucció i veure tots els valors que es van calculant, de manera que podem trobar els errors.

L'entorn de desenvolupament dels nostres programes serà sota **Linux**. Farem servir un editor qualsevol de Linux (vi, gedit, xedit, kwrite, etc.). L'assemblador s'anomena **sisa-as**, el muntador s'anomena **sisa-ld**, i el programa simulador que permet l'execució i/o depuració dels executables es diu **sisa-dbg**.

Enunciats de la sessió

Activitat 0.A: Començament

Les pràctiques de l'assignatura es poden fer en els laboratoris del DAC, al mòdul D6 o bé en els de la FIB, als mòduls A5, B5 o C6. Quan es posen en marxa surt un menú que permet triar el sistema operatiu que volem posar en marxa. Nosaltres hem de triar **`Linux`**.

Cada vegada que s'engega el PC s'ha de carregar la imatge del sistema operatiu triat. Aquesta operació és costosa: pot trigar de l'ordre de 3 minuts a baixar la imatge, descomprimir els fitxers i posar en marxa el sistema operatiu. Només ens podrem estalviar aquesta operació quan l'últim usuari que hagi estat en el mateix PC, hagi utilitzat Linux, i hagi sortit correctament amb la comanda **`logout`**. Quan se surt amb aquesta comanda, llavors el següent usuari estarà en disposició d'introduir el login directament, sense carregar la imatge del sistema operatiu. La identificació en el sistema és diferent segons l'aula on es treballi:

A les **aules del DAC** heu d'entrar al sistema amb el nom d'usuari **`alumne`**, i amb clau d'accés **`alumne`**. Per fer les sessions normals en tenim prou amb aquest compte. Per fer la sessió d'examen entrarem en uns comptes especials que s'explicaran en el mateix moment de l'examen. Un cop dins el sistema cal saber que teniu accessible el disc dur local del PC en què treballeu per guardar-hi els fitxers de treball. Aquesta zona de disc pertany a l'usuari genèric "alumne", hi té accés qualsevol, i s'esborra cada cop que es reinicialitza. Per tant, la manera de conservar els fitxers creats d'una sessió per una altra és endur-se'ls en una clau de memòria USB. Sols cal insertar-la i en pocs instants estarà visible el seu contingut. Des del navegador, o bé des d'una finestra de terminal.

A les **aules de la FIB** heu d'entrar al sistema amb el vostre nom d'usuari i password personal. Un cop engegat el sistema, tindreu accés a la vostra zona de disc personal al servidor, on anireu guardant els vostres fitxers de treball durant el curs. També en aquest cas podem llegir i copiar fitxers en una clau USB. El procediment requereix insertar la clau, clicar la icona dels "dispositius USB", seleccionar la memòria USB, i seleccionar "Muntar dispositiu". Abans de retirar la clau és convenient seguir el mateix procés, però seleccionant "Desmuntar dispositiu".

En aquest curs usarem la "finestra de terminal", un tipus de programa que ens ofereix una interfície de comandes per activar altres programes del sistema i gestionar fitxers. S'activa des d'una opció del menú, o simplement clicant la icona de terminal a la barra d'aplicacions.

Un cop oberta la finestra de terminal, podem **l**li**s**tar els fitxers del nostre directori principal, i observar-ne la grandària, les dates de la darrera modificació i altres informacions:

```
$ ls -l
```

També podem crear (**m**a**k**e) o esborrar (**r**e**m**ove) subdirectoris:

```
$ mkdir nomcarpeta
$ rmdir nomcarpeta
```

Podem **c**anviar de **d**irectori de treball, a un subdirectori, esbrinar quin és el seu camí complet des de l'arrel del disc (**p**rint **w**orking **d**irectory), i retornar al directori 'pare' inicial:

```
$ cd nomcarpeta
$ pwd
$ cd ..
$ pwd
```

Si hem insertat la clau de memòria seguint les indicacions de la pàgina anterior (depèn de l'aula), podem examinar els fitxers de la clau de memòria i **c**o**p**iar-hi qualsevol fitxer del nostre directori. A les aules de la FIB les comandes serien:

```
$ ls -l /media/usb
$ cp fitxer_a_salvar /media/usb
$ ls -l /media/usb
```

Al DAC, el directori de la clau de memòria es diu: **`'/media/USB MEMORY'`**

A través de la xarxa tindreu accés a un directori anomenat **`/assig/ec1`**. Aquest directori només és de lectura i no hi podeu escriure. En aquest directori hi haurà el software d'assemblatge i execució de programes escrits en SISA-F, així com els fitxers-plantilla de cada sessió. Al principi de cada sessió, inclosa aquesta sessió 0, heu de copiar els fitxers corresponents al vostre espai de treball. Executeu la següent comanda (atenció! el segon argument de la comanda **cp** és un punt):

```
cp  /assig/ec1/sessio0/* .
```

Si ara mireu el contingut del vostre directori de treball amb la comanda

```
ls -l
```

veureu el fitxer **`s0.s`**. Aquest fitxer conté el programa assemblador que farem servir posteriorment (Figura 1). A partir d'ara veurem el procés de desenvolupament d'un programa en llenguatge assemblador SISA-F.

Activitat 0.B: Edició de fitxers

L'edició de fitxers la farem amb un editor qualsevol que ens permeti generar text pla, ja sigui en mode text (`vi`, `joe`) o gràfic (`gedit`, `kwrite`, `xedit`, etc). Exemples:

```
$ gedit s0.s &
$ vi s0.s
```

Si activem un editor gràfic com en el primer cas, el signe **&** al final de la línia de comandes evitarà que l'editor bloquegi la finestra de comandes (el mateix efecte s'obtindria activant l'editor des de un menú o bé clicant la seva icona). En el segon cas, l'editor **vi** ocuparà la finestra fins que en sortim, però podem obrir altres finestres de comandes simultàniament.

En aquest cas, ja us donem el programa creat i en sortirem sense gravar:

```
.include "macros.s"
N = 10
.data
RES:        .word  0
V:          .word  -1, -2, -3, -4, -5, -6, -7, -8, -9, -10

.text
main:
     MOVI   R0, 0                ; Posem R0 a 0 (no sempre ha de valer 0)
     MOVI   R1, 0                ; Posem R1 a 0 (comptador del bucle)
     MOVI   R2, N*2              ; Límit del comptatge
     MOVI   R3, 0                ; Suma parcial
     $MOVEI R4, V                ; Adreça inicial de V
bucle:
     CMPLT  R5, R1, R2
     BZ     R5, fibucle          ; Després de 10 voltes acaba
     ADD    R5, R4, R1
     LD     R6, 0(R5)            ; Llegim un element
     ADD    R3, R3, R6           ; Acumulem
     ADDI   R1, R1, 2            ; Ara cada element ocupa 2 bytes !
     BZ     R0, bucle
fibucle:
     $MOVEI R4, RES              ; adreça de RES
     ST     0(R4), R3
     HALT                        ; Serveix per acabar el programa
```

Figura 0.1: Programa `s0.s`, que suma els elements de V, i ho guarda a la variable `RES`.

Nota: per utilitzar els programes d'EC1 als laboratoris de la FIB, editeu abans el fitxer `.tcshrc` i afegiu-hi al final la següent línia: `setenv PATH /assig/ec1/bin:$PATH`
A continuació, tanqueu la finestra de terminal, i obriu-ne una de nova.

Activitat 0.C: Assemblatge, muntatge i depuració

La generació de codi màquina a partir del codi assemblador la farem amb el programa assem-

blador executant la següent comanda:

`sisa-as --gstabs+ -o s0.o s0.s`

Aquesta comanda assemblarà el fitxer de codi font `s0.s`, i generarà un fitxer de codi objecte `s0.o` En aquesta instrucció, l'opció `--gstabs+` serveix per incloure informació de depuració, l'opció `-o s0.o` serveix per indicar quin nom ha de tenir el fitxer objecte resultant, i `s0.s` és el nom del nostre fitxer font. A continuació, cal enllaçar o muntar aquest fitxer objecte per crear el fitxer executable:

`sisa-ld -o s0 s0.o`

L'opció `-o s0` indica quin nom ha de tenir el fitxer executable resultant. Cal tenir en compte que el procés d'assemblatge pot generar errors. Sempre que ens trobem amb un error, el que haurem de fer és fixar-nos en el comentari que l'acompanya i anotar el número de línia on s'ha detectat. Llavors cal intentar corregir-lo editant novament el fitxer i tornant a assemblar el programa. Això és un procés iteratiu fins que no s'obtinguin errors.

Per exemple, editeu el fitxer `s0.s`, i canvieu la instrucció `MOVI R2,N*2` per `MOVI R8,N*2`. A continuació torneu a assemblar el programa i observeu el missatge d'error. Ara editeu-lo de nou, corregiu l'error, i torneu-lo a assemblar i muntar. El fitxer `s0` conté la traducció a llenguatge màquina del codi que s'ha mostrat anteriorment. Aquest fitxer ja es podrà carregar directament al depurador.

El depurador (*debugger*) s'anomena **`sisa-dbg`**. És convenient maximitzar la finestra de terminal abans d'invocar-lo, per tenir més informació a la vista. El podeu invocar sense arguments o posant-li com a argument el fitxer que voleu executar:

`sisa-dbg s0`

A continuació s'explicaran breument les comandes més usuals del depurador. Totes les comandes del depurador s'han d'introduir amb el teclat (no amb el ratolí!). La majoria estan disponibles en dos menús, un de general que s'activa/desactiva amb la tecla **F12**; i un altre de particular per a cada panell, que s'activa/desactiva amb la tecla **F11** (o bé amb la tecla **?**, en alguns terminals). En cada menú, que pot oferir diversos nivells de submenús, se seleccionen les opcions amb la tecla **INTRO**. O bé es pot abandonar sense seleccionar res prement la mateixa tecla F11(o bé ?)/ F12 amb què s'ha entrat. No obstant, les comandes més usades del depurador estan disponibles també amb una sola polsació (curtcircuit), ja sigui amb les tecles **F1**-**F10**, o bé amb altres tecles que anirem axplicant. Per exemple, prement la tecla **F1** apareixerà una pantalla d'**ajuda** informant-nos de totes les comandes disponibles, amb una descripció escueta. Per sortir del depurador premeu **F12->File->Exit** o bé simplement **Ctrl-X**

Activitat 0.D: Panells i Vistes del depurador

L'aparença que obtindrem inicialment per pantalla es mostra a la Figura 0.2, i està formada per

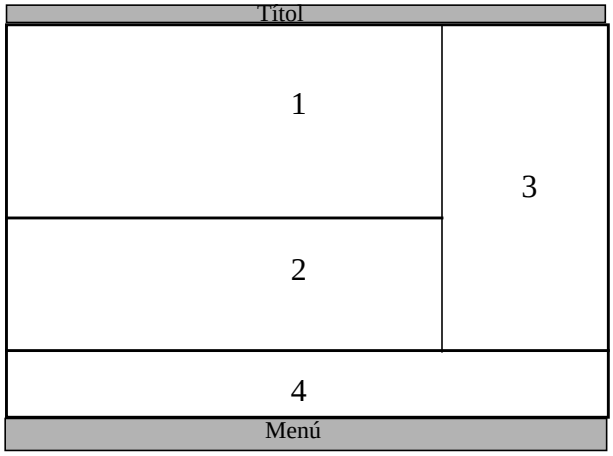


Figura 0.2: Els 4 panells

4 *panells* o finestres. Cada panell mostra una *vista* o tipus d'informació. En general, per tal d'operar canvis en un dels panells, cal enfocar-lo primer (seleccionar-lo) prement la tecla TAB repetidament fins que el seu títol queda destacat en vídeo invers.

Exercici 0.1: Podeu provar per exemple de prémer TAB un cop i seleccionar el panell 2 (Vista de Dades), i a continuació moure els cursors amunt i avall per examinar qualsevol part de la memòria.

Activitat 0.E: Vista del Codi Desassemblat i Vista del Codi Font

La Vista del Codi Font es troba inicialment al panell 3, i mostra el contingut del fitxer font a partir del qual hem generat el programa executable, amb els comentaris i les macros.

La Vista del Codi Desassemblat es troba inicialment al panell 1, i mostra una porció de la memòria desassemblada, una instrucció per línia. Cada línia està composta de tres columnes. A la primera columna apareixen símbols del programa: seccions com `<.text>` o `<.data>`, etiquetes com `<bucle>`, `<fibucle>`, constants com `<N>` (que mostra erròniament com si fos una etiqueta), i variables com `<V>` (la variable `<RES>` no apareix perquè coincideix a la mateixa adreça que la secció `<.data>`, i l'etiqueta `<main>` tampoc perquè coincideix amb `<.text>`). A la segona columna apareixen les adreces de memòria en hexadecimal. A la tercera columna hi ha el contingut de memòria en hexadecimal (el codi de cada instrucció), i a continuació la instrucció en assemblador. Per a algunes instruccions pot aparèixer al final, en forma de comentari, el valor en hexadecimal d'un dels operands (etiquetes, immediats, etc).

Exercici 0.2: Enfoqueu el panell 1 (TAB). Podeu recórrer tota la memòria polsant les tecles de cursor amunt i avall, però el que es mostri no tindrà gaire sentit en aquelles regions de memòria que continguin dades en lloc de codi.

La memòria és molt gran, i si heu perdut el punt on estàveu, podeu retornar-hi amb les tecles A o G. La tecla A (anar a posició "Actual") sitúa el cursor a l'adreça indicada pel PC. La tecla G ("Go") sitúa el cursor sobre l'adreça que li indiquem. Per exemple, polseu G, i quan us demani l'adreça escriviu: `bucle` o bé `main`.

Activitat 0.F: Vista de Dades de memòria

Es troba per defecte en el panell 2, mostra el contingut d'una porció de la memòria, en un format compacte: en cada línia de pantalla es mostra primer una adreça, i a continuació les dades emmagatzemades en aquesta i en les següents adreces de memòria, fins a omplir la línia de pantalla. Per tant, l'adreça mostrada en primer lloc correspon tan sols a la primera de les dades de la línia. El nombre de dades mostrades per línia depèn del format de visualització elegit. Al final de cada línia apareix per segon cop el contingut de les mateixes posicions, però en format ASCII (tan sols els valors que corresponen a caràcters tipogràfics, els altres apareixen en blanc).

Aquesta vista es pot mostrar en els següents formats:

- ASCII. Mostra byte per byte, sense espais entremig, en format de caràcters.
- Hexadecimal, en mida byte: els valors de cada byte en hexadecimal
- Hexadecimal, en mida word: una sola dada per a cada parell de posicions de memòria. Recordeu que els words s'emmagatzemen en format little endian, i per tant els seus dígits apareixen en ordre invers a com es veuen en el format byte (però el contingut de la memòria no canvia, és sols la manera com ens ho presenta el depurador!)
- Hexadecimal, en mida long: una sola dada en hexadecimal per a cada quatre posicions de memòria, segons el conveni little-endian.
- Decimal entera amb signe, ja sigui en mida byte o word. Per veure enters.
- Decimal entera sense signe, ja sigui en mida byte o word. Per veure naturals.
- Decimal fraccionària, ja sigui en notació exponencial o de punt fix. Per veure reals codificats en coma flotant (words).

Exercici 0.3: Enfoqueu la vista de dades (**TAB**) i premeu repetidament la tecla **F** per canviar el format de visualització per un dels 9 possibles. Proveu també amb el menú: **F11**(o bé **?**)->**Format->Dec->SignedWord**. Ara podreu observar a l'adreça 0x0022 la variable suma (val 0), i a continuació, a l'adreça 0x0024 i següents, els elements enters del vector V (valors -1, -2, etc).

Activitat 0.G: Vista dels Registres

Es troba per defecte en el panell 4. Mostra el contingut de R0..R7, de S0..S7, i del PC en hexadecimal, així com el contingut de F0..F7 en decimal, en coma fixa.

Activitat 0.H: Vista de Guaites i altres Vistes

La configuració inicial mostra la vista de Codi Desassemblat ("Disasm") al panell 1, la vista de Dades ("Data") al panell 2, la vista de Codi Font ("Source") al panell 3 i la vista de Registres ("Registers") al panell 4. Sols hi ha 4 panells, però en cada un podem posar-hi fins a 7 vistes diferents, a triar entre les ja explicades o bé una de les següents: Traça ("Trace") i Guaites ("Watches"). Per canviar la vista associada a un panell, premeu la següent opció de menú: F12->Pannels->*Núm del panell*->*Nom de la vista*.

- Traça mostra una llista ordenada de les instruccions executades.
- Guaites mostra el valor actualitzat de les variables que vulguem monitoritzar. Inicialment no conté cap variable, però es poden afegir i treure, i també podem indicar-ne el format.
- També podem tancar el panell ("Close"), per tal de fer més espai per als altres panells.

Exercici 0.4: A continuació mostrarem la vista de guaites al panell 2 i hi afegirem dues guaites per monitoritzar les variables RES i V. Seguim els següents passos:

- Associem la vista de Guaites al panell 2: F12->Pannels->2->Watch
- Enfoquem el panell 2 amb TAB
- F11(o bé ?)->Add->Format->Dec->SignedWord->1->RES
- F11(o bé ?)->Add->Format->Dec->SignedWord->10->V

Ara podrem moure el cursor amunt i avall i també a dreta i esquerra, per veure els elements de V si no hi caben al panell.

Al igual que en la vista de dades, les guaites actualitzen el seu valor a mesura que executem el programa. Per a cada guaita es mostren 3 informacions:

- A la primera columna, l'expressió introduïda per nosaltres (per exemple: `RES`). L'expressió pot contenir números, constants simbòliques i operadors en el format usual del llenguatge C. També pot contenir símbols del programa com ara etiquetes i variables, i fins i tot noms de registres.
- A la segona columna, el valor numèric equivalent de l'expressió, entre parèntesis, en hexadecimal. Quan el depurador avalua l'expressió, substitueix les etiquetes i variables per les seves adreces equivalents. Els registres se substitueixen pel seu contingut actual. El depurador interpretarà el valor final de l'expressió com una adreça de memòria.
- A continuació, el valor emmagatzemat a memòria a l'adreça calculada en la segona columna i a les adreces següents, tants elements com li hàgim indicat, en el format indicat.

Quan afegim una nova guaita, un cop introduït el format i el nombre d'elements, ens demana una expressió que determini l'adreça inicial a mostrar. Ha de ser una expressió matemàtica que pot incloure operadors (+, -, *, /, parèntesis), números, o bé símbols del programa com etiquetes o constants. Però NO ADMET indexats (p.ex. V[3]). Així, si volem que ens mostri V[3], podem posar: V+2*3

Exercici 0.5: Quina expressió hauríeu d'introduir per mostrar únicament l'element V[N-1]? Proveu-ho, afegint una guaita i comprovant que val -10. Si us equivoqueu, esborreu-la i afegiu-la de nou.

Activitat 0.I: Execució de programes

Aquesta és la funcionalitat més important d'aquest depurador. Hi ha dues maneres principals d'executar un programa: execució pas a pas o bé execució contínua amb punts d'aturada.

a) Execució Pas a pas: El PC sempre conté l'adreça de la següent instrucció a executar, la qual està marcada amb un petit signe * a la dreta de la seva adreça (punt d'execució). Prement la tecla **F5** s'executa la instrucció apuntada pel PC i s'actualitza automàticament el valor dels registres i dades de memòria en tots els panells. Si s'ha fet un accés a la memòria, la posició accedida queda ressaltada en vídeo invers per facilitar el control visual, i si s'ha modi-

ficat un registre, aquest també queda ressaltat. Aquest és el mètode més útil per a programes petits com els que nosaltres escriurem.

Exercici 0.6: Executeu pas a pas tot el programa prement F5, i observeu a la vista de dades com es ressalten els elements de V i la variable RES a mesura que són llegits o escrits. A la vista de registres podeu observar com es van modificant.

- A la vista del codi desassemblat podeu moure el cursor amunt o avall per examinar parts del codi que no apareixen en pantalla (observeu que el cursor en aquesta vista és independent del punt d'execució), però quan premeu F5 el programa executarà la següent instrucció allí on estigui el punt d'execució (el signe *), i tornarà a situar-hi el cursor al damunt. Una altra manera de ressituar el cursor damunt d'aquest punt sense executar cap instrucció és amb la tecla **A** (anar a la posició "Actual").
- **Reinicialització** dels registres (també del PC) i variables de memòria del programa: Premeu **F9** per tornar a executar el programa des del principi.
- **Recàrrega completa** del programa: Premeu **F12->File->Load->s0** per tornar a recarregar el programa des del disc. Aquesta opció és útil si heu modificat i recompilat el programa sense sortir del depurador: carregarà la nova versió.

b) Execució contínua: Prement la tecla **F4**, el programa s'executarà fins al final (o fins al primer punt d'aturada que trobi, segons veureu al següent apartat). És el mètode més útil per veure ràpidament els resultats d'un programa llarg, suposant que no té errors.

Exercici 0.7: Premeu F9 per recarregar el programa i executeu-lo fins al final amb F4. Comproveu el valor final de la variable RES (a l'adreça 0x0022 de la vista de dades, o bé en una guaita), ha de ser -55.

c) Execució contínua amb punts d'aturada: Sovint, els programes tenen errors i cal executar-los pas a pas per descobrir l'error. Però si són molt llargs, resulta pesat i és molt més pràctic executar-los de forma contínua, però aturant-nos en algun punt clau. Aquest punt rep el nom de punt d'aturada, i és una marca situada en una instrucció perquè el programa s'aturi abans d'executar-la. Per afegir un punt d'aturada sols cal enfocar la vista de Codi Desassemblat, situar el cursor sobre la instrucció en qüestió, i polsar la tecla **B** ("breakpoint") o bé **INTRO**. Apareixerà un signe + a l'esquerra de la instrucció corresponent. Amb la mateixa

tecla es pot eliminar el punt d'aturada. Es poden afegir i treure tants punts d'aturada com es vulguin o eliminar-los tots de cop amb l'opció del menú corresponent. Polsant **F4** s'executarà el programa de forma contínua fins al primer punt d'aturada que trobi. Tornant a polsar F4, el programa continua executant-se fins al següent punt d'aturada que trobi, ja sigui el mateix o un altre.

Exercici 0.8: Recarregeu de nou el programa amb F9. Enfoqueu la vista del codi i situeu el cusor sobre la instrucció següent al LD (instrucció ADD situada a l'adreça 0x0014). Polseu la tecla B per afegir-hi un punt d'aturada, i observeu el signe + a l'esquerra.

A continuació executeu fins al punt d'aturada polsant F4. Observeu l'asterisc a la dreta de l'adreça. Observeu a la vista de Registres que R5 ha estat modificat, i està ressaltat. El seu valor és `0xffff`, ja que és el primer element del vector. Si aneu polsant F4 repetidament, anireu executant iteració per iteració i podreu comprovar quins valors llegeix de la memòria la instrucció LD.

Activitat 0.J: Modificació de dades de memòria i de registres

Per poder modificar el contingut d'un registre o posició de memòria heu d'enfocar el panell corresponent (amb TAB), i a continuació situar el cursor al damunt de l'element a modificar. Teniu en compte que en la vista de dades, la dada que introduïu serà de la mida igual a la que estigueu visualitzant en aquell moment. Per tant, abans de procedir a modificar, assegureu-vos que teniu la visualització adequada. A continuació, polseu **F11**(o bé **?**)->**Modify**, o bé la tecla **M**, i introduïu l'expressió. S'accepten les mateixes expressions en llenguatge C que ja hem descrit en l'apartat anterior de guaites.

Exercici 0.9: Recarregueu el programa de nou (F9). Modifiqueu el contingut d'alguns elements del vector. Per exemple, modifiqueu el primer element del vector, posant-li valor 0x0001 (visualitzant prèviament la vista de dades en mida word, en decimal amb signe). Després, modifiqueu el darrer element del vector posant-li el valor -11.

A continuació executeu el programa fins al final (F4). Comproveu que el resultat de la suma que tenim a la variable RES (adreça 0x022) ha de ser ara -54.

Activitat 0.K: Depuració de programes erronis

El programa que us donem en s0.s ja és correcte. Tanmateix, en moltes ocasions haurem programat un codi sense cap error d'assemblatge però que després no fa la tasca esperada. Ens caldrà utilitzar el depurador per trobar l'error. Tanmateix, no hi ha un procediment universal de depuració, ja que depèn de cada cas. El més recomanable és anar executant el programa pas a pas o amb punts d'aturada, i comprovant que cada pas fa el que s'espera que faci, per detectar el punt on comença a fallar. Descobrir els *bugs* és tot un art.

Exercici 0.10: Introduirem una errada en el programa i el depurarem.

- Recarregueu el programa (F9). Enfoqueu la vista de dades visualitzant-la en mida word i modifiqueu la posició de memòria 0x000E posant-hi el word 0x6b05.
- Executeu tot el programa fins al final (F4). Quin valor té RES (adreça 0x0022)?
- Recarregueu el programa de nou (F9) i executeu-lo pas a pas (F5). Quin comportament estrany observeu? Quin és exactament l'error que hem introduït?

Sessió 1: Naturals, enters i reals

Objectiu: En aquesta sessió programarem petits codis que treballin amb números naturals, enters i reals. Farem la representació d'alguns números en els respectius formats i utilitzarem les operacions aritmètiques i de comparació per traduir sentències d'assignació i condicionals (if-then-else).

Lectura prèvia

Números naturals

Els números naturals es codifiquen en el sistema posicional binari, on el valor numèric N d'una tira d'n bits que codifica un número natural ve determinat per l'expressió:

$$N = \sum_{i=0}^{n-1} b_i \bullet 2^i$$

El processador SISP-F té suport per executar les operacions aritmètiques de suma, resta, multiplicació i divisió de números naturals ADD, SUB, MULU i DIVU; així com la multiplicació en doble precisió MULHU. El processador també té tres instruccions específiques de comparació entre naturals: CMPLTU, CMPLEU i CMPEQ. Les comparacions x>y i x>=y es poden programar a partir de les anteriors invertint l'ordre dels operands, però per simplificar la programació hem definit macros per a totes cinc comparacions: $CMPLTU, $CMPLEU, $CMPGTU, $CMPGEU, $CMPEQ. No hi ha suport explícit per saber si hi ha sobreeiximent en el resultat d'una operació aritmètica, ni tampoc produeix cap excepció. Per tant, si es vol detectar caldrà programar els algorismes corresponents en assemblador. La divisió per zero produeix una excepció.

Números enters

La codificació dels enters que suporta el processador SISP-F és en complement a 2, i hi ha suport per realitzar les quatre operacions aritmètiques ADD, SUB, MUL, DIV així com la multiplicació en doble precisió MULH. També hi ha suport per a les comparacions CMPLT, CMPLE i CMPEQ (el repertori complet està disponible en forma de macros, com en el cas dels naturals). No hi ha suport explícit per a la detecció d'overflow en enters, ni tampoc

produeix cap excepció. Per tant, si es vol detectar caldrà programar els algorismes corresponents en assemblador. La divisió per zero produeix una excepció.

Números reals

Els números reals es representen en el format exponencial format per signe (+/-), mantissa (m), base (b) i exponent (e)

$$R = \pm m \times b^e$$

S'utilitzen 16 bits per representar un número real. El bit de més pes (R_{15}) codifica el *signe* (0, positiu; 1, negatiu). La mantissa ha d'estar normalitzada (el bit més significatiu ha de valdre 1), i amb bit ocult (el bit més significatiu no es guarda, ja que val sempre 1). La coma es considera situada a la dreta del dígit més significatiu. Per tant, es guarden els 9 bits fraccionaris de la mantissa *m*, situats a la dreta de la coma ($R_{8:0}$). La base *b* del número és 2 per defecte (no s'ha de guardar a la representació). L'exponent *e* té 6 bits ($R_{15:9}$) i es codifica en excés a 31.

El processador SISP-F té instruccions per executar les 4 operacions aritmètiques amb reals ADDF, SUBF, MULF i DIVF, així com les comparacions CMPLTF, CMPLEF i CMPEQF (la resta de comparacions es completa amb macros). Tant els overflows com la divisió per zero són detectats pel processador i produeixen les respectives excepcions. Els underflows produeixen com a resultat un zero (zero positiu o negatiu segons el signe del resultat) però no són detectats explícitament pel processador. Degut a l'existència de dos codificacions per al zero, la comparació per igualtat entre reals s'ha de fer sempre amb CMPEQF, ja que no és sempre vàlid el resultat que s'obtindria amb la instrucció CMPEQ.

Activitat 1.A: Resta d'enters en doble precisió

El programa que hi ha al fitxer **`s1a.s`** realitza la resta en doble precisió de dos números enters A1 i A2, deixant el resultat a la variable R. En aquesta activitat heu de comprovar el seu funcionament:

```
     .include "macros.s"
     .include "crt0.s"
     .data
A1:  .long  3
A2:  .long  -2
R:   .long  0

     .text
main:
     $MOVEI R5, A1
     $MOVEI R6, A2
     LD     R1, 0(R5)
     LD     R2, 0(R6)
     SUB    R0, R1, R2          ; restem els bits baixos
     $CMPLTUR3, R1, R2          ; calculem el borrow
     LD     R1, 2(R5)
     LD     R2, 2(R6)
     SUB    R4, R1, R2          ; restem els bits alts
     SUB    R4, R4, R3          ; restem el borrow
     $MOVEI R5, R
     ST     0(R5), R0           ; guardem la part baixa
     ST     2(R5), R4           ; guardem la part alta
     HALT
```

Figura 1.1: Programa del fitxer `s1a.s`

Exercici:

- Escriviu en paper, per a cada instrucció i cada macro del programa, quin és el seu resultat (contingut dels registres i de les posicions de memòria que modifica).
- Assembleu i munteu el programa. Carregueu-lo al depurador. Observeu que al panell dret apareix `[Source: crt0.s]` i el codi del fitxer `crt0.s` en comptes del nostre fitxer s1a.s. Això és degut a la directiva `.include` que hem escrit al principi del programa. No hi ha problema, executeu "pas a pas" (amb F5) les 9 instruccions inicials. Veureu que s'executa un salt a l'etiqueta `main`, i llavors el panell dret ja mostra el codi del vostre fitxer `[Source: s1a.s]`. Seguiu executant pas a pas (F5), i comproveu a cada pas els valors que havíeu calculat al punt anterior.

- Localitzeu en memòria l'adreça de la variable R. Feu-ho de 2 maneres: a la vista de Dades (comanda "Go", adreça R) o a la vista de Guaites (afegint una guaita per a R). Comproveu que el seu valor final és 5 (0x00000005). És a dir, si ho visualitzeu en mida byte, els 4 bytes a partir de l'adreça de R han de ser: 0x05, 0x00, 0x00, i 0x00.

A continuació comprovarem que el mateix programa també resta correctament nombres naturals, en comptes d'enters. Sols modificarem els enters A1 i A2 pels següents nombres naturals de 32 bits: 65538 i 65535, respectivament:

Exercici:

- Editeu el fitxer `s1a.s`: modifiqueu el valor inicial de les variables A1 i A2: A1=65538 (0x00010002) i A2=65535 (0x0000FFFF).
- Escriviu en paper, per a cada instrucció i macro del programa, quin és el seu resultat (contingut dels registres i de les posicions de memòria que modifica).
- Assembleu i munteu el programa. Carregueu-lo al depurador, i executeu-lo "pas a pas" (amb la tecla F5), comprovant a cada pas els valors que havíeu calculat al punt anterior.
- Comproveu que el valor final de R és 3.

Activitat 1.B: Representació d'enters i reals

Exercici: Convertir enters del format decimal amb signe, a hexadecimal en complement a 2:

- Convertiu els següents números enters decimals a complement a 2 amb 16 bits. Escriviu el resultat en hexadecimal amb paper i llapis, seguint l'exemple del primer:

```
a) 1024        =  0x400
b) -2          =
c) 65536       =
d) -32768      =
e) 32768       =
f) 12345       =
```

- A continuació engegueu el depurador sense carregar-hi cap programa, per comprovar que ho heu fet bé, de la següent manera. Enfoqueu la vista de Dades, i visualitzeu-la en el format d'enters de tamany word en decimal amb signe. Introduïu cada un dels anteriors números hexadecimals en diferents posicions de memòria, prement la tecla M. Comproveu si el numero que apareix al panell de dades coincideix amb el que es volia representar, i en cas contrari trobeu-hi una explicació.

Exercici: Convertirem nombres reals del format decimal en punt fix o exponencial a hexadecimal en coma flotant SISA-F:

- Convertiu els següents números reals a coma flotant SISA-F (16 bits). Escriviu el resultat en hexadecimal, amb paper i llapis:

```
a) 1.0         =
b) -32.125     =
c) 1E2         =
d) 0           =
e) 100000      =
f) -23.4       =
```

- A continuació, comprova les teves respostes amb el simulador, modificant el contingut de la memòria i visualitzant-ho després en el format desitjat. Per introduir pel teclat números reals en base 10 cal visualitzar el panell de Dades en format de reals (F11->Format->Real->FixedPoint). I viceversa, per introduir per teclat reals codificats en hexadecimal, cal que visualitzar el panell de Dades en format hexadecimal (F11->Format->Hex->Word). Un cop introduides les dades, canvia el format de visualització i comprova que la codificació és correcta.

Activitat 1.C: Multiplicació i divisió d'enters

En aquest apartat calcularem a mà diverses operacions amb enters i després comprovarem amb el simulador si ho hem fet bé.

Exercici:

- Calculeu sobre paper el resultat de les següents operacions aritmètiques aplicades sobre números enters de 16 bits, indicant en quins casos es produeix sobreeiximent:

```
a) (+2) * (-3)            =                    Ovf?
b) (-32768) * (-1)        =                    Ovf?
c) (+1) / (0)             =                    Ovf?
d) (-1024) / (+130)       =                    Ovf?
e) (+32767) / (-32767)    =                    Ovf?
f) (+1234) * (-512)       =                    Ovf?
```

  - Per comprovar cada un dels apartats, editarem el programa del fitxer **`s1c.s`** (Figura 1.2) escrivint en la inicialització de A1 i A2 la parella de valors que volem provar. Llavors assemblarem, muntarem i executarem el programa en el depurador, executant-lo fins al final. Finalment, comprovarem el resultat de la variable M o D segons si volem comprovar el resultat d'una multiplicació o d'una divisió.

```
     .include "macros.s"
     .include "crt0.s"

     .data
A1:  .word  0
A2:  .word  0
M:   .word  0
D:   .word  0

     .text
main:
     $MOVEI R1, A1
     LD     R1, 0(R1)
     $MOVEI R2, A2
     LD     R2, 0(R2)
     MUL    R3, R1, R2
     DIV    R4, R1, R2
     $MOVEI R5, M
     ST     0(R5), R3
     $MOVEI R5, D
     ST     0(R5), R4
     HALT
```

Figura 1.2: Programa del fitxer `s1c.s`

Activitat 1.D: Ús de la pila

Quan un compilador tradueix un programa a assemblador, sol generar molts resultats temporals que ha de guardar en algun lloc. L'opció més senzilla i ràpida és guardar-los en registres. Però sovint els 8 registres de què disposa el processador són insuficients per a aquestes necessitats. En tal cas, el compilador recorre a la pila per guardar els resultats temporals. En aquesta activitat escriurem un programa que guardarà un resultat temporal a la pila, i després el recuperarà. Aquest programa fa una funció simple: intercanviar els valors dels dos registres.

Exercici:

- Editeu un programa[1] (en un fitxer nou que es digui **`s1d.s`**) que inicialitzi R1 i R2 amb els valors 3 i -4, respectivament, i que intercanvïi els valors dels registres R1 i R2 sense modificar cap altre registre (excepte R7, lògicament). L'algorisme que ha de seguir és el següent:

  1) Inicialitzar R1 i R2
  2) Apilar el registre R1 (això crea temporalment una còpia d'aquest valor a la pila)
  3) Copiar el registre R2 en R1
  4) Desapilar un word de la pila, escrivint-lo en R2 (elimina la còpia temporal)
- Assembleu i depureu el vostre programa. Verifiqueu que el contingut dels registres s'hagi intercanviat al final del programa, i que R7 hagi recuperat el seu valor inicial.
- Recarregueu el programa (F9), i abans de tornar-lo a executar, intenteu visualitzar la pila dins la vista de Dades. Executeu pas a pas novament, i observeu com es modifica la pila.

1. Quan editeu programes és recomanable que al final del codi aparegui sempre un salt de línia, del contrari el compilador us donarà un avís (warning).

Activitat 1.E: Depuració d'un codi erroni

Considereu el següent programa en C--:

```
float f=-1.0,g=-2.0;
int i=0;

main()
{
       if (f<=g) i=1;
}
```

El fitxer **`s1e.s`** conté el programa de la figura 1.3, que pretén ser la traducció del programa anterior a assemblador SISA-F. Tanmateix, el codi no fa la feina indicada en alt nivell.

```
     .include "macros.s"
     .include "crt0.s"

     .data
F:   .word  0xBE00        ; 0xBE00 és la codificació de -1.0
G:   .word  0xC000        ; 0xC000 és la codificació de -2.0
I:   .word  0

     .text
main:
     MOVI   R6, 0
     $MOVEI R1, F
     LD     R1, 0(R1)
     $MOVEI R2, G
     LD     R2, 0(R2)
     CMPLE  R3, R1, R2
     BZ     R3, fi
     MOVI   R6, 1
     $MOVEI R5, I
     ST     0(R5), R6
fi:
     HALT
```

Figura 1.3: Programa del fitxer `s1e.s`

Exercici:

- Comproveu que és incorrecte, ja que per a les dades inicials, la variable I hauria d'emmagatzemar finalment el valor 0. Executeu-lo **pas a pas** diverses vegades verificant el resultat de cada instrucciño fins que descobriu quina és errònia.
- Un cop descobert l'error, editeu el programa modificant-lo perquè funcioni de forma correcta. En acabat, torneu-lo a assemblar, muntar i verificar de nou amb el depurador. Repasseu, si us cal, la Lectura Prèvia d'aquesta sessió.

Activitat 1.F: Traducció d'expressions complexes en alt nivell

En aquesta activitat haureu de traduir a assemblador SISA-F el següent programa escrit en C--:

```
int a, b, c, d, e, x;

main () {
        if (a == b) {
                x = c-d*(e+1);
        }
        else {
                x = (c-d*(e+1)+1)/(a-b);
        }
}
```

Exercici: Escriviu el vostre programa en un nou fitxer que s'anomeni s1f.s. Assembleu-lo, i munteu-lo. Heu d'escriure 2 versions del programa:

- En la primera, els valors inicials són a=0; b=0; c=1; d=2 i e=3. Verifiqueu amb el depurador que el valor final de x és -7.
- Editeu el programa de nou amb els valors inicials a=2; b=0; c=1; d=2 i e=3. En aquest cas, verifiqueu que el valor final de x és -3.

Sessió 2: Tipus de dades estructurats

Objectiu: En aquesta sessió es posarà en pràctica l'accés a variables de tipus estructurats. És a dir, a vectors, matrius i tuples.

Lectura prèvia

Punters

Un punter és una variable que ocupa 16 bits i conté l'adreça de memòria d'una altra variable. Es declara en assemblador SISA-F

```
punter: .word 0
```

i es pot inicialitzar en la pròpia declaració:

```
punter: .word nom_variable_apuntada
```

o en el codi:

```
$MOVEI R1, nom_variable_apuntada
$MOVEI R2, punter
ST     0(R2), R1
```

Vectors

Un vector és un conjunt unidimensional d'elements del mateix tipus, i s'emmagatzemen en memòria consecutivament a partir d'una adreça inicial. Cada element s'identifica amb un índex, i en llenguatge C aquest es numera començant sempre des de 0 per al primer element fins a N-1 per a l'últim (on N és el nombre d'elements del vector). Per accedir a un element d'un vector, l'índex indica quants elements s'han de saltar des de l'adreça inicial per trobar l'element que es busca:

```
@v[i] = @v + (i * mida_d'un_element)
```

En SISA-F es pot declarar un vector inicialitzat de la següent manera:

```
nomvector:    directiva valor_element_0, valor_element_1, ...
```

on la directiva d'assemblador pot ser `.byte`, `.word` o `.long` segons la mida dels elements. Però si el vector té un nombre gran d'elements i tots ells s'inicialitzen amb el mateix valor, llavors podem usar la directiva `.fill`

```
nomvector:    .fill  num_elements, bytes_per_element, valor_elements
```

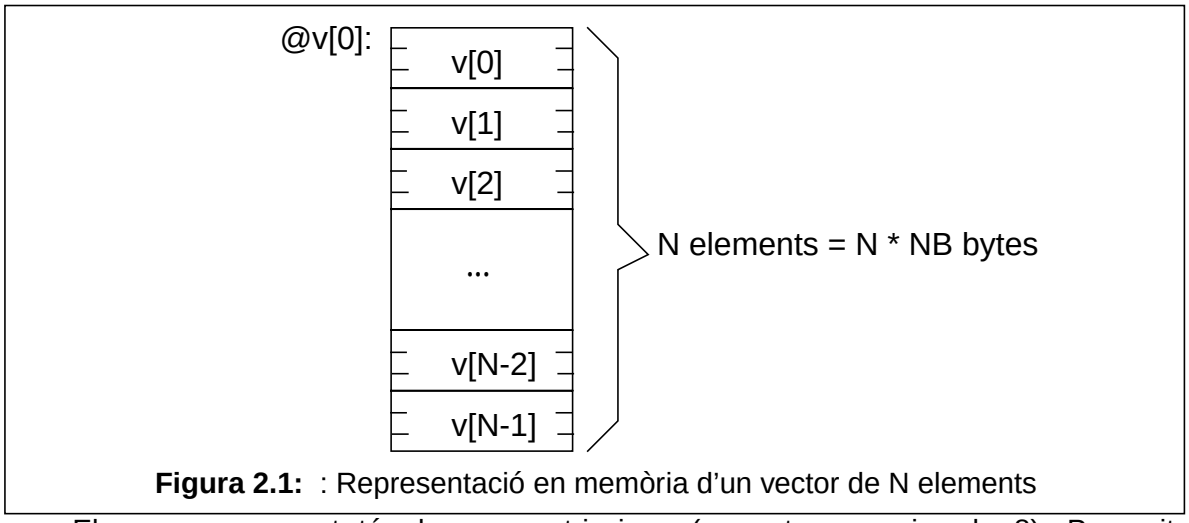


Figura 2.1: : Representació en memòria d'un vector de N elements

El segon argument té algunes restriccions (no pot ser major de 8). Per evitar problemes, com que sovint s'inicialitzen tots els elements a zero, a la pràctica aquesta directiva s'usa així:

```
nomvector:    .fill num_elements*bytes_per_element, 1, 0
```

Matrius

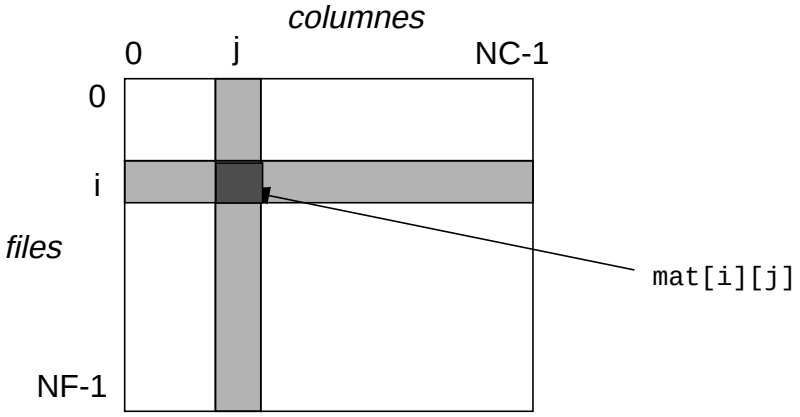
Una matriu és un conjunt bidimensional de NFxNC elements d'un mateix tipus, on NF és el nombre de files i NC el de columnes. La declaració en C d'una matriu d'enters és:

```
int mat[NF][NC];                    /* NF i NC han de ser constants */
```

i hi podem inicialitzar els elements, com per exemple:

```
int mat[2][3] = {{-1, 2, 4}, {0, 5, -2}};
```

Cada element s'identifica amb dos índexos, el primer és la fila, el segon la columna:



Aquests elements s'emmagatzemen en memòria en posicions consecutives per files, i en cada fila ordenats per columnes (vegeu figura 2.1). En SISA-F no hi ha una directiva específica per a declarar una matriu, però es pot fer declarant-la com un vector de NFxNC elements, ordenats per files:

```
nommatriu:    directiva valor_element_0, valor_element_1, ...
```

o bé:

```
nommatriu:    .fill num_elements, mida_element, valor_elements
```

Exemples: M1, M2 són matrius de caràcters, i M3, M4 són matrius d'enters (NF=3, NC=4):

```
M1:    .byte  'F, 'I, 'L, 'A, 'f, 'i, 'l, 'a, 'F, 'I, 'L, 'A
M1:    .byte  'F, 'I, 'L, 'A
       .byte  'f, 'i, 'l, 'a
       .byte  'F, 'I, 'L, 'A              ; per visualitzar les files
M2:    .fill  NF*NC, 1, 0                 ; tos zeros

M3:    .word  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
M3:    .word  0, 1, 2, 3
       .word  4, 5, 6, 7
       .word  8, 9, 10, 11                ; per visualitzar les files
M4:    .fill  NF*NC, 2, 0                 ; tots zeros
```

Suposant que `mat` és una matriu de `NF` files i `NC` columnes d'elements de mida `NB` bytes, per accedir a un element qualsevol `mat[i][j]` ho farem saltant `i` files completes, i després `j` elements, a partir de l'adreça inicial `mat (figura 2.2)`. L'adreça es calcularà de la següent manera (NB = núm bytes per element):

```
@mat[i][j]  =   @mat + (i*NC + j)*NB   =   @mat + i*NC*NB + j*NB
```

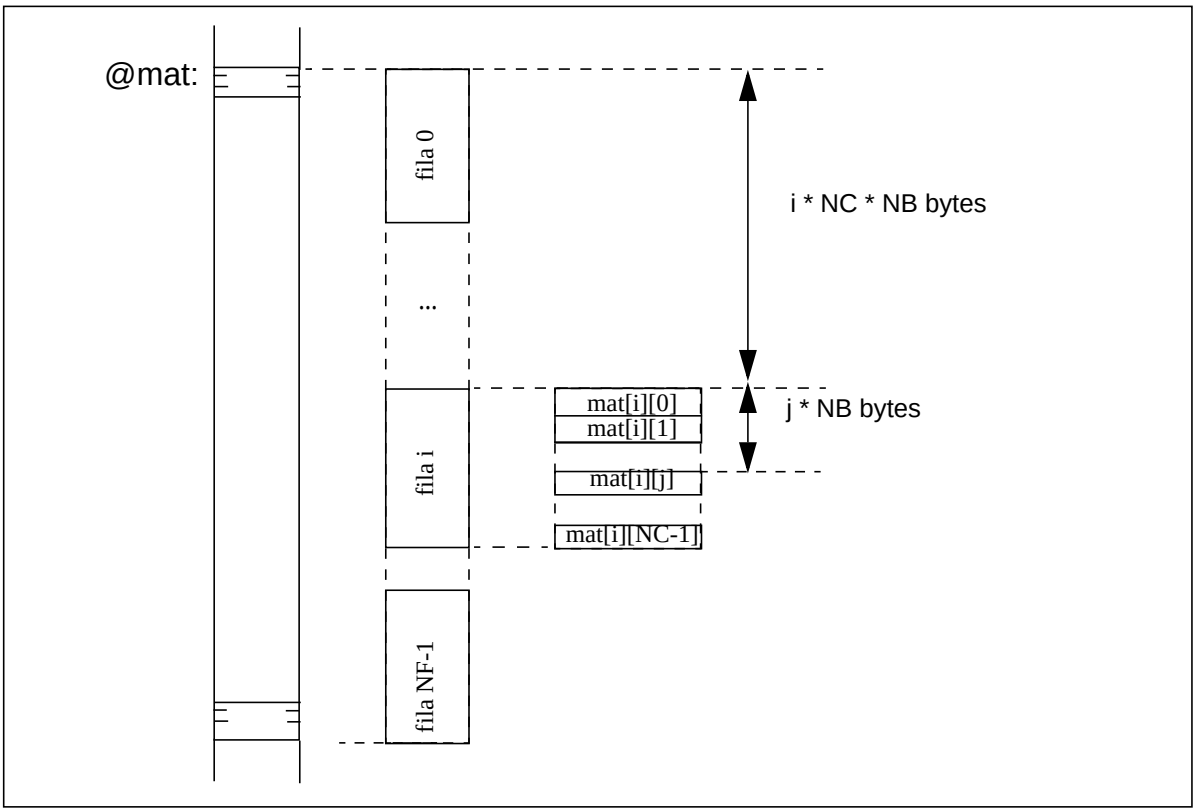


Figura 2.2: : Emmagatzematge de matrius per files, i accés a l'element `mat[i][j]`

Tuples

Una tupla és una agrupació d'elements de tipus heterogenis, a diferència dels vectors i matrius, que agrupen elements del mateix tipus. Cada element rep el nom de *camp*. Els camps s'emmagatzemen en memòria en posicions consecutives a partir de l'adreça inicial de la tupla. Tots els camps s'emmagatzemen en adreces parelles. Per tant, si la mida d'un camp és senar, llavors cal deixar un byte buit entre aquest camp i el següent per tal que aquest ocupi una adreça parella. En alt nivell es declara primer el tipus i a continuació les variables:

```
struct nomdetipus {          /* Definim el tipus struct nomdetipus */
       tipus1 camp1;
       tipus2 camp2;
       ...
};
struct nomdetipus var1, var2;/* Declarem les variables var1, var2 */
```

Per declarar una tupla en SISA-F, no tenim una directiva específica sinó que reservarem tants bytes com ocupi la tupla, amb la següent declaració:

```
var1:  .fill  numbytes, 1, 0
       .balign 2
```

o bé, distingint els camps individuals:

```
var1:  .word  0      ; camp1, ocupa 2 bytes
       .fill  3      ; camp2, ocupa 3 bytes (p.ex. vector de caràcters)
       .balign 2     ; alineem el camp 3
       .byte  0      ; camp3, ocupa 1 byte
       .balign 2     ; alineem la següent variable
```

Sigui quina sigui la declaració emprada, no oblideu alinear novament l'adreça per a les següents declaracions, si la tupla té mida senar (observeu l'alineació al final, en els 2 casos).

Per accedir a un camp d'una tupla en calcularem l'adreça sumant a l'adreça inicial de la tupla el nombre de bytes que hi ha fins al camp que busquem. Per exemple, si considerem les següents declaracions:

```
struct exemple {
       int a;
       int b;
       int c;
};
struct exemple var;
```

llavors el camp `c` està situat a 4 bytes de l'adreça inicial de la tupla, i la sentència en C--

```
var.c = 0;
```

es podria traduir a SISA-F de la següent manera:

```
$MOVEI        R1, var+4
MOVI          R0, 0
ST            0(R1), R0
```

Activitat 2.A: Accés indirecte a una variable a través d'un punter

Sigui el següent programa escrit en C:

```
int dada;
int *pdada = &dada;

void main()
{
        *pdada = *pdada + 1;
}
```

Figura 2.3: Programa que modifica una variable de forma indirecta a través d'un punter

Completeu el següent exercici:

Exercici 2.1: Traduïu a assemblador el programa de la figura 2.3. Teniu en compte que la inicialització del punter `pdada` s'ha de fer en la declaració, i no pas en el codi.

```
.data



.text
main:
```

A continuació, copieu el codi de l'anterior exercici a l'arxiu **s2a.s**.

Verifiqueu el correcte funcionament del programa, obrint el panell de guaites i afegint una guaita per a la variable `dada` (`F11->Add->Format->Dec->SignedWord->1->dada`). Comproveu que al finalitzar el programa pren el valor `dada = 6`.

Activitat 2.B: Accés aleatori als elements d'un vector

2.B.1) Suposem un vector global de N elements de tipus enter declarat així: `int vec[N];`
Completeu el següent exercici:

Exercici 2.2: : Escriviu la fórmula per al càlcul de l'adreça de l'element `vec[i]`, en funció de l'adreça inicial de `vec` i del valor `i`:

```
@vec[i] =
```

A partir de la fórmula anterior, escriviu un fragment de codi en assemblador tal que copïi en el registre R1 el valor de `vec[i]`, és a dir: `R1<-vec[i]`.

```
main:
```

Copieu el programa anterior a l'arxiu **s2b1.s**, i verifiqueu-lo comprovant a la Vista de Registres que el resultat final de R1 és `R1 = 0x0004`.

2.B.2) Sigui el següent programa en C: donat un número natural `num`, aquest programa obté, d'un en un i començant pel de menys pes, els dígits de la seva representació en decimal mitjançant divisions successives i emmagatzema aquests dígits en el vector de naturals `vec`:

```
#define N 5
unsigned int num;
unsigned int vec[N];

void main()
{
     register int i = 0;

     while (num != 0) {
            vec[i] = num % 10;
            num = num / 10;
            i++;
     }
}
```

Figura 2.4: Programa que calcula la representació decimal d'un número natural

Completa el següent exercici:

Exercici 2.3: : Tradueix a SISA-F el codi del programa de la figura `2.4`. Recorda que SISA-F no té cap instrucció que calculi el residu de la divisió (%) i que per obtenir-lo s'ha de calcular mitjançant la següent fórmula: residu = dividend - (divisor * quocient).

```
main:
```

A continuació, copia el codi de l'anterior exercici a l'arxiu **s2b2.s**. Verifica'l afegint una guaita per a `vec` (`F11->Format->Dec->UnsignedWord->5->vec`) i comprovant que al final del programa val `vec={5 4 3 2 1 }`. Comprova també que al final `num` valgui `0`.

Activitat 2.C: Cadenes de caràcters (strings)

Sigui el vector de naturals `vec`, que conté els dígits (números del 0 al 9) de la representació en decimal d'un número natural tal i com s'ha descrit a l'activitat 2.B.2. El següent programa en C escriu en l'*string* `cadena` els caràcters que representen cada un dels dígits de `vec`.

```
#define N 5
char cadena[N+1];
unsigned int vec[N];

void main()
{
     register int i;

     for (i=0; i<N; i++) cadena[i] = (char)vec[N-i-1] + '0';

     cadena[N]=0;          /* posa la marca de final de string! */
}
```

Figura 2.5: Programa que converteix a ASCII una llista de dígits decimals

Observeu que el programa de l'activitat 2.B.2 (figura 2.4) escriu al vector `vec` els dígits decimals de `num` començant pel de menor pes. Però com que l'*string* `cadena` és un text i

volem que es pugui llegir d'esquerra a dreta, el primer caràcter de la cadena ha de representar el dígit de major pes, és a dir en ordre invers. Per aquesta raó, en el programa de la figura 2.5, a cada iteració del bucle es converteix el dígit `vec[N-i-1]` en comptes de convertir el dígit `vec[i]`. La conversió a ASCII es fa sumant 48 al dígit en decimal, o el que és el mateix, el codi ASCII de '0'. Fixeu-vos també que quan es declara el vector de caràcters `cadena` es reserva espai per a `N+1` elements, per tal de poder guardar el valor *sentinella* 0, que senyala el final de l'*string*. Completeu el següent exercici:

Exercici 2.4: : Traduïu a assemblador SISA-F el codi del programa de la figura `2.5`.

```
main:
```

Després copieu el codi de l'anterior exercici a l'arxiu **s2c.s**. Verifiqueu el programa afegint una guaita per visualitzar el vector de caràcters `cadena` (`F11->Add->Format->Ascii->6->cadena`) i comprovant que al final del programa val `cadena = "12345"`.

Activitat 2.D: Accés aleatori a una matriu de tuples

Siguin les següents declaracions de variables globals d'un programa en C, on `var` és una variable de tipus `tupla` i `mat` és una matriu de tuples de tipus `tupla`

```
#define N 3
#define M 4
struct tupla {
     int    a;
     char   b;
     int    c;
};
struct tupla var;
struct tupla mat[N][M];
int i, j;
```

Figura 2.6: Declaracions de variables globals

Completeu el següent exercici:

Exercici 2.5:

1) Escriviu la secció `.data` corresponent a la traducció a SISA-F de la declaració anterior:

```
N = 3
M = 4
.data
```

2) Escriviu la fórmula de l'adreça de cada un dels camps de `var`, en funció de l'adreça de `var`:

```
@var.a =
@var.b =
@var.c =
```

3) Escriviu la fórmula per al càlcul de l'adreça de `mat[i][3].a` i l'adreça de `mat[2][j].c`, en funció de l'adreça inicial de `mat` i els valors de `i` i `j`:

```
@mat[i][3].a =
@mat[2][j].c =
```

4) Traduïu a SISA-F la següent sentència en C, utilitzant les anteriors fórmules

```
mat[i][3].a = mat[2][j].c;
```

Intenteu reduir el nombre de línies del codi, deixant que el `sisa-as` faci les operacions on només hi intervenen constants (no depenen de les variables `i` i `j`). Per exemple, @mat[2][3] es podria calcular en una única línia: `$MOVEI R1,mat+(2*M+3)*NB` (on `NB` és la mida d'un element de tipus tupla), ja que `mat` és una etiqueta, i `M` i `NB` són constants.

A continuació, copieu el codi de l'anterior exercici (apartat 4) a l'arxiu **s2d.s**.

Verifiqueu el correcte funcionament del programa, mostrant la Vista de Dades en format SignedWord (`F11->Format->Dec->SignedWord`) i comprovant que, després d'executar el programa, l'adreça de memòria `@mat[i][3].a` conté el valor 5. Per localitzar aquesta dada a la Vista de Dades, polseu la tecla G (o bé F11->Goto->Address) i indiqueu l'adreça: `mat+66`.

Activitat 2.E: Accés seqüencial a la columna d'una matriu

Suposem la següent declaració de la matriu `mat`, de 4 files per 6 columnes, de nombres reals:

```
float mat[4][6];
```

Completeu el següent exercici:

Exercici 2.6:

1) Escriviu la fórmula per calcular l'adreça de `mat[i][2]`, en funció de l'adreça de `mat` i de `i`

```
@mat[i][2] =
```

2) Fent servir aquesta fórmula, calculeu la distància en bytes (*stride*) entre les adreces de dos elements consecutius d'una columna :

```
@mat[i+1][2] - @mat[i][2] =
```

Observeu que, donats dos elements consecutius d'una columna, l'adreça del segon element s'obté sumant una quantitat constant a l'adreça de l'element anterior. En general, la distància entre dos elements consecutius d'un vector o d'una fila d'una matriu també és constant. A aquest valor constant se l'acostuma a anomenar *stride*, i és el resultat que has calculat a l'apartat 2) de l'exercici anterior. Quan recorrem vectors o matrius utilitzant aquesta propietat diem que estem fent un "accés seqüencial" als seus elements, i seguim els següents 3 passos:

- Al principi, inicialitzar un punter (un registre) amb l'adreça del 1er element a recórrer.
- Accedir a cada element fent servir sempre aquest punter.
- Just a continuació de cada accés, incrementar el punter tants bytes com hi hagi de diferència entre un element i el següent (veure l'apartat 2 de l'exercici anterior).

Sigui el següent programa en C, que suma els elements de la columna 2 de la matriu `mat`:

```
float mat[4][6];
main() {
     register int i;
     register float suma;

     suma = 0.0;
     for (i=0; i<4; i++)
            suma += mat[i][2];
}
```

Figura 2.7: Programa que suma la columna 2 d'una matriu

Ara observeu com es converteix en el següent programa equivalent, per tal de fer accés

seqüencial:

```
float mat[4][6];
main() {
     register int i;
     register float suma;
     register float *p;

     suma = 0.0;
     p = &mat[0][2];             /* inicialitzar punter */

     for (i=0; i<4; i++) {
            suma += *p;          /* Accés a mat usant el punter */
            p += 6;              /* Incrementar el punter */
     }
}
```

Figura 2.8: Programa que suma la columna 2 d'una matriu fent accés seqüencial

Fixeu-vos en la diferència que hi ha entre el valor que heu calculat a l'apartat 2 de l'exercici 2.6 (el *stride*) i el valor amb què s'incrementa el punter `p` a la figura 2.8. Això és degut a l'aritmètica de punters en C: un increment d'1 unitat en un punter suposa sumar-li, en assemblador, la mida de l'element al qual apunta. En el nostre cas, la mida de l'element és 2 bytes, ja que `p` apunta a un element de tipus float. Per tant, el valor que apareix en el codi en C s'haurà de multiplicar per 2 en la traducció a assemblador.

Completeu el següent exercici:

Exercici 2.7: : Traduïu a assemblador SISA-F el codi del programa de la figura `2.8` (la versió que fa accés seqüencial). Fixeu-vos que tan sols `mat` està en memòria, la resta de variables es guarden en registres. Feu servir F1 per guardar la variable `suma`.

main:

A continuació, copieu el codi de l'anterior exercici a l'arxiu **s2e.s**.

Comproveu el correcte funcionament del programa verificant a la Vista de Registres que, després d'executar el programa, el registre `F1` (variable suma) val `F1=1.00000e+01`, és a dir `F1`=10.0.

Feu també la següent comprovació: enfoqueu la Vista de Dades de memòria (tecla TAB); mostreu-la en fromat "Real FixedPoint" (tecla F); reinicieu el programa (tecla F9); executeu-lo pas a pas (tecla F5); i observeu com, després de cada `LDF`, s'il·lumina l'element de la matriu accedit. Comproveu que sols s'accedeixen els elements de la columna 2 de `mat`:

```
float mat =   {{0.0, 0.0, 1.0, 0.0, 0,0, 0.0},
               {0.0, 0.0, 2.0, 0.0, 0,0, 0.0},
               {0.0, 0.0, 3.0, 0.0, 0,0, 0.0},
               {0.0, 0.0, 4.0, 0.0, 0,0, 0.0}};
```

Objectius: Entendre i saber aplicar les regles per al pas de paràmetres i resultats en les crides a subrutines.

Lectura prèvia: Subrutines

Subrutines

A continuació trobareu un resum amb els conceptes més importants que s'han introduït en aquest tema de subrutines: definicions, fases de la seva gestió, i traducció de codi en alt nivell que utilitza funcions i/o procediments.

Definicions

Subrutina: Conjunt d'instruccions de llenguatge màquina (o assemblador) que realitza una tasca parametritzable. Serveix per implementar les funcions i accions que existeixen en els llenguatges d'alt nivell.

Vegeu en la figura 3.1 els diferents elements que apareixen en un codi en alt nivell on es defineix una funció i es fa una crida a aquesta funció.

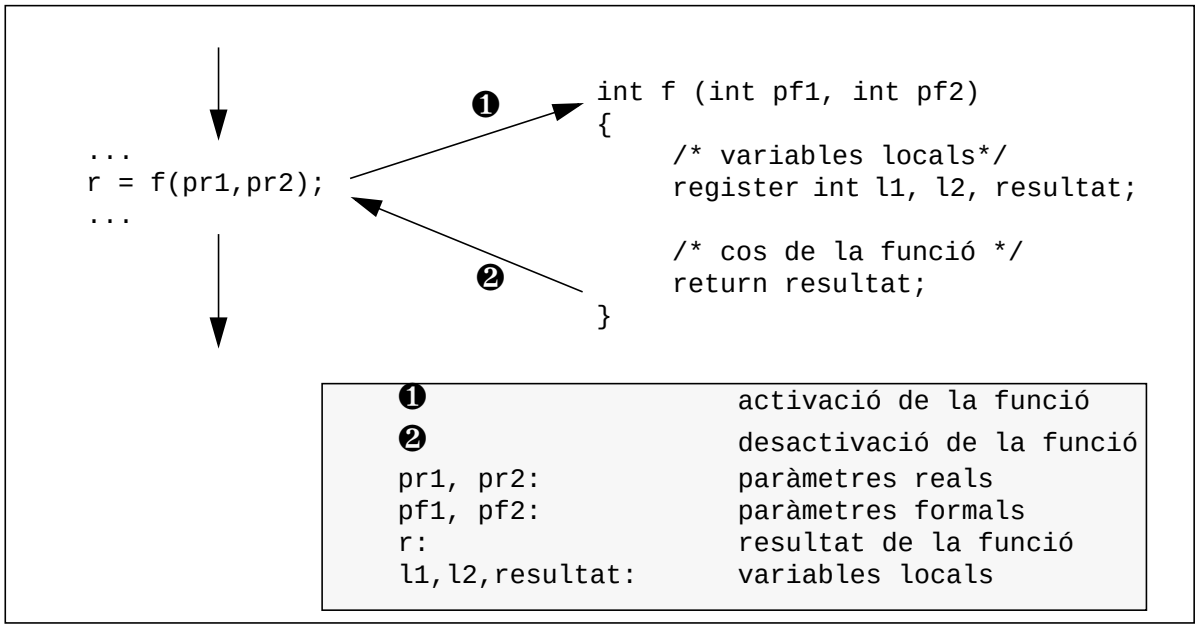


Figura 3.1: Definició i invocació d'una funció

Paràmetres reals/formals: Els paràmetres reals són els que es passen en cadascuna de les invocacions que es fan a una subrutina. Els paràmetres formals són els que s'utilitzen dins la definició d'una subrutina.

Paràmetres per valor/ per referència: Un paràmetre per valor és únicament d'entrada a una subrutina. Un paràmetre per referència pot ser d'entrada i/o sortida.

Fases de la gestió d'una subrutina

A continuació s'expliquen les regles a seguir per a una correcta interfície entre el programa que crida a una subrutina i el codi d'aquesta subrutina. Aquestes regles corresponen a la interfície de programació adoptada en l'assignatura de EC1, i apta solament per a subrutines senzilles. Es deixa per a un curs posterior l'explicació de tècniques basades en l'ús de la pila, que permeten programar subrutines més complexes. Aquestes regles estan exposades en el mateix ordre en què s'executen. Així doncs, els apartats 1, 2, 3, 6 i 7 fan referència al codi del programa que fa la crida i la resta correspon al codi de la subrutina que és cridada.

1.-Salvar l'estat: Consisteix en guardar a la pila tots els registres que volguem preservar ja que tots podran ser modificats en el cos de la subrutina (d'R0 a R6 i d'F0 a F7). Per exemple, `$PUSH R1,R2,R6` o `$PUSHF F3,F5`. En el cas de funcions, R0 o F0 s'usaran per a retornar el resultat, i per tant no s'hauràn de salvar ni restaurar.

2.-Passar els paràmetres: Consisteix en col.locar els *paràmetres reals* que es passen a un subrutina en el lloc adequat perquè la subrutina pugui utilitzar-los quan necessiti accedir als seus *paràmetres formals*. En SISA-F els paràmetres sempre es passen mitjançant registres (d'R1 a R5 i d'F1 a F7) tenint en compte que R0 o F0 es reserven per al retorn en les funcions i R6 per l'adreça de retorn. La numeració dels registres utilitzats correspondrà a l'ordre en què estan declarats els paràmetres en alt nivell, és a dir, el primer paràmetre sempre està a R1 o F1 (e.g. si tenim (`int a, float b, int c`) utilitzarem R1, F1 i R2).

Un paràmetre per valor es passa col.locant una CÒPIA del paràmetre real en el registre. Un paràmetre per referència es passa col.locant l'ADREÇA del paràmetre real en el registre. Considerarem que els tipus de dades elementals els podrem passar per valor o per referència a una subrutina, però els tipus de dades estructurats sempre es passaran per referència. Donat que les variables locals de les subrutines estaran allotjades en registres, aquestes no es podran passar per referència.

3.-Cridar a la subrutina: Guardar l'adreça de retorn al registre R6 i saltar a la primera instrucció de la subrutina. Això es fa amb la macro `$CALL R6,subrutina`, que equival a fer:

```
$MOVEI R6, subrutina
JAL R6, R6
```

4.-Executar el cos de la subrutina: Les variables locals s'allotjaran sempre en registres, i no podran ocupar ni R6 (adreça de retorn) ni R7 (punter a la pila). Si una subrutina és una funció, que retorna un tipus diferent de *void* llavors traduirem les sentències `return expressió` copiant el valor de l'expressió en R0 (o bé F0 si és del tipus float), i saltant al final del cos de la subrutina. A banda d'això, el programador disposa de tota la llibertat per gestionar els registres, podent utilitzar els registres que allotjen paràmetres per altres propòsits un cop ja no facin més falta, o fins i tot mapejar algunes variables locals sobre registres de paràmetres o sobre R0 o F0 (fins i tot en funcions).

5.-Retornar de la subrutina: Saltar a executar la instrucció següent a la de la crida a la subrutina, amb la instrucció `JMP R6`.

6.-Restaurar l'estat: Recuperar el valor que tenien els registres que s'hagin salvat abans de la crida i que està guardat a la pila, desapilant-los en l'ordre invers a com han estat apilats. Per exemple, `$POPF F5,F3` o `$POP R6,R2,R1`.

7.-Recollir el resultat: Fer ús del resultat que ha deixat la subrutina en el registre R0 o F0 si es tracta d'una funció.

Exemple de traducció d'una subrutina de C a SISA-F

A continuació repassarem quina és la traducció a llenguatge assemblador que resulta d'un codi en alt nivell que defineix una funció i d'un codi que la invoca. Vegeu aquest codi en la figura 3.2.

```
int r;
int pr1[10] = {1, 2, 3, 0, 0, 0, 0, 0, 0, 0};     // pr1 és un vector
int pr2 = 1;

int exemple(int pf1[10], int pf2) {            // pf1 és un vector
     register int l1,l2;

     l1 = pf1[0];
     l2 = pf2;
     return pf1[l1+l2];
}

main() {
     register int a=1, b=2, c=3;

     r = exemple(pr1,pr2);                     // crida
     r = r + a + b;
}
```

Figura 3.2: Exemple de codi en C que utilitza una funció

En aquest exemple veiem que es defineix una funció `exemple` que té dos paràmetres d'entrada i retorna un enter. El paràmetre formal `pf1` *és* un vector i es passa per referència.

El paràmetre formal `pf2` és un enter i es passa per valor. En el programa principal es fa una invocació de la funció on es passen com a paràmetres reals les variables globals `pr1` i `pr2`, i el resultat de la funció es guarda a la variable global `r`. A l'hora de traduir aquest codi és recomanable que tinguem clarament identificat quin és el paper que jugarà cada registre.

La figura 3.3 mostra el paper dels diferents registres en una crida a la subrutina que estem considerant en aquest exemple.

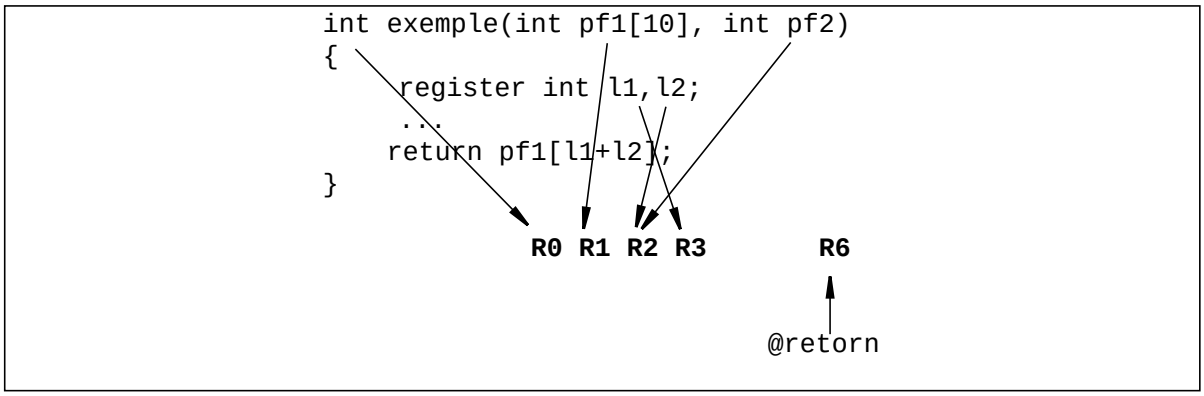


Figura 3.3: Paper dels registres durant l'execució de la subrutina `exemple`

Observem que els dos paràmetres `pf1` i `pf2` s'han mapejat als registres R1 i R2 respectivament, i que el registre R0 s'utilitzarà per retornar el resultat, ja que es tracta d'una funció. El codi que fa la crida (el programa principal) és conscient que R0 servirà per retornar el resultat, de forma que abans de la crida reservarà aquest registre (per això no l'utilitzarà per a les variables locals `a`, `b` o `c` del `main`) i no caldrà ni salvar-lo ni restaurar-lo.

La variable local `l1` es mapejarà al registre R3 i la variable local `l2` es mapejarà directament sobre el registre R2. Aquí podem observar que el programador del cos de la subrutina té llibertat per reaprofitar els registres que guarden paràmetres (en aquest cas R2), quan ja no els necessita més.

La figura 3.4 mostra la traducció literal (sense optimitzar) d'aquest codi a assemblador SISA-F. Abans de començar a fer les activitats de la pràctica, mireu com s'han implementat les diferents fases de la gestió de subrutines i com s'ha traduït literalment cadascuna de les sentències que hi ha en el cos de la subrutina exemple.

Observem que, dins el programa principal (main), els registres R1, R2 i R3 inicialment guarden el valor de les variables locals `a`, `b` i `c`. Al fer la crida, només estem obligats a salvar aquells registres que necessitem preservar per a després de la crida, i per tant en aquest cas només hem de salvar a la pila els registres R1 i R2 (que s'utilitzen a la darrera instrucció).

```
.include "macros.s"
.include "crt0.s"
.data
     r:   .word 0
     pr1: .word 1, 2, 3, 0, 0, 0, 0, 0, 0, 0
     pr2: .word 1
.text
main:
     MOVI      R1, 1             ; a = 1
     MOVI      R2, 2             ; b = 2
     MOVI      R3, 4             ; c = 3
     ;crida
     $PUSH     R1, R2            ; salvem l'estat
     $MOVEI R1, pr1              ; 1er parametre: adreça de PR1
     $MOVEI R2, pr2              ; 2on paràmetre: ...
     LD     R2, 0(R2)            ; ... valor de PR2
     $CALL  R6, exemple          ; cridar la subrutina
     $POP      R2,  R1           ; restaurem l'estat
     ;ficrida
     ADD       R0, R0, R1        ;r = r + a ...
     ADD       R0, R0, R2        ;... + b
     $MOVEI    R3, r
     ST     0(R3), R0            ; recollir el resultat
     HALT                        ; Nota: Resultat = 6

exemple:                         ; COS DE LA SUBRUTINA
     LD     R3, 0(R1)            ; R3 serà l1 = pf1[0]
                                 ; R2 serà l2 (no cal fer res)
     ADD    R0, R3, R2           ; R0 = l1+l2
     ADD    R0, R0, R0           ; R0 = (l1+l2)*2
     ADD    R0, R0, R1           ; R0 = pf1+(l1+l2)*2 = &pf1[l1+l2]
     LD     R0, 0(R0)            ; R0 = pf1[l1+l2]
     JMP    R6                   ; retornar de la subrutina
; Fi de f
```

Figura 3.4: Exemple de codi en SISA-F que utilitza una subrutina

Enunciats de la sessió

Activitat 3.A: Programar una subrutina senzilla

En aquesta activitat has de traduir a SISA-F la subrutina `f` del programa de la figura 3.5. Tingues en compte les regles descrites a la lectura prèvia (preservació de l'estat, registres on s'esperen els paràmetres, registre amb l'adreça de retorn, etc.).

```
int n;
main() {
     n = f(3, 10, 5);            // 6.resultat: n=9
}

int f (int a, int b, int c) {    // 1.inici f: R1=3, R2=10, R3=5
register int r;

     r = g(c, a);                // 4.resultat: r=1
     return b - r;               // 5.final f: R0=9
}

int g(int x, int y) {            // 2.inici g: R1=5, R2=3
     return y*2 - x;             // 3.final g: R0=1
}
```

Figura 3.5: Programa considerat en l'activitat 3.A

Completa l'exercici 3.1 abans de continuar:

Exercici 3.1: Programa en assemblador la subrutina `f` del programa de la figura 3.5.

```
f:
```

Al fitxer **s3a.s** estan ja programats en SISA-F el programa principal (`main`) i la subrutina `g`. Escriu-hi el codi de l'exercici 3.1. A continuació, assembla'l i executa'l. Comprova el resultat pas a pas (F5), tenint en compte els resultats parcials indicats a la Figura 3.5 (sols cal mirar la Vista de Registres). Al final del programa el valor de la variable `n` ha de ser `9`.

Activitat 3.B: Una crida a subrutina dins d'un bucle

En aquesta activitat volem traduir a SISA-F el programa de la figura 3.6. El vector `alfabet` és un *string* que conté la llista ordenada de les lletres majúscules i, com és costum en els strings, acaba amb un byte que val 0. Volem escriure una funció `codifica` tal que donada una paraula d'entrada ens generi una paraula de sortida, on cada lletra ha estat intercanviada, de la següent manera: una 'A' es convertirà en una 'Z' o viceversa; una 'B' en una 'Y' o viceversa, etc. El programa principal fa dues crides a `codifica`. La primera vegada li passem un string d'entrada `w1` = "ARQUITECTURA", i retorna un string de sortida `w2`. En la segona crida li passem com a entrada `w2`, i retorna un string de sortida w3.

```
char alfabet[27]  = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
char w1[16] = "ARQUITECTURA";
char w2[16];
char w3[16];
int count=0;

main()
{
     count  = codifica(w1, w2);
     count += codifica(w2, w3);
}

char g(char alfa[], char *pfrase)
{
     return alfa[25-(*pfrase - 'A')];
}

int codifica(char *pfrasein, char *pfraseout)
{
     register int i;

     i = 0;
     while (*pfrasein != 0)
     {
            *pfraseout = g(alfabet, pfrasein);
            pfrasein++;
            pfraseout++;
            i++;
     }

     *pfraseout = 0;
     return i;
}
```

Figura 3.6: Programa de l'activitat 3.B

Podeu veure a la figura 3.6 que la funció `codifica` recorre seqüencialment els dos vectors fins que troba el byte 0; en cada iteració, crida a la funció auxiliar `g` perquè calculi la lletra canviada; i al final retorna el nombre total de lletres modificades.

Completeu l'exercici 3.2 abans de continuar:

Exercici 3.2: Programeu en SISA-F la subrutina `codifica`, la qual fa crides a la funció `g`.

```
codifica:
```

A continuació, escriviu el codi de la subrutina `codifica` en el fitxer **s3b.s**. Aquest fitxer ja conté correctament programat el `main`, la funció `g` i les variables globals. Assembleu el programa.

Per començar, executeu el programa fins al final (tecla F4) i comproveu que les variables `count`, `w2` i `w3` valen respectivament: `0x18`, "`ZIJFRGVXGFIZ`" i "`ARQUITECTURA`". Si és així, enhorabona! Si no, caldrà que depureu el programa pas a pas (tecla F5):

Depurar un programa complex com aquest no és simple, i els errors no són fàcils de trobar a simple vista. Un consell abans de començar a depurar! Si polseu una tecla equivocada o us perdeu pel camí, reinicieu l'execució de nou (tecla F9). Si en un punt determinat no us dóna el resultat esperat, penseu què és el que està equivocat i com ho corregiríeu, sortiu del depurador (Ctrl-X) i modifiqueu el programa amb l'editor.

Sessió 4: E/S per enquesta i tractament de bits

Objectiu: Aprendre a escriure directament a la pantalla del SISP-F. Entendre el mecanisme de sincronització per enquesta i construir programes senzills per llegir del teclat i escriure a la impressora. Practicar la manipulació de bits (desplaçaments i operacions lògiques bit a bit).

Lectura prèvia:

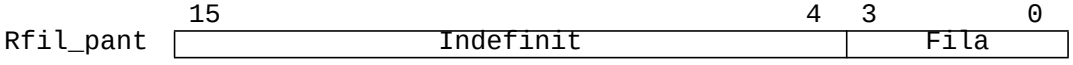
A continuació es descriuen els registres dels principals dispositius que s'utilitzaran en aquesta pràctica, i el seu funcionament. En aquells registres que són exclusivament d'escriptura, els bits que no tenen un significat concret s'indiquen com "indefinit", i vol dir que no importa quin valor s'hi escrigui, ja que el dispositiu ignora aquests bits. A més, el valor d'aquests registres resta indefinit un cop finalitzada la transferència, excepte per als registres Rfil_pant, Rcol_pant i Rdat_pant, els quals són "persistents", és a dir que conserven l'últim valor que s'hi ha escrit.

Pantalla

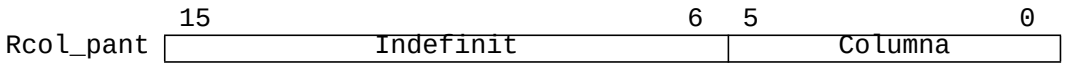
La pantalla és un display rectangular de 16 línies per 64 columnes. Els caràcters que s'hi escriuen poden usar sols blanc o negre per al primer pla i per al fons, adoptant dues apariències, anomenades modes o atributs: mode vídeo normal (mateix fons que la resta de pantalla) i mode vídeo invers (fons del caràcter diferent que la resta de la pantalla). Aquest dispositiu és únicament de sortida, és a dir, no es poden llegir caràcters que hi hagin estat escrits.

La pantalla no necessita sincronització ja que és un dispositiu ràpid. Es considera que quan acaba la instrucció que dóna l'ordre d'escriptura en pantalla, el caràcter ja s'ha escrit. Aquest dispositu es controla amb 4 registres, associats als ports 0x04, 0x05, 0x06 i 0x07:

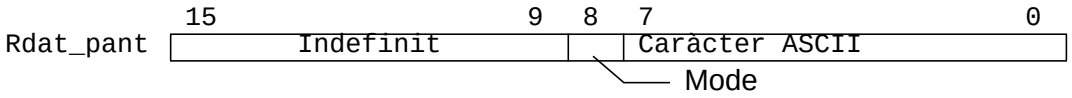
Rfil_pant: Registre d'escriptura. Indiquem la fila on volem escriure un caràcter (4 bits). Aquest registre està associat al port 0x05 (àlies Rfil_pant al fitxer crt0.s). El seu format és:



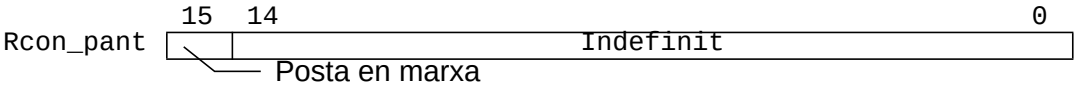
Rcol_pant: Registre d'escriptura. Indiquem la columna on volem escriure un caràcter (6 bits). Està associat al port 0x06 (àlies Rcol_pant). El seu format és:



Rdat_pant: Registre d'escriptura. Indiquem el codi ASCII del caràcter (8 bits) i el mode de vídeo en què ha d'aparèixer (1 bit). Si el bit de mode val 0, s'escriu en mode normal; si val 1, en mode invers. Està associat al port 0x07 (àlies Rdat_pant). El seu format és:



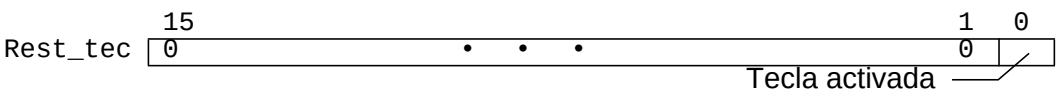
Rcon_pant: Registre d'escriptura. L'escriptura d'un caràcter en pantalla pot requerir accedir a un o més registres, però no es fa realment efectiva fins que escrivim un 1 en el bit de posta en marxa d'aquest registre. Quan ho fem, el darrer caràcter (i mode) escrit al registre Rdat_pant es mostrarà per pantalla, a la darrera fila i columna escrites en Rfil_pant i Rcol_pant respectivament. Està associat al port 0x04 (àlies Rcon_pant). El seu format és:



Teclat

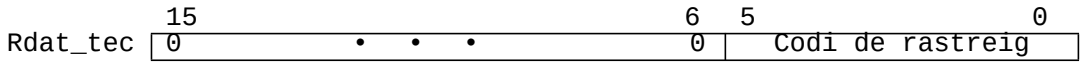
El teclat consta de 43 tecles totes elles situades al bloc principal del teclat. Inclou les tecles marcades amb les lletres de la A a la Z (excepte la Ñ i la Ç); amb els números del 0 al 9; amb els signes de puntuació coma, punt, guió i suma (, . - +); i les tecles ESPAI, INTRO i BACKSPACE. Cadascuna d'aquestes tecles està associada a un identificador (entre el 0 i el 42) que anomenem **codi de rastreig**. El teclat es controla amb 3 registres, associats als ports 0x08, 0x09 i 0x0A.

Rest_tec: És un registre de lectura. El bit de tecla activada es posa a 1 quan es polsa alguna tecla, i es manté així encara que la alliberem. El bit es torna a posar a 0 automàticament quan una instrucció llegeix el registre Rdat_tec. Està associat al port 0x0A (àlies Rest_tec). El seu format és:



Rdat_tec: És un registre de lectura, d'on es pot llegir el codi de rastreig (6 bits) que identifica la tecla polsada. Quan es llegeix aquest registre, automàticament s'indica al teclat que aquest codi s'ha llegit i aquest posa a 0 el bit de tecla activada del registre Rest_tec. El

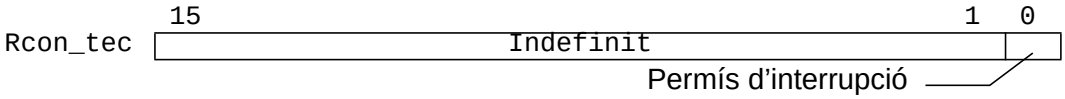
resultat de llegir Rdat_tec estant el bit de tecla activada a 0 és indefinit (p.ex., en la segona de dues lectures seguides!). Està associat al port 0x09 (àlies Rdat_tec) i el seu format és:



A més a més, en el fitxer **crt0.s** s'ha definit globalment un vector de 43 caràcters anomenat `tteclat`. Conté els caràcters que associarem a cada tecla o codi de rastreig, de forma que el codi ASCII s'obté simplement llegint `tteclat[codi de rastreig]`.

```
tteclat:      .ascii  "1234567890\bQWERTYUIOP+ASDFGHJKL\nZXCVBNM,.- "
              .balign 2
```

Rcon_tec: És un registre d'escriptura, i li indiquem al teclat si s'ha de sincronitzar per interrupcions (permís d'interrupcions = 1) o per enquesta (permís = 0). Està associat al port 0x08 (àlies Rcon_tec) i el seu format és:



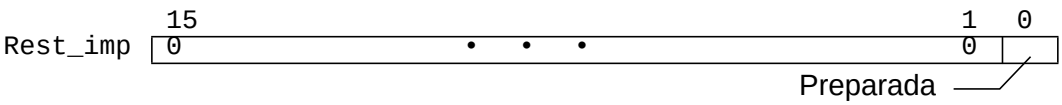
Impressora

La impressora permet imprimir línies de 64 caràcters sobre una cinta de paper. El funcionament de la impressora és similar al del teclat. El controlador d'impressora té 3 registres.

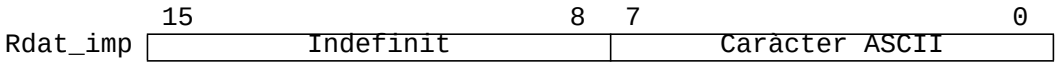
Rest_imp: És un registre de lectura, i indica l'estat de la impressora. El bit de "preparada" indica si la impressora ha acabat la feina i està llesta per a rebre caràcters (preparada = 1) o està ocupada imprimint (preparada = 0). En el primer cas, la finestra mostra el missatge "Ready" i en el segon cas el missatge "Busy". No es pot predir el temps que tardarà la impressora en imprimir un caràcter, entre altres raons perquè es pot produir un error del dispositiu (manca de paper o tinta, o un embús, o posta fora de línia) que retardi l'escriptura fins que l'usuari resolgui el problema manualment.

El depurador sisa-dbg simula aquests errors generant-los de forma aleatòria pero amb una freqüència molt baixa, de fet és molt poc probable que t'hi trobis. En cas d'error la finestra de la impressora mostra el missatge "Out of Paper" i el registre d'estat té el bit de preparada a zero. La impressora no retorna a l'estat de preparada fins que l'usuari "subsani" manualment el problema, enfocant aquesta finestra (amb TAB) i polsant la tecla d'espai.

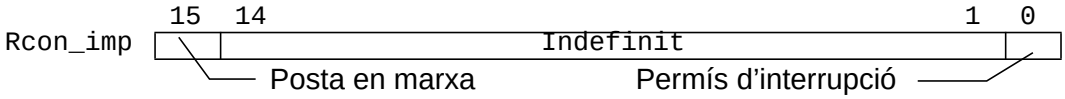
Aquest registre està associat amb el port 0x3A (àlies Rest_imp). El seu format és:



Rdat_imp: És un registre d'escriptura, on li indiquem el caràcter ASCII que volem enviar a la impressora (8 bits). Està associat al port 0x39 (àlies Rdat_imp). El seu format és:



Rcon_imp: És un registre d'escriptura, i permet indicar a la impressora si se sincronitzarà per interrrupcions (permís d'interrupcions = 1) o per enquesta (permís = 0). A més, escrivint un 1 al bit de posta en marxa li indiquem que ha de començar a imprimir el caràcter emmagatzemat al registre de dades (Rdat_imp). Tanmateix, el comportament de la impressora és impredecible si es fa la posta en marxa sense assegurar-se que està a 1 el bit de preparada. Està associat amb el port 0x38 (àlies Rcon_imp). El seu format és:



Estat Inicial dels dispositius

Quan el SISP-F s'inicia, tots els dispositius estan preparats per sincronitzar-se per enquesta, ja que els respectius bits de permís d'interrupcions estan a 0, així com el bit I del registre PSW del processador (bit 1 de S7). A més a més, la impressora està inicialment preparada.

Tractament de bits

Desplaçaments de bits

Per desplaçar els bits d'un número una posició a la dreta o a l'esquerra es copia cada bit a la posició adjacent. En un desplaçament de múltiples posicions, el resultat és el mateix que s'obtindria de desplaçar un lloc múltiples vegades.

Desplaçar els bits d'un número una posició a la dreta es pot fer de dues formes diferents: desplaçament lògic i desplaçament aritmètic. Mentre que en un desplaçament lògic a la dreta, el bit de major pes passa a valdre 0, en un desplaçament aritmètic a la dreta, aquest bit no es modifica. Per als desplaçaments d'una posició a l'esquerra no hi ha diferència entre lògic i aritmètic, i el bit de menor pes sempre passa a valdre 0.

El repertori SISA-F disposa de les instrucions SHL i SHA per als desplaçaments lògics i aritmètics de k posicions, respectivament. Valors positius de k indiquen desplaçaments a l'esquerra, valors negatius a la dreta.

Exemples:

```
MOVI    Rb, 3
SHL     Rd, Ra, Rb          ; Rd <- Ra despl. lògic a l'esqerra 3 bits
                            ; En desplaçaments a l'esquerra SHL == SHA
MOVI    Rb, -3
SHL     Rd, Ra, Rb          ; Rd <- Ra despl. lògic a la dreta 3 bits
MOVI    Rb, -3
SHA     Rd, Ra, Rb          ; Rd <- Ra despl. aritmètic a dreta 3 bits
                            ; Diferent de SHL ja que extén el signe
```

Un dels usos freqüents dels desplaçaments és el de multiplicar i dividir per potències de 2. El desplaçament d'un lloc a l'esquerra d'un nombre natural o enter equival a multiplicar-lo per 2. Tant el desplaçament lògic d'un lloc a la dreta d'un nombre natural com el desplaçament aritmètic d'un lloc a la dreta d'un nombre enter parell o positiu equival a dividir el nombre per 2. En canvi, per als enters senars negatius l'operació de desplaçament a la dreta no equival a la divisió per 2, com la que fa la instrucció DIV, on el reste té sempre el signe del dividend, sinó que obté un quocient i reste diferents, on el reste és sempre positiu.

Operacions lògiques bit a bit

Les operacions lògiques AND i OR bit a bit s'usen freqüentment per modificar bits individuals dintre de una dada. Un operand conté la dada a modificar i l'altre operand conté un patró de bits, anomenat màscara, que indica quins bits es modificaran i quins no. L'operació AND serveix per posar bits a zero, i la màscara corresponent ha de tenir a zero solament aquells bits a modificar. A la inversa, l'operació OR serveix per posar bits a u, i la màscara corresponent ha de tenir a u solament aquells bits a modificar. El repertori d'instruccions SISA-F disposa de les instrucions AND i OR bit a bit. Per exemple:

```
$MOVEI R2, 0b1111111111110011
AND    R1, R1, R2                ; posa a 0 els bits 2 i 3 de R1
$MOVEI R2, 0b0000000000001100
OR     R3, R3, R2                ; posa a 1 els bits 2 i 3 de R3
```

També està disponible la instrucció NOT bit a bit. En l'anterior exemple, hauríem pogut substituir la tercera línia (macro $MOVEI) per una NOT, obtenint idèntic resultat:

```
$MOVEI R2, 0b1111111111110011
AND    R1, R1, R2
NOT    R2, R2
OR     R3, R3, R2
```

La darrera instrucció bit a bit és el XOR, que ens permetrà complementar els bits que tinguin un 1 a la màscara:

```
$MOVEI R1, 0x0F0F
$MOVEI R2, 0x00FF
XOR    R1, R1, R2                ; R1 = 0x0FF0
```

Enunciats de la sessió

Les activitats d'aquesta sessió treballen amb els dispositius d'entrada/sortida. No oblideu incloure al principi del programa el fitxer de definicions dels dispositius, amb la directiva:

```
.include "crt0.s"
```

Per observar els dispositius d'entrada/sortida en el simulador cal activar el mode "emulador", amb la tecla F2. En aquest mode es mostren 3 panells. A dalt a l'esquerra, la sortida per pantalla del SISP-F, en format de 16 files per 64 columnes. A baix, el que surt per la impressora (s'escriu en la última fila, i es va desplaçant amunt a mesura que hi escrivim línies). A dalt a la dreta, el contingut actual dels registres d'estat de tots els dispositius, així com dels registres "persistents" de la pantalla i del disc (Rfil_pant, Rcol_pant, Rdat_pant, Rcara_disc, Rpist_disc, Rsect_disc i Radr_disc). En el mode "emulador" estan disponibles la majoria de les comandes relatives a l'execució contínua o pas a pas. Si es desitja observar el contingut dels registres o la memòria, sols cal passar novament al mode "depurador" amb la mateixa tecla F2. Podem passar d'un mode a l'altre tants cops com calgui.

Activitat 4.A: La pantalla

En aquesta activitat accedirem a la pantalla. Completa l'exercici 4.1 abans de continuar.

Exercici 4.1: Escriu un programa en alt nivell que mostri una 'A' a la posició [4,8] de la pantalla, amb atribut invers, i una 'B' a la posició [4,9] amb atribut normal. Observa que la fila és la mateixa en els dos casos, així que sols cal escriure-la un cop.

Tradueix el codi anterior a assemblador SISA-F, en el fitxer **s4a.s**. Assembla'l, i verifica amb el simulador que funcioni correctament.

Activitat 4.B: El teclat

L'objectiu d'aquest apartat és provar l'accés al teclat sincronitzant-lo per enquesta. Completa l'exercici 4.2 abans de continuar:

Exercici 4.2: Escriu un programa en alt nivell que esperi fins que es polsi una tecla qualsevol, i llavors escrigui una 'A' a la posició [4,8] de la pantalla, amb atribut normal.

A continuació tradueix el codi de l'exercici 4.2 a assemblador SISA-F, en el fitxer **s4b.s** i verifica'n el correcte funcionament amb el simulador.

Un cop funcioni l'anterior programa, completa l'exercici 4.3:

Exercici 4.3: Escriu una nova versió del programa en alt nivell anterior per tal que, en comptes d'una 'A', escrigui el caràcter associat a la tecla polsada. Fixa't que es tracta de fer sols un petit canvi en la finalització. Recorda que per traduir el codi de rastreig del teclat es disposa del vector `tteclat`.

Ara, tradueix l'anterior codi a assemblador SISA-F en el mateix fitxer **s4b.s** (sols cal afegir les modificacions). Assembla'l i comprova que funciona, amb el simulador.

Un cop comprovat el programa anterior, completa l'exercici 4.4:

Exercici 4.4: Escriu en alt nivell una nova versió del programa anterior, tal que faci la mateixa tasca (llegir una tecla i escriure-la en pantalla) però repetidament, fins que la tecla polsada sigui una 'F'. Fixa't que es tracta sols d'insertar l'anterior programa dins un bucle.

Tradueix el programa anterior a assemblador SISA-F en el mateix fitxer **s4b.s** (afegint sols les modificacions). Verifica'l amb el simulador.

Activitat 4.C: La impressora

L'objectiu d'aquest apartat és provar l'accés a la impressora sincronitzant-se per enquesta. Completa l'exercici 4.5 abans de continuar:

Exercici 4.5: Fes un programa en alt nivell que imprimeixi la paraula "`Fi`". Abans d'imprimir cada caràcter cal esperar que estigui preparada. I al final del programa, també.

A continuació tradueix el codi de l'exercici 4.5 a assemblador SISA-F en el fitxer **s4c.s** i verifica que funcioni correctament, amb el simulador. Recorda que el simulador pot generar errors de la impressora de forma aleatòria (el panell corresponent mostrarà el missatge "Out of paper"), fent que la impressora es quedi ocupada indefinidament. Si passa això, tan sols has d'enfocar el panell d'impressora (tecla TAB) i polsar la tecla d'espai, i la impressora es posarà preparada novament. En acabat, no oblidis d'enfocar novament el panell de pantalla.

Un cop funcioni l'anterior programa, completa l'exercici 4.6:

Exercici 4.6: Escriu en alt nivell una nova versió del programa anterior que escrigui el vector `frase` a la impressora. Fixa't que el vector és un *string*, Per tant, acaba amb un byte a zero (valor binari 0). Es tracta doncs de fer un bucle que iteri fins a trobar el zero final.

```
char frase[30] = "Aquest programa funciona";
```

Tradueix el programa anterior a assemblador SISA-F en el mateix fitxer **s4c.s**. Assembla'l i comprova amb el simulador el seu funcionament.

Activitat 4.D: Visualització d'operacions de tractament de bits

L'objectiu d'aquest apartat és combinar l'accés a la pantalla, al teclat, i les operacions de tractament de bits, i ho farem en 3 parts:

En primer lloc, escriurem un programa que mostri a les posicions [0,0]-[0,15] de la pantalla els 16 bits de la representació en binari (uns i zeros) de la variable *w* de mida word (figura 4.1), mostrant el bit més significatiu a la columna 0 i el menys significatiu a la columna 15. El programa *main* fa una crida a la subrutina *mostra* passant-li com a paràmetre per valor la variable *w*. Aquesta subrutina no retorna res ja que és una acció, però mostra la representació binària del paràmetre *i* a les posicions [0,0]-[0,15]. Completa l'exercici 4.7 abans de seguir:

Exercici 4.7: Tradueix a SISA-F el programa `main` i la subrutina `mostra` de la figura 4.1


```
int w=0x8888;
main() {
     mostra(w);
}

int mostra(int i) {
     register int col=16;
     out (Rfil_pant, 0);
     do {
            col--;
            out (Rcol_pant, col);
            out (Rdat_pant, '0' + (i & 0x1));
            out (Rcon_pant, 0x8000);
            i = i >> 1;
     } while (col > 0);
}
```

Figura 4.1: Codi en alt nivell del programa `main` i de la subrutina `mostra`

Escriu el codi anterior al fitxer **s4d.s,** i verifica que funcioni bé amb el simulador.

En segon lloc, volem extendre el programa anterior, perquè es comporti així:

1) Mostrar el valor de *w* (el codi del programa anteerior*)*.
2) Esperar fins que es polsi una tecla.
3) Modificar *w* segons quina sigui la tecla polsada:
   - Si la tecla és una 'A', fer un desplaçament **lògic** de *w* (`SHL`), 1 lloc a l'**esquerra**.
   - Si és una 'B', fer un desplaçament **lògic** de *w* (`SHL`), 1 lloc a la **dreta**.
   - Si és una 'C', fer un desplaçament **aritmètic** de *w* (`SHA`), 1 lloc a la **dreta**.
   - Si és una 'D', s'ha de **dividir per dos** l'enter *w* (`DIV`).
   - Si és un número *n* (tecles '0', ... '9'), s'han de **complementar** els *n* bits de menys pes de *w*. Nota: aquesta operació es pot escriure en C així: `w = w^((1<<n)-1);` on l'operador `^` significa xor bit a bit, i l'operador `<<` significa desplaçament a esquerra
4) Repetir des del pas número 1 (el programa no acaba, fa un bucle sense fi).

Completa l'exercici 4.8 abans de continuar:

Exercici 4.8: Escriu en alt nivell (en C) el nou programa principal (`main`), suposant que l'acció `mostra` és la que està declarada a la figura 4.1.

```
main()
{









}
```

A continuació, tradueix el programa anterior a SISA-F, en el mateix fitxer **s4d.s**, substituint el codi del `main` que hi teníeu abans, per aquest. Assembla'l i executa'l.

Verifica que el programa funciona, per exemple amb un valor inicial de `w = 0x8888`. La primera línia de la pantalla hauria de mostrar la cadena "`1000100010001000`". Si es prem la tecla 'A', la primera línia de la pantalla hauria de mostrar la cadena "`0001000100010000`". Ves provant a prémer totes les tecles habilitades ('A', 'B', 'C', 'D', '0', ... '9') i verifica que realitzen la funció desitjada.

En tercer lloc volem provar el funcionament del programa amb un enter negatiu. Modifica la inicialització de `w` en el fitxer **s4d.s** per tal que `w = 0xFFF9`. Torna a assemblar i muntar el programa, i executa'l.

Exercici 4.9: Comprova que quan `w` és negatiu (`w = 0xFFF9`), la divisió `DIV` per 2 (tecla 'D') dóna un resultat diferent que el desplaçament SHA una posició a la dreta (tecla 'C'). **Per què?**

Sessió 5: Entrada/sortida per interrupcions

Objectiu: Entendre el mecanisme bàsic de sincronització per interrupcions, a través de programes senzills que mostrin el seu funcionament. Aprendre a escriure correctament la inicialització del vector d'interrupcions, la programació d'una RSI específica, l'ús de les variables de sincronització, etc.

Lectura prèvia

En aquesta sessió aprendrem a fer petits programes en llenguatge assemblador SISA-F que se sincronitzen per interrupcions amb els dispositius de rellotge, teclat i impressora, i escriuen els seus resultats a la pantalla o a la impressora. A continuació trobareu un resum dels punts clau a tenir en compte per programar per interrupcions els dispositius del SISP-F.

El mecanisme d'interrupcions en el SISP-F

Quan un dispositiu d'entrada/sortida requereix l'atenció del processador per a realitzar una transferència de dades, i a més a més té activat el permís corresponent, genera un senyal de petició d'interrupció (INT) per avisar al processador. Si, a més a més, el processador té activat el permís genèric per atendre interrupcions (bit I del PSW = 1), llavors atendrà aquesta petició interrompent temporalment el programa que estigui executant, i passant a executar la rutina de servei genèrica (RSG) per a excepcions i interrupcions, l'adreça de la qual està escrita en el registre especial S5. Aquesta rutina identificarà el dispositiu responsable de la petició i invocarà la rutina de servei d'interrupció (RSI) específica per atendre a aquest dispositiu. Anem a examinar alguns detalls d'aquest procés:

Quan el processador ha d'invocar la RSG fa les següents accions: salva en S0 una còpia del registre S7 i posa a 0 el bit de permís d'interrupcions (bit 1 de S7); escriu en S2 el valor 15, per informar a la rutina que l'event ha estat una interrupció (externa); copia en S1 el valor del PC actualitzat (PCup), el qual conté l'adreça de retorn al programa interromput; i finalment copia en el PC el registre S5, per saltar a la RSG.

Per defecte, tota l'execució de la RSG i la RSI discorre amb les interrupcions inhibides. Aquest comportament és convenient per a les rutines RSI senzilles que farem en EC1. La majoria dels sistemes operatius, amb rutines molt més complexes, procuren tenir les

interrupcions inhibides durant el mínim temps indispensable, però això requereix programar les RSI considerant la concurrència, i no ho veurem en EC1.

Per tal d'identificar el dispositiu, la RSG executa la instrucció `GETIID Ri`. Aquesta instrucció envia un senyal (INTA) als dispositius, de forma que tan sols el dispositiu que ha generat la interrupció (o el més prioritari, en cas que n'hi hagués més d'un) respon a aquest senyal enviant un identificador de dispositiu (id), que el processador escriurà en el registre `Ri`. Usant aquest identificador com a índex, la RSG accedeix al vector d'interrupcions (`interrupts_vector`) per obtenir l'adreça de la RSI específica, i a continuació la invoca amb una instrucció `JAL`:

```
GETIID R1                          ; obtenim el id
$MOVEI R2, interrupts_vector
ADD    R1, R1, R1
ADD    R2, R2, R1
LD     R2, 0(R2)                   ; @RSI = interrupts_vector[id]
JAL    R6, R2                      ; crida la RSI
```

Com que la invocació de la RSG modifica la paraula d'estat de S7, aquesta rutina retorna fent servir una instrucció de salt especial RETI que restaura S7. Aquesta instrucció copia en S7 el valor prèviament salvat en S0, i retorna al programa interromput copiant en el PC el valor prèviament guardat en S1.

Les rutines de servei d'interrupcions específiques (RSI)

Una RSI ha de seguir les normes convencionals de programació de subrutines. És a dir, que per retornar ho fa amb la instrucció `JMP R6`, i si ha d'invocar una altra subrutina, ha de salvar a la pila prèviament els registres que pugui necessitar més tard, tal com s'ha explicat al tema de subrutines. Però a més, hem de tenir en compte alguns aspectes addicionals:

1.- La RSG i les RSI no han de ser mai invocades per cap instrucció JAL del programa: la RSG s'invoca a instàncies d'events externs impredecibles, i la RSI la invoca tan sols la RSG.

2.- La RSI no rep paràmetres de cap mena ni retorna cap resultat. Qualsevol informació que s'hagi de comunicar entre la RSI i el programa principal s'ha de fer a través de variables globals (variables "de sincronització"). Teòricament, per a programes molt simples del SISP-F, seria possible passar paràmetres a la RSI en alguns registres prèviament convinguts, però és una mala pràctica de programació ja que no és possible fer-ho en els sistemes reals, amb programes notablement complexos, i possiblement molts d'ells executant-se de forma concurrent. Així doncs, en EC1 no s'admetran rutines RSI que es comuniquin via registres amb el programa principal.

El programa principal

Quan el SISP-F s'inicia, cap dispositiu té interrupcions permeses i el SISP-F té el bit I de la paraula d'estat (S7<1>) a 0, i no atén interrupcions. El programa principal és responsable de:

- Activar els bits de permís d'interrupcions en els dispositius (en els registres de control)
- Activar l'atenció d'interrupcions en el bit I de la paraula d'estat (amb la instrucció `EI`)
- Modificar l'element que correspongui del vector d'interrupcions per tal que apunti a la nostra rutina RSI (que ja tindrem escrita). P. ex., en el cas de la impressora (id=7):

```
$MOVEI        R1, la_meva_RSI_de_impressora
$MOVEI        R2, interrupts_vector
ST            7*2(R2), R1
```

El teclat

Aquest dispositiu ja s'ha estudiat a la sessió anterior. Sols ens cal afegir que:

- Genera una petició d'interrupció cada cop que polsem una tecla, amb id = 1
- La petició es manté activa fins que el processador llegeix el registre de dades del teclat (port 0x09, àlies Rdat_tec). Per tant, convé assegurar-se que dins la RSI de teclat es faci sempre la lectura d'aquest registre, per tal de desactivar la petició, encara que no ens interessi el seu contingut, escrivint: `IN Ri,Rdat_tec`

La impressora

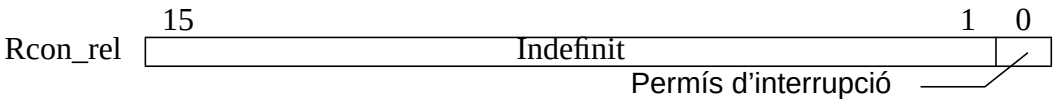
Aquest dispositiu també s'ha estudiat a la sessió anterior. Sols ens cal afegir que:

- Genera una petició d'interrupció cada cop que finalitza la impressió d'un caràcter, id = 7
- La petició es manté activa fins que el dispositiu rep el senyal INTA (quan executem la instrucció `GETIID`, dins la RSG)

El rellotge

Es tracta en realitat d'un temporitzador (no el confongueu amb el senyal de rellotge que serveix per seqüenciar els circuits del processador!), que no produeix entrades ni sortides de dades, sinó tan sols interrupcions a intervals regulars.

- Genera una petició d'interrupció cada 0,1 segons, amb id = 0
- La petició es manté activa fins que el dispositiu rep el senyal INTA.
- Només té un registre, d'escriptura, associat al port 0 (àlies **Rcon_rel**): El bit 0 indica si té permís per generar peticions d'interrupció (permís=1) o no (=0). El seu format és:



Enunciats de la sessió

Les activitats d'aquesta sessió treballen amb els dispositius d'entrada/sortida. No oblidis incloure al principi del programa el fitxer d'inicialització `crt0.s`, amb la directiva:

```
.include "crt0.s"
```

Recorda que per observar els dispositius d'entrada/sortida en el simulador cal activar el mode "emulador" (F2), tal com s'ha explicat a la sessió anterior. La tecla F2 permet alternar entre els dos modes "depurador" i "emulador" en qualsevol moment de l'execució.

Nota important. Per depurar rutines RSI resulta especialment útil activar un punt d'aturada dins la RSI (portar el cursor al punt desitjat i polsar la tecla B). A continuació, ja sigui en mode emulador o en mode depurador, polsem "execució contínua" (F4) i esperem fins que l'execució s'aturi al nostre punt d'aturada, a la RSI. En aquest punt serà millor seleccionar el mode depurador i seguir executant pas a pas (F5) per observar en detall què fa la RSI: quins valors llegeix o escriu en els registres, en les variables globals, etc. Amb la tecla F4 podem proseguir l'execució contínua (fins al següent punt d'aturada).

Activitat 5.A: Rellotge

L'objectiu d'aquest apartat és programar el rellotge. Fes l'exercici 5.1:

Exercici 5.1: Escriu un programa en alt nivell que esperi fins que hagin transcorregut 5 segons, i llavors escrigui una 'A' a la posició [4,8] de la pantalla, amb atribut normal. Escriu el `main` i la RSI de rellotge

Tradueix aquest programa a SISA-F, en el fitxer **s5a.s**. Verifica'l amb el simulador.

Activitat 5.B: Teclat

Farem un programa que se sincronitzi per interrupcions amb el teclat. Completa l'exercici 5.2.

Exercici 5.2: Escriu un programa en alt nivell que esperi fins que polsem una tecla qualsevol, i llavors escrigui en pantalla, a la posició [4,8], el caràcter associat a aquesta tecla, en mode invers. Escriu el `main`, la RSI de teclat, i les variables que necessitis.

Tradueix aquest programa a SISA-F en el fitxer **s5b.s** i verifica'l amb el simulador.

A continuació completa l'exercici 5.3

Exercici 5.3: Modifica el programa anterior perquè repeteixi la mateixa tasca per a cada tecla que polsem, i acabi quan la tecla polsada sigui la 'F'.

Tradueix aquest programa a SISA-F en el mateix fitxer **s5b.s** Verifica'l amb el simulador.

Activitat 5.C: Rellotge i impressora

L'objectiu d'aquest apartat és provar l'accés a la impressora, i combinar-lo amb el rellotge. Tots dos dispositius s'han de sincronitzar per interrupcions. Completa l'exercici 5.4:

Exercici 5.4: Escriu un programa en alt nivell que escrigui el vector `frase` a la impressora, deixant que aquesta resti inactiva durant 1 segon entre lletra i lletra. Cal escriure el main i les dues RSI. Tingues en compte que el vector és un string, i acaba amb un byte que val 0.

```
char frase[32] = "Aquest programa funciona";
```

Tradueix el programa a SISA-F en el fitxer **s5c.s**, i verifica'l amb el simulador[1].

1. Recorda que, encara que és molt improbable, el simulador pot generar errors de la impressora de forma aleatòria, fent que es quedi ocupada indefinidament. Si sospites que pot estar passant això (comprova-ho al registre d'estat, en el panell dret) sols has de polsar una tecla i la impressora es posarà preparada novament.

Activitat 5.D: (Opcional) Rellotge i teclat

L'objectiu d'aquest apartat és escriure un petit joc una mica més complex, i que faci ús dels dispositius de rellotge, teclat i pantalla. La sincronització serà per interrupcions. Completa l'exercici 5.5 abans de continuar:

Exercici 5.5: Feu un programa en alt nivell que escrigui una 'X' a la posició [4,8] de la pantalla, en mode normal. Cada 0,4 segons, sobreescriurà la 'X' amb un espai en blanc (s'esborrarà) i la reescriurà a la posició adjacent (inicialment considerem adjacent la posició a la dreta de la posició anterior). El programa acaba quan la 'X' surt de la pantalla.

Tradueix el programa a SISA-F en el fitxer **s5d.s**, i verifica'l amb el simulador.

Un cop comprovat el programa anterior, completa l'exercici 5.6:

Exercici 5.6: Escriu una nova versió del programa anterior, amb la següent modificació: en comptes de moure la 'X' sempre cap a la dreta, volem que la direcció es pugui variar a voluntat per mitjà de les tecles 'A','S' (esquerra i dreta),'L' i 'P' (avall i amunt). Quan es polsi una d'aquestes tecles, la 'X' adoptarà la direcció corresponent. Nota: Les polsacions del teclat sols modifiquen la direcció, no "mouen" la 'X'. La 'X' sols es mou a intervals de temps. El programa acaba quan la 'X' se surt de la pantalla. Tingues ben clar quins son els distints casos que cal tractar, tant en la gestió del teclat com del rellotge.

Tradueix el programa anterior a assemblador, en el mateix fitxer **s5d.s** (afegint-hi sols les modificacions). Assembla'l i comprova'n el funcionament amb el simulador.