

Pràctica 1

Instruccions i registres

1.1 Objectius

- Presa de contacte amb aspectes bàsics de l'arquitectura del MIPS.
- Presa de contacte amb un llenguatge d'assemblador.
- Presa de contacte amb el simulador MARS.
- Escriure el primer programa en assemblador.

1.2 Material

Simulador MARS i un codi font de partida

1.3 Introducció teòrica

Introduïm a continuació una sèrie de conceptes que ens ajudaran a comprendre més bé la pràctica.

1.3.1 La màquina Von Neumann

La màquina de Von Neumann és el disseny teòric d'un computador de programa emmagatzemat realitzat el 1945 pel físic matemàtic John von Neumann (1903-1957). Aquest disseny és la base de la pràctica totalitat dels computadores d'avui dia. El concepte bàsic que hi ha darrere d'aquest disseny es la possibilitat de emmagatzemar instruccions de programes i les dades en una memòria i l'habilitat que aquestes instruccions operen amb les dades.

Una màquina de Von Neumann està formada per tres components diferents (o subsistemes): una unitat central de processament (CPU), una memòria i unes interfícies d'entrada i d'eixida

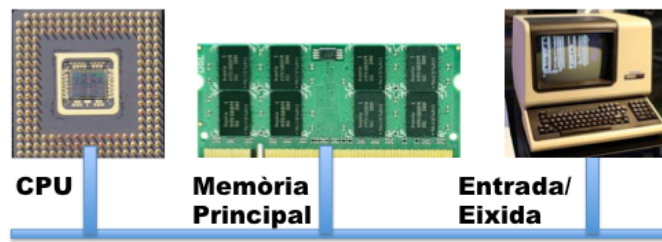


Figura 1: components d'una arquitectura Von Neumann

La unitat central de processament es dedica al processament de la informació i està formada per tres components principals, la unitat aritmètica i lògica, la unitat de control i un conjunt de registres.

La memòria principal s'utilitza per a emmagatzemar les instruccions dels programes i les dades.

L'entrada i eixida permet a la memòria del computador rebre informació i enviar dades a dispositius externs. També permet al computador comunicar-se amb l'usuari i amb dispositius externs d'emmagatzematge.

Tots tres components esmentats es comuniquen a través del bus del sistema.

1.3.2 Programa emmagatzemat

Un concepte inherent a la introducció de màquina de Von Neumann és el de programa emmagatzemat, que significa que tant les instruccions del programa com les dades sobre les qual operarà estan emmagatzemades en memòria. Això vol dir, per exemple, que poden ser modificades independentment de la màquina a diferència dels computadores anteriors, que s'havien de programar mitjançant commutadors i cables d'interconnexió.

1.3.3 Programa d'alt nivell versus codi màquina

Quan escrivim un programa en un llenguatge en alt nivell (per exemple C o Java), el programa no és més que una cadena de caràcters ASCII que hem introduït a través del teclat i hem emmagatzemat en un fitxer. Per a poder executar el programa necessitem traduir-lo a un conjunt de instruccions executables (*codi de màquina*, format per zeros i uns) que indiquen a un computador determinat què ha de fer. El programa s'ha de traduir en instruccions que puga entendre el processador particular que conté el computador, siga, per exemple d'AMD, d'Intel, etc. Aquesta traducció la fa el *compilador*.

1.3.4 Llenguatge d'assemblador

En els primers computadors construïts els anys quaranta del segle passat els programadors escrivien els programes directament en codi de màquina, assignant directament els valors de 0 i 1 mitjançant l'activació d'interruptors físics. Per a facilitar aquesta tasca es van introduir el llenguatges d'assembladors. Un programa en llenguatge d'assemblador és un fitxer ASCII que necessita ser traduït en codi de màquina, una traducció que és relativament fàcil i directa. La traducció a llenguatge de màquina la fa un programa anomenat *assemblador*. No estudiarem com ho fa exactament l'assemblador, però sí que estudiarem com programar en un llenguatge d'assemblador i entendrem la correspondència exacta del programa intel·ligible pel processador. En la figura podeu observar el procés de traducció d'un compilador i d'un assemblador:

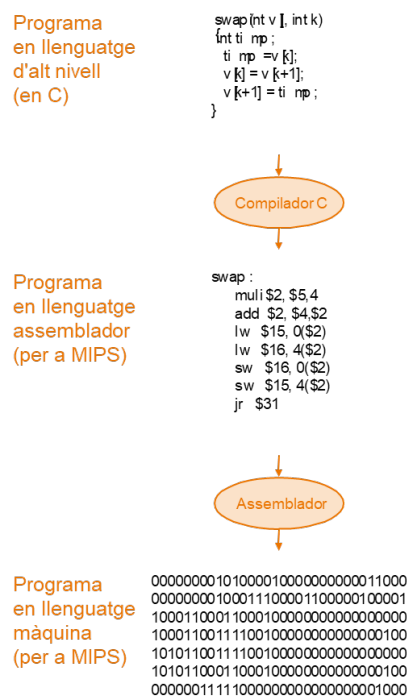


Figura 2. Procés de compilació i assemblatge

1.3.5 Cicle d'instrucció

Tot processador requereix un cicle d'instrucció per a executar una instrucció particular. El cicle d'instrucció constarà d'una sèrie de passos que permeten al processador llegir la instrucció emmagatzemada en la memòria i realitzar les accions oportunes sobre els operands requerides per la instrucció.

1.3.6 MIPS

En aquest curs ens fixarem en una màquina particular, el processador MIPS, (*Microprocessor without Interlocked Pipeline Stages*) i encara que hi ha diverses versions (32 bits (R2000 i R3000), 64 bits (R4000)) ens centrarem en el MIPS R2000 i aprendrem a programar en el seu llenguatge d'assemblador denominat també *MIPS*.

Abans de res hem de introduir el model de la màquina que ha de conèixer el programador d'assemblador d'aquesta màquina. Els elements del processador que són visibles al programador són:

- Un banc de registres d'enters o de registres de propòsit general que està format per 32 registres de 32 bits. Això significa que necessitem 5 bits per seleccionar cada registre. Encara que el MIPS té altres bancs de registres, els deixarem de banda i els estudiarem més avant.
- Una memòria principal formada per paraules de 32 bits. La memòria és adreçable per byte, mitja paraula i paraula sencera. Per a seleccionar una paraula es necessiten adreces de 32 bits. En practiques posteriors estudiarem com fer l'accés a la memòria.
- Una unitat aritmètica i lògica que permet realitzar operacions aritmètiques i lògiques sobre les dades d'entrada.
- Un conjunt de funcions d'Entrada/Eixida del sistema. És programari bàsic vingut de fàbrica.
- El registre PC (*Program Counter*) que sempre conté l'adreça en memòria on es troba la instrucció a executar. També hi ha altres registres especials que anirem introduint en el moment que siguin necessaris.

En la figura 3 podeu observar el model del processador MIPS simplificat:

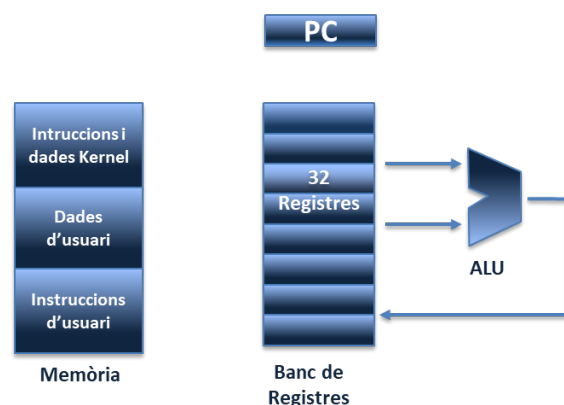


Figura 3. Model del processador MIPS

1.3.7 Model del simulador MARS

El MARS és un simulador programari que permet executar programes escrits en assembleador MIPS R2000/R3000. Es tracta d'un entorn de desenvolupament interactiu que permet la introducció del codi en assembleador. Amb aquest simulador podrem fer el seguiment de l'execució dels programes i veure en tot moment el contingut dels registres del processador.

Es tracta d'un programa lliure implementat en Java i es pot descarregar des de la pàgina següent:

<http://courses.missouristate.edu/kenvollmar/mars/index.htm>

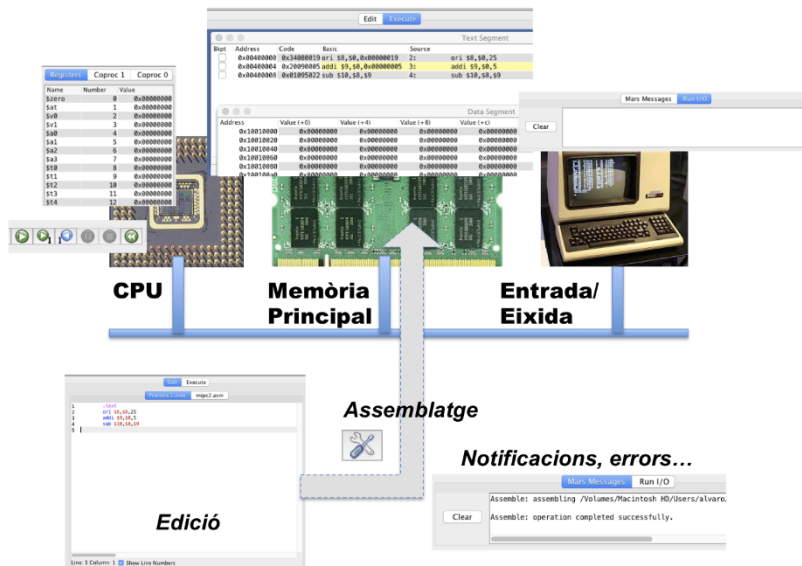


Figura 4. Visió general del processador

Amb l'objectiu d'evitar qualsevol mala interpretació terminològica, a continuació recollim el vocabulari general que usarem quan programarem en assembleador:

- *Llegir* (un registre, una paraula): és seleccionar un registre o una paraula, prendre'n el contingut i dur-lo a altre lloc: registre, paraula o operador. Llegir no canvia mai el contingut.
- *Escriure* (un registre, una paraula): és seleccionar un registre o una paraula i canviar-ne el contingut a un nou valor provinent d'una altre registre, paraula o operador.
- *Operador*: és un component del processador que fa un càlcul. A partir d'una o dues paraules, l'operador subministra el resultat del càlcul. Exemple: la suma.

- *Operació*: és l'acció que fa el processador. Pot implicar un operador o més. Les operacions més habituals tenen un significat aritmètic.
- *Instrucció*: En el cas del MIPS és una paraula de 32 bits que codifica una operació i els operands escaients. Veurem que hi ha maneres diverses d'especificar els operands.

n	Contingut	adreça	Contingut	\$v0	Nom
0	0x00000000	0x00000000	0x00000000	1	Print Integer
1	0xFFFFFFFF	0x00000004	0x00000000	2	Print float
2		0x00000008	0x00000000		
3					
31		0xFFFFFFFF	0x00000000		

Figura 5. Elements de processador.

1.4 Realització pràctica

Començarem la pràctica familiaritzant-nos amb el simulador MARS sobre el qual escriurem i provarem els programes en assembleador del MIPS que dissenyem. La pantalla principal del MARS es divideix en diferents parts: un conjunt de menús i submenús i un conjunt de finestres. Des d'aquesta pantalla podrem accedir a l'editor de textos per a escriure el codi, en el simulador de MIPS, en la finestra de registres, i observar els missatges que ens mostra l'assembleador.

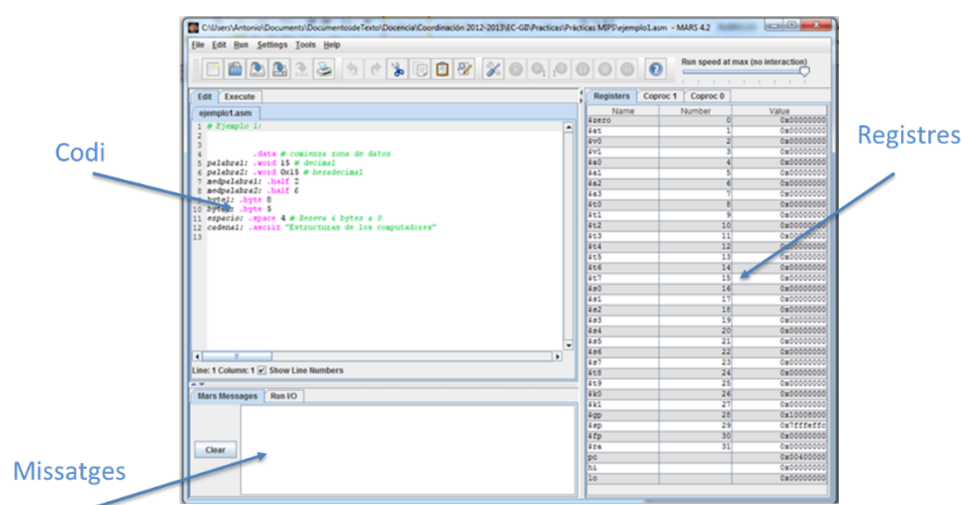


Figura 6. Pantalla principal del simulador MARS

1.4.1 La finestra *Registers*

En la finestra de registres podem observar el conjunt de registres disponibles en el processador MIPS amb el contingut. Vegeu que els registres \$0 a \$31 tenen dos noms i que, llevat del \$sp (*Stack pointer*), \$gp (*Pointer to global area*) i \$pc (*Program counter*), valen 0.

Podeu escollir la base de numeració activant o desactivant la casella corresponent, que la podeu trobar en (*Settings>Values Displayed in Hexadecimal*) o canviant a la pestanya *Execute (Hexadecimal Values)*.

1.4.1.1 Activitat 1

- Proveu de modificar el contingut d'algun registre. Noteu que hi podeu escriure en decimal o hexadecimal i que no hi podeu canviar \$0, \$31 ni \$pc.
- Proveu d'escriure valors negatius en els registres.

Qüestió 1

- Quin és el major positiu que pot contenir un registre del MIPS? N'hi ha prou que ho digueu en hexadecimal.
- Quin és el major negatiu que pot contenir un registre del MIPS? N'hi ha prou que ho digueu en hexadecimal.

1.4.2 El primer programa – anàlisi

Partirem del programa següent:

```
#####  
#                                     #  
#           Primer programa          #  
#                                     #  
#####  
  
.text 0x00400000  
addi $9,$8,25  
addi $10,$8,5
```

El símbol # marca l'inici d'un comentari. L'assemblador ignorarà el que hi haja a la dreta d'aquest símbol.

La línia *.text 0x00400000* diu en quina adreça de la memòria comença el programa. 0x00400000 és l'adreça per defecte; si no hi fiquem res, l'assemblador assumeix aquest valor.

La instrucció *addi* (*suma immediata*) s'escriu en llenguatge d'assemblador de la forma

addi rt,rs,K

en què *rt* i *rs* poden ser qualsevol número de registre del 0 al 31 i *K* qualsevol nombre codificat en complement a 2 amb una grandària de 16 bits. La instrucció fa $rt = rs + K$, és a dir, llig un registre font (*rs*), fa la suma del seu contingut i la constant *K* i escriu el resultat de la suma en un registre destinació (*rt*).

Gràficament podem expressar-ho com s'observa en la figura 7:



Figura 7. Representació gràfica de la instrucció *addi*

Hi notem dues maneres d'expressar els operands: assenyalant el número d'un registre com pot ser \$8 o \$9, o bé indicant constants com el valor 25 o 5.

Per tant, el programa que hem escrit fa la suma del contingut de \$8 i la constant 25 i emmagatzema el resultat en el registre \$9. Després fa la suma del contingut de \$8 i 5, i el resultat el guarda en \$10.

Les instruccions es codifiquen en binari per a emmagatzemar-les en la memòria. Totes les instruccions del MIPS es codifiquen en 32 bits. La instrucció *addi \$9,\$8,25* es codifica com s'observa en la figura 8.

<i>Codi op (6 bits)</i>	<i>Rs (5 bits)</i>	<i>Rt (5 bits)</i>	<i>K</i>
0010 00	01 000	0 1001	0000 0000 0001 1001

Figura 8. Codificació de la instrucció *addi \$9, \$8, 25*

En hexadecimal quedaria com: 0x21090019.

El primer camp és el codi de operació de 6 bits i indica que es farà la suma del registre font *Rs* i el nombre *K*.

Els següents dos camps són els números dels registres font i destinació, en l'exemple el 8 i el 9.

El tercer camp es el valor en complement a dos de la constant *K*, en l'exemple el valor 25.

Qüestió 2

- Com es codifica la instrucció `addi $t0,$8,5`? Escriviu el codi resultant en hexadecimal

1.4.3 Assemblatge

Escriviu, anomeu i guardeu el codi font en un arxiu de text. Assembleu (Run->Assemble) i pareu atenció en la finestra Mars Messages. Vegeu si hi diu Assemble: operation completed successfully.

En el moment d'assemblar apareixen dues finestres anomenades *Text segment* i *Data Segment*. Ara ens fixarem únicament en la de *Text Segment*.

1.4.4 La finestra *Text segment*


Si ens fixem en aquesta finestra veiem que la informació està tabulada, i hi podem veure l'adreça en hexadecimal on es troba la instrucció en memòria (*columna Address*) i la instrucció assemblada en codi de màquina (*columna Code*). A més, ens assenyalarà realçat en groc quina és la instrucció que s'executarà.


1.4.4.1 Activitat 2

- Observeu la finestra *Text Segment*. En quina adreça s'emmagatzema cada instrucció del programa?
- Comproveu la codificació de les instruccions en codi de màquina.
- Observeu la finestra *Registers*. Quin és el valor del PC?

1.4.5 El cicle d'instrucció

Ha arribat el moment de provar el funcionament del programa que heu escrit. Per a fer-ho, seguiu els passos que indiquem a continuació:

1. Escriviu un valor en el registre \$8 en la finestra *Registers*.
2. Busqueu l'acció *Step*. (Run>Step, F7, ) i feu-la una vegada. Amb aquesta acció heu simulat un cicle d'instrucció! Noteu que el PC ha avançat i que el contingut del registre \$9 ha canviat. El seu contingut serà la suma del valor del registre \$8 més 25.

3. Completeu els dos cicles d'instrucció i confirmeu el resultat.
4. Busqueu l'acció *Reset*. (*Run>Reset*, F12, ) i doneu un valor inicial a \$8. Executeu ara el programa sencer *Run>Go*, F5)

1.4.5.1 Activitat 3

- Doneu el següent valor inicial $\$8 = 0x7FFFFFFF$ i torneu a executar el programa pas a pas (acció *Step*) fixant-vos en la finestra missatges del Mars, Mars Messages. Què ha ocorregut?

Qüestió 3

- Quin és el valor més gran que podrà contenir \$8 perquè no s'avorte el programa?

1.4.6 Usos alternatius d'*addi*

Podem utilitzar la instrucció *addi* per a usos diferents de la suma, com pot ser ficar una constant K en un registre R : *addi \$R,\$0, K* . O per a copiar el contingut d'un registre R en un altre registre T : *addi \$T,\$R,0*.

1.4.6.1 Activitat 4

- Modifiqueu el programa per a donar un valor inicial al registre \$8 utilitzant *addi*.
- Afegiu una instrucció perquè el resultat final es trobe en \$12.
- Podríem usar la instrucció *addi* per a fer una resta?

Qüestió 4

- Com s'escriu la instrucció que fa $\$8 = \$8 - 1$?
- Com en quedaria la codificació en binari?

1.4.7 Ajudes a la programació. Noms alternatius dels registres

El banc de registres de propòsit general del MIPS està format per 32 registres. Amb aquests registres es poden realitzar càlculs amb adreces i amb nombres enters. Recordeu que el MIPS també té altres bancs, però ara mateix no els estudiarem, ho deixarem per a més endavant. Certs registres del banc s'utilitzen per a propòsits molt específics facilitant

d'aquesta manera la compilació habitual de programes d'alt nivell. Aquest conveni és acceptat pels programadors en MIPS; és per això que ens resultarà molt convenient seguir aquestes regles o convenis d'utilització dels registres. Cada registre del banc té un nom convencional reconegut per l'assemblador i serà el que d'ara endavant utilitzarem. En la taula 1 els podeu veure

D'ara endavant, per al càlcul d'expressions qualssevol, feu servir els registres temporals \$t0...\$t7

Qüestió 5

- Reescriuiu el programa anterior utilitzant el conveni de registres i torneu-lo a executar.

Número del registre	Nom convencional	Utilització
\$0	\$zero	Sempre conté el valor 0.
\$1	\$at	Reservat per a l'assemblador
\$2 - \$3	\$v0 - \$v1	Utilitzats per a resultats i avaluació d'expressions.
\$4 - \$7	\$a0-\$a3	Utilitzats per a passar arguments a rutines.
\$8 - \$15	\$t0-\$t7	Temporals usats per a l'avaluació d'expressions (no preservats a través de crides a procediments)
\$16 - \$23	\$s0-\$s7	Registres guardats (preservats a través de crides a procediments)
\$24 - \$25	\$t8-\$t9	Més temporals (no preservats a través de crides a procediments)
\$26 - \$27	\$k0 - \$k1	Reservats per al nucli del sistema operatiu.
\$28	\$gp	Conté el punter global.
\$29	\$sp	Conté el punter de pila (stack pointer)
\$30	\$fp	Conté el punter d'enquadrament.
\$31	\$ra	Conté l'adreça de retorn usat en les crides a procediments.

Taula 1. Números dels registres, nom i utilització

1.4.8 Més sobre la suma immediata

En el apartat 1.4.5 heu vist que l'execució s'avorta quan es produeix desbordament (*overflow*) quan es fea la suma i la consola mostra el missatge `Error: Runtime exception at 0x0---: arithmetic overflow`. En certes ocasions pot interessar al programador que l'execució del programa continue sense avortar; en aquest cas és responsabilitat del programador prendre les accions oportunes si fóra necessari. El MIPS ens proporciona una instrucció que ho permet, és la instrucció `addiu rt,rs,K`

1.4.8.1 Activitat 5

- Torneu a escriure el programa canviant `addi` per `addiu` i doneu com a valor inicial de \$t0 el positiu més gran possible (`$t0=0x7FFFFFFF`) i executeu-lo observant el contingut de \$t1 en hexadecimal i en decimal. Què ha ocorregut?
- Si el programador considera que està operant amb números naturals, el resultat que hi ha en \$t1 seria correcte? Quin seria el seu valor en decimal?

Qüestió 6

- Escriu el codi que faci les següents accions utilitzant el conveni de registres i utilitzant la instrucció *addi*:

```
$12=5  
$10= 8  
$13=$12 + 10  
$10=$10 - 4  
$14=$13 - 30  
$15=$10
```

- Assembleu i executeu el programa i comproveu que el resultat final és $\$t7=\$t2=4$, $\$t6=-15$, $\$t4=5$, $\$t5=15$.

Qüestió 7

- Es podria escriure el mateix codi emprant la instrucció *addiu*? Fes la prova.
- Quin és el codi d'operació de la instrucció *addiu*?
- Codifiqueu en binari la instrucció *addiu \$v0, \$zero, 1*

1.5 Resum

- Les instruccions es codifiquen en binari, l'assemblador és l'encarregat de codificar-les.
- La memòria principal té adreces de 32 bits que seleccionen paraules de 32 bits.
- El banc de registres té 32 registres de 32 bits.
- Hi ha un cicle d'instrucció en què el processador processa la instrucció indicada pel registre PC.
- Hi ha instruccions que calculen la suma d'un registre amb una constant.