FONAMENTS DE COMPUTADORS

Pràctica 6
Introducció al simulador PCspim

OBJECTIUS

L'objectiu d'aquesta sessió de laboratori és conèixer el PCspim. Mitjançant l'arquitectura MIPS i el seu llenguatge d'assemblador ficarem en pràctica els coneixements bàsics sobre representació de les instruccions i del nombres enters, així com les operacions de multiplicació i divisió de nombres enters. Finalment escriurem un programa en assemblador.

INTRODUCCIÓ A L'ARQUITECTURA MIPS

El simulador PCspim és una aplicació que permet la simulació de l'execució de programes escrits en el llenguatge d'assemblador dels processadors amb arquitectura MIPS I (actualment un subconjunt de MIPS32) desenvolupada per l'empresa MIPS Technologies (**¡Error! No se encuentra el origen de la referencia.**). És possible trobar processadors MIPS en molts sistemes informàtics: consoles de videojocs PlayStation de 1988 (R3000) i Nintendo G64 de 1992 (R4000), impressores làser HP LJ4000 de 1996 (R5000) i estacions de treball com Silicon Graphics Indigo 2 de 1995 (R10000). En l'actualitat, altres dispositius empren processador amb aquesta arquitectura, com càmeres fotogràfiques i routers.

Figura 1. MIPS Technologies (http://imgtec.com/mips/)

El llenguatge d'assemblador és un llenguatge de baix nivell, extremadament pròxim al llenguatge màquina que pot executar el processador. Encara que el llenguatge d'assemblador és quasi el mateix que el llenguatge màquina, el llenguatge d'assemblador ofereix al programador ajudes i eines per facilitar el desenvolupament de programes. Algunes d'aquestes ajudes són la utilització de codis mnemònics per recordar les instruccions, etiquetes per indicar posicions de memòria, i directives per donar indicacions al programa assemblador.

És important no confondre **llenguatge d'assemblador** i **programa assemblador**. El programa assemblador és l'equivalent al compilador dels llenguatges d'alt nivell com el C o Java, és el programa encarregat de traduir el codi font a llenguatge màquina, que és el llenguatge que el processador entén i pot executar. El programa assemblador és l'encarregat de traduir codi en llenguatge d'assemblador a llenguatge màquina. Aquesta traducció es du a terme tenint en compte les directives que el programador, junt amb les instruccions d'assemblador, inclou en el codi del seu programa. Les directives serveixen, per exemple, per a seleccionar en quines posicions de memòria l'emmagatzemen les variables, o en quina posició de memòria està ubicada la primera instrucció del programa.

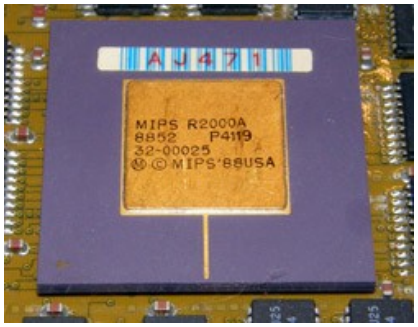


Figura 1 Encapsulat de un processador MIPS R2000

El PCspim incorpora, en una sola aplicació, un programa assemblador i un simulador del processador MIPS R2000 (Figura 1). El MIPS R2000 és un processador de 32 bits dissenyat en l'any 1986 per la companyia MIPS Computer Systems. Els primers processadors podien treballar a una freqüència de 15 MHz i comptaven amb poc més de cent mil transistors. Per a millorar el rendiment al treballar amb nombres reals representats en coma flotant, es podia instal·lar un xip extern, un coprocessador matemàtic, anomenat R2010 amb una FPU (*Floating Point Unit*), encarregant-se aquest xip de realitzar les operacions en coma flotant mitjançant circuits hardware, molt més ràpids que el processador simulant la coma flotant amb instruccions per a nombres enters. Aquesta solució també hi era disponible al primers PCs (Personal Computer) d'IBM, que utilitzaven com a processador l'Intel 8086, i que es podia complementar amb el coprocessador matemàtic 8087.

INTRODUCCIÓ AL SIMULADOR PCSPIM

Inici i configuració

El programa s'executa des de l'accés directe que hi ha al menú de programes.

Quan PCspim s'executa per primera vegada apareix la finestra de configuració següent:

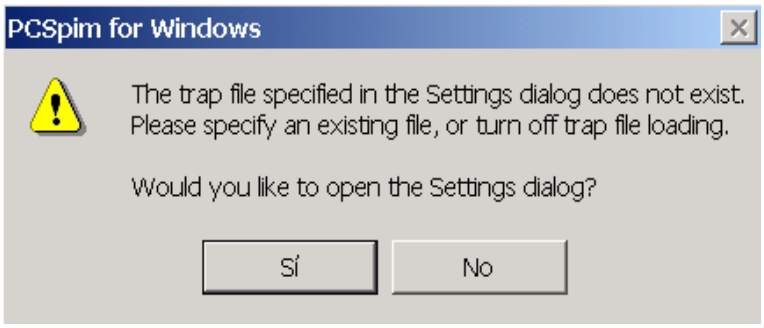


Figura 2 Finestra de diàleg que apareix la primera vegada que s'executa PCspim.

Feu clic sobre el botó Sí, per a configurar les opcions del simulador mitjançant l'ajuda de la finestra de diàleg següent:

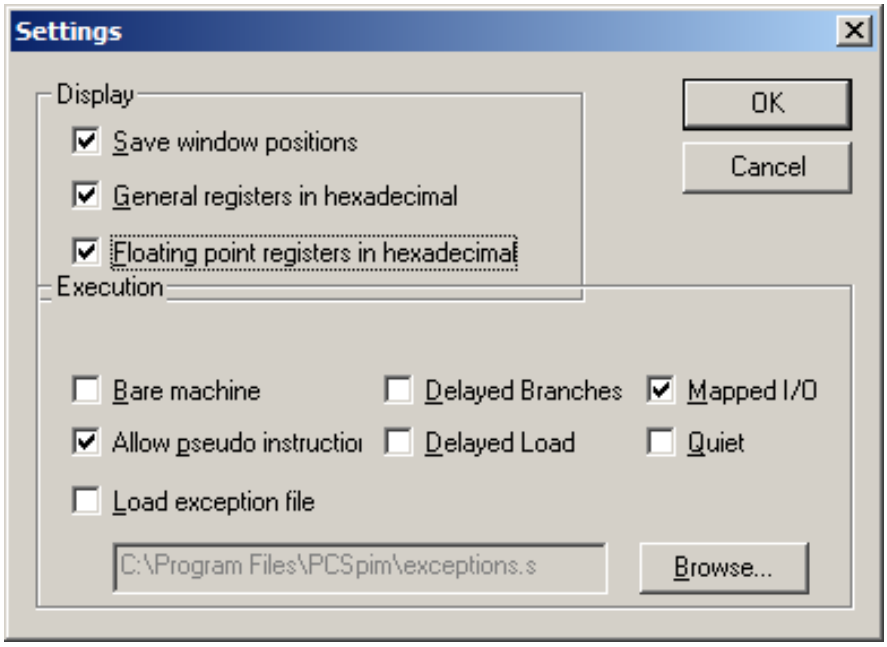


Figura 4. Configuracions recomanades.

Seleccioneu les opcions de manera que la configuració quede com mostra la Figura 4.

Les opcions principal són:

- La informació del registres de nombres enters (`$0...$31`) i de nombres reals (`$f0...$f31`) es mostren en hexadecimal. Com que tots els registres són de 32 bits, les dades es mostraran amb 8 dígits hexadecimal. L'alternativa a la visualització en hexadecimal és que el programa ens done el contingut dels registres en decimal.

- El programador pot utilitzar pseudoinstruccions, molt útils per a desenvolupar programes però que han de ser traduïdes a una o varies instruccions màquina.

Una vegada que PCspim ha estat configurat, feu clic sobre el botó OK. Llavors, es visualitzarà l'entorn de treball del simulador que es mostra en la figura 5. En la pantalla es mostra el resultat obtingut de carregar el programa contingut en el fitxer `helloword.s` que ve inclòs amb el simulador. Després de la càrrega d'un programa el simulador fica a zero la major part dels continguts dels registres i la memòria.

És important que dediqueu durant aquesta sessió de laboratori el temps necessari per a examinar la pantalla del PCspim i localitzar els elements més importants, per a poder traure el major rendiment a aquesta aplicació.

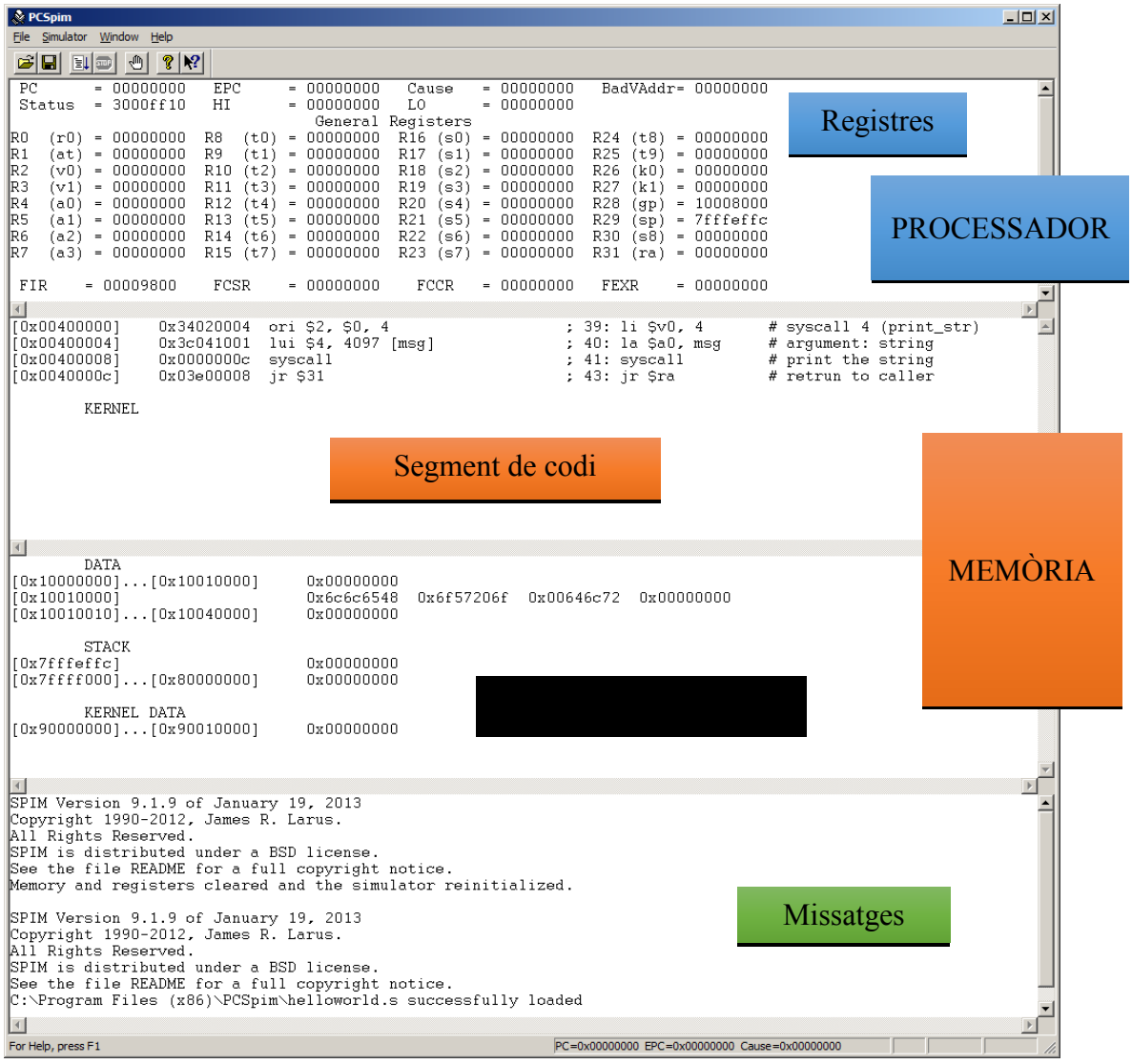


Figura 3 Entorn del simulador PCspim dividit en quatre seccions: registres del processador (nombres enters i reals), segment de codi (programa), segment de dades (variables) i àrea de missatges.

A continuació es descriuen amb detall les diferents seccions o finestres del PCspim:

Finestra de registres: mostra el valor emmagatzemat en els registres del MIPS. En aquesta finestra podem veure el comptador de programa (PC, *program counter)* i els registres `HI` i `LO` (registres especials que emmagatzemen el resultat de les multiplicacions i divisions de nombres enters..

En la part superior de la finestra podem veure els registres (*General Registers*) per a nombres enters, anomenats amb el símbol $ i un nombre del 0 al 31 (`$0`…`$31`); aquests registres també es poden referir per un nom alternatiu que ens permet donar-li una funció específica a cadascun d'ells. Per exemple, podem utilitzar l'identificador `$8` o també `$t0` (en aquest cas, la lletra *t* indica que el registre s'utilitza per a emmagatzemar dades de manera temporal).

Baix dels registres d'enters podem trobar (baixant la barra de desplaçament de la finestra) els registres per a nombres reals anomenats símbols `$f0`…`$f31`. Aquest registres es poden interpretar com 16 registres de precisió doble (*Double Floating Point Registers*) anomenats `$f0,$f2,$f4,…,$f30` o 32 registres de precisió simple (*Single Floating Point Registers*) anomenats `$f0,$f1,$f2,…,$f31`.

Finestra de segment de codi: en aquesta finestra es mostren quatre columnes.

1. En la primera columna es mostra en hexadecimal i entre claudàtors l'adreça on està emmagatzemada cadascuna de les instruccions del programa carregat en el simulador, una vegada assemblat.

2. En la segona columna es mostra en hexadecimal la codificació de cadascuna de les instruccions que componen el programa.

3. En la tercera columna es mostren els codis mnemònics d'aquestes instruccions.

4. Finalment, en la quarta columna, es mostren les instruccions corresponents tal com apareixen en el programa font. En alguns cassos, les pseudoinstruccions es tradueixen per més d'una instrucció.

En l'exemple de la Figura 3 la primera instrucció màquina del programa està emmagatzemada en l'adreça de memòria `0x00400000`, es codifica com `0x34020004`, el seu mnemònic és `ori $2,$0,4` i és el resultat de traduir la pseudoinstrucció del programa `li $v0,4` (acrònim de *load immediate*. Aquesta pseudoinstrucció s'utilitza per a inicialitzar registres amb constants).

Finestra de segment de dades: en aquesta finestra es mostra el contingut de la memòria. La memòria es desplega i mostra rangs d'adreces de quatre paraules (4 x 32 bits) entre claudàtors i el contingut en memòria d'aquest rang.

En l'exemple mostrat en la Figura 3 podem veure que el segment de dades comença en l'adreça `0x10010000`, el contingut de la qual és `0x6c6c6548` i correspon amb la cadena "`Hell`" codificada en ASCII. És important veure que els bytes estan ordenats en *little endian*, i per tant la "`H`" és en l'adreça `0x10010000`, la "`e`" en la `0x10010001`, la primera "`l`" en la `0x10010002` i així successivament. Com que en aquesta pràctica els programes no fan ús de la memòria de dades aquesta finestra mostrarà tot zeros en els seues continguts.

Finestra de missatges: en aquesta finestra podem veure informació diversa, com la versió del simulador, o que el programa ha sigut correctament carregat i assemblat, o missatges d'error.

Simulació de l'execució de programes

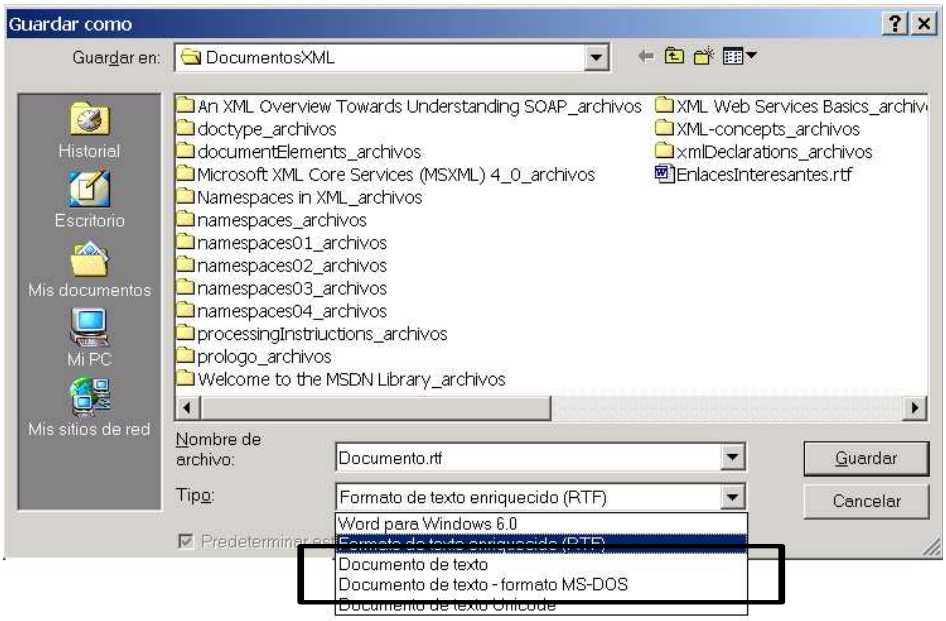
Els passos que cal seguir per a utilitzar el simulador són:

- Escriure un programa en llenguatge d'assemblador del MIPS R2000 utilitzant qualsevol editor de textos, però ***assegureu-vos d'emmagatzemar el fitxer en un document de text sense format amb extensió .s o .asm.***
- Carregar el programa en el simulador (opció *File* | *Open*...).
- Seguir pas a pas l'execució de les instruccions (F10) i observar amb detall l'evolució del valor dels registres.

1. Edició del programa

NOTES IMPORTANTS sobre l'edició d'un programa:

- El fitxer s'ha d'escriure en format de text. La millor opció és utilitzar el Notepad++. Si utilitzeu el programa WORDPAD o el programa WORD assegureu-vos que emmagatzemeu el fitxer com a **document de text** o com a **document de text en format MS-DOS**.



- Si empreu l'editor WORDPAD, assegureu-vos d'emmagatzemar el fitxer en format de text i canviar l'extensió .txt per .s o .asm
- L'extensió del fitxer ha de ser .s o .asm.
- Cal recordar que l'etiqueta predefinida *__start* comença amb dos caràcters de subratllat.

Figura 4. Selecció del tipus de format per Salvar un fitxer

2. Càrrega del fitxer amb el programa

Per a realitzar la càrrega d'un programa, s'utilitza el menú *File*, el qual té les opcions següents:

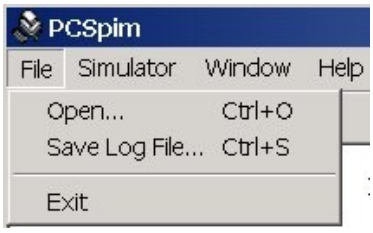


Figura 2. Opcions del menú *File*

L'opció *Open* és la que ens permet carregar fitxers en el simulador. El procés de càrrega inclou l'assemblatge del programa i la càrrega del codi màquina en la memòria. Si el codi font presenta cap error de sintaxi, el simulador no podria traduir-lo a codi màquina i per tant tampoc carregar-lo en la memòria del computador.

L'opció *Save Log File* té com a finalitat emmagatzemar en un fitxer els missatges que han aparegut en la finestra de missatges. El seu objectiu és el de poder analitzar les sortides obtingudes en executar el programa i detectar possibles causes de funcionaments inadequats.

Finalment, l'opció *Exit* acaba l'execució del simulador.

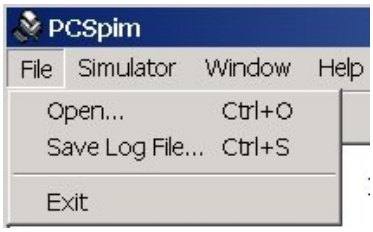


Figura 5 Opcions del menú file

3. Simulació de l'execució del programa

Amb les diferents opcions del menú *Simulator* és possible simular l'execució d'un programa carregat prèviament. En aquesta sessió farem ús de les opcions següents:

Go (F5). Realitza l'execució completa del programa, des de la primera fins a l'última instrucció.
Single Step (F10). Permet executar pas a pas, és a dir, instrucció per instrucció, tot el programa. Aquesta opció és molt útil per observar com evolucionen els registres i les variables a l'executar cada instrucció.
Set Value. Permet donar o assignar valor a un registre o posició de memòria.

OPERACIONS AMB ENTERS REPRESENTATS EN COMPLEMENT A 2

Teclegeu el codi següent en un editor (per exemple el Notepad++), graveu-lo en un fitxer amb el nom `ejemplo1.s` i carregeu-lo en el PCspim.

```
            .text 0x00400000# Adreça inici codi
            .globl __start        # Etiqueta global
__start:
            addi $8,  $0,  1      # Inicialitza $8 amb 1
            addi $9,  $0, -2      # Inicialitza $9 amb -2
            add  $10, $8, $9      # $10 <- $8 + $9
            .end                  # Final del programa
```

Pregunta 1. Què és el que fan les dues instruccions `addi`?

Son de tipus inmediat i suman un valor a un altre.

Pregunta 2. Quin valor, expressat en decimal, és el que s'espera que continga el registre `$10` en acabar l'execució del programa?

-1

Pregunta 3. Ompliu la taula següent amb el contingut mostrat en la finestra de registres després de carregar el programa i abans d'executar-lo. És important que anoteu tots els dígits hexadecimals per tenir una idea de les longituds de les paraules de bits.

Registre	Valor en hexadecimal (amb tots el bits)	Valor en decimal
`$8`	00000000	0
`$9`	00000000	0
`$10`	00000000	0

Pregunta 4. Ara executeu el programa emprant l'opció *GO* del menú *SIMULATOR*. Una vegada finalitzada l'execució, ompliu la taula següent amb el nou contingut dels registres.

Registre	Valor en hexadecimal (amb tots el bits)	Valor en decimal
`$8`	00000001	1
`$9`	fffffffe	-2
`$10`	ffffffff	-1

Pregunta 5. Quantes instruccions màquina té el codi del programa?

Tres, dos addi i una add.

Pregunta 6. Quants bytes de memòria ocupa el codi màquina del programa?

Hi han 12 bytes

Pregunta 7. En quina adreça de memòria s'emmagatzema la instrucció `addi $8,$0,1`? Quina és la codificació d'aquesta instrucció?

En $8, i la codificacion 0x20080001

Pregunta 8. En quina adreça de memòria s'emmagatzema la instrucció `add $10,$8,$9`? Quina és la codificació d'aquesta instrucció?

$10

Pregunta 9 Quina és la funció o utilitat de les directives utilitzades en el programa?

Sumar valores .globl declara que la etiqueta __ es global y el .tex indica el lugar donde se van

a empezar a copiar los datos.

LA MULTIPLICACIÓ DE NOMBRES ENTERS

El codi següent és un programa que utilitza la instrucció de multiplicació.

```
            .text 0x00400000      # Adreça inici codi
            .globl __start        # Etiqueta global
__start:
            addi $8, $0, 10000    # Inicialitza $8 amb 10^4
            mult $8, $8           # HI-LO <- $8 * $8
            mflo $9               # $9 <- LO
            mult $8, $9           # HI-LO <- $8 * $9
            .end                  # Final del programa
```

A més d'utilitzar la instrucció de multiplicació, es fa servir una instrucció de moviment de dades entre el banc de registres d'enters (`$0…$31`) i el registre especial `LO` de la unitat aritmeticològica. Aquesta instrucció és `mflo`, acrònim de *Move from LO*.

És important recordar que en el cas de la multiplicació aritmètica el resultat pot ocupar el doble de bits que la grandària del operands. En aquest cas, com que els operands ocupen 32 bits cadascun, el resultat por arribar a ocupar 64 bits, per això el resultat de la multiplicació s'emmagatzema en els registres HI i LO.

Editeu el programa amb el Notepad++ i graveu-lo com `ejemplo2.s`. A continuació carregeu-lo en el PCspim.
Si no hi ha errors en la carrega, executeu el programa PAS a PAS fins que es complete la segona instrucció del programa mult $2, $2). Per fer l'execució pas a pas cal polsar dues vegades la tecla F10, fins que quede marcada amb blau la instrucció mflo $3. El PCspim marca amb blau la instrucció SEGÜENT a executar.

Pregunta 10. Ompliu la taula següent al finalitzar l'execució de les dues primeres instruccions (`add`, `mult`). Escriviu tots els dígits hexadecimals.

Registre	Valor en hexadecimal (els 32 bits)	Valor en decimal
HI		
LO		

Pregunta 11. Quin és el resultat de la multiplicació? Expresseu el resultat en hexadecimal (64 bits) i en decimal.

Valor en hexadecimal (els 64 bits)	Valor en decimal

Pregunta 12. Executeu ara, polsant una vegada més la tecla F10, la instrucció `mflo $9`. Què és el que fa aquesta instrucció?

Pregunta 13. Comproveu que ara el registre `$9` conté la part baixa del resultat de la primera multiplicació. Es pot emmagatzemar el resultat complet en 32 bits? Per què?

Pregunta 14. Finalment, completeu l'execució del programa polsant una quarta vegada la tecla F10 per a executar l'última instrucció. En quins registres trobarem el resultat de la multiplicació? Quin és el valor que s'ha obtingut? És possible emmagatzemar-lo en 32 bits?

Valor en hexadecimal (64 bits)	Valor en decimal

LA DIVISIÓ DE NOMBRES ENTERS

El programa següent fa ús de la instrucció de divisió entera. La divisió entera genera dos resultats: d'una banda el quocient de la divisió, i d'altra banda, el residu.

```
            .text 0x00400000      # Adreça inici codi
            .globl __start        # Etiqueta global
__start:
            addi $8, $0, 43       # Inicialitza $8 amb 43
            addi $9, $0, 12       # Inicialitza $9 amb 12
            div  $8, $9           # Divideix $8 entre $9
            mflo $10              # LO -> $10
            mfhi $11              # HI -> $11
            .end                  # Final del programa
```

Editeu el programa amb Notepad++ i graveu-lo amb el nom `ejemplo3.s` y carregeu-lo en el PCspim. Executeu el programa pas a pas i observeu com canvien els valors emmagatzemats en els registres que utilitza el programa.

Pregunta 15. En quin registre s'emmagatzema el quocient de la divisió? I el residu?

Pregunta 16. Quina és la utilitat de les instruccions `mflo` y `mfhi` en aquest programa?

Pregunta 17. Canviaria el resultat del programa si eliminem les dues instruccions anteriors?

Pregunta 18. En quina adreça de memòria s'emmagatzema l'última instrucció del programa?

Pregunta 19. Quants bytes ocupa el programa en la memòria del computador?

DISSENY D'UN PROGRAMA EN LLENGUATGE D'ASSEMBLADOR

Emprant els continguts treballats en aquesta sessió de pràctiques, dissenyeu un programa en llenguatge d'assemblador equivalent al codi de llenguatge d'alt nivell següent. L'operador `%` correspon a l'operació mòdul, que és el residu de la divisió entera..

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

a = 20;
b = -8;
c = (a*15)/(a-b);
d = (a*15)%(a-b);
```

Pregunta 20. Calculeu emprant la calculador, el valor resultant en les variables `c` i `d`.

Pregunta 21. Escriviu el codi assemblador en el Notepadd++, graveu-lo amb el nom `ejemplo4.s` i carregeu-lo en el PCspim. Executeu el programa i comproveu que el resultat coincideix amb el càlculs de la pregunta anterior.
Donat que encara no sabem declarar variables en assemblador, utilitzarem registres per a emmagatzemar els valor de les variables. En concret, guardarem `a` en `$8`, `b` en `$9`, `c` en `$10` i `d` en `$11`. Si necessiteu més registres podeu emprar `$12..$15`.

Pregunta 22. Indiqueu les adreces de la primera i l'última instrucció del programa.

Pregunta 23. Quantes paraules de 32 bits ocupa el programa? I quants bytes ocupa el programa?