PRÀCTIQUES DE L'ASSIGNATURA

# INTRODUCCIÓ ALS ORDINADORS

UNIVERSITAT DE BARCELONA

# Índex

# *1. Normes de les Pràctiques*

## Avaluació

Les pràctiques d'Introducció als Ordinadors són una part integrant de l'assignatura. Aquestes pràctiques es realitzen a l'aula IE de la facultat de Matemàtiques i comporten la realització de 10 pràctiques avaluables, un miniprojecte i un examen pràctic final. El resultat de l'avaluació d'aquestes pràctiques constitueix el 50% de la nota de l'assignatura. Per poder aprovar l'assignatura la nota mitjana de pràctiques ha de ser superior o igual a 5 i caldrà presentar-se a totes les sessions i entregar tota la documentació requerida.

## Sessions Pràctiques

L'assignatura consta de un total de 10 pràctiques dividides en 10 sessions i un miniprojecte com a treball a realitzar per l'alumne a casa. S'establiran períodes d'entrega a través del Campus Virtual de l'assignatura. Si l'entrega no es realitza en el període establert, la pràctica constarà com suspesa.

## Normativa

- Els alumnes treballaran a l'aula individualment (si hi ha ordinadors suficients). En cas de que es disposi d'un ordinador portàtil propi, pot portar-se a l'aula per a la realització de les pràctiques si el alumne així ho vol.
- Les pràctiques es realitzaran utilitzant el sistema operatiu Windows XP. La contrasenya d'entrada és l'assignada per accedir als ordinadors de la facultat
- Queda totalment prohibit instal·lar o utilitzar programes propis als ordinadors de l'aula
- No està permesa la utilització dels ordinadors per realitzar treballs d'altres assignatures durant les pràctiques.

## *Pràctica 1: Introducció a la Màquina SIMR*

(1 sessió)

## Objectius

Familiaritzar-se amb el simulador de la màquina rudimentària (SiMR)

Entendre què fan les instruccions

Comprendre l'analogia entre llenguatge màquina i el llenguatge assemblador

## Introducció Teòrica

SIMR és un simulador on hi ha un processador RISC i una memòria principal. És una eina que permet aprendre els conceptes bàsics sobre estructura i arquitectura de processadors. L'arquitectura del processador es tan senzilla com elegant, amb 8 registres de propòsit general, un que sempre val zero i on no es permet l'escriptura de cap valor (R0) i la resta dels 7 registres on es poden fer lectures, escriptures i desplaçament cap a la dreta. Com a registres específics tenim:

1. l'acumulador (RA), registre encarregat de guardar les operacions realitzades a la ALU.
2. El registre d'estat (RNZV), on tenim els flags de zero, overflow i negatiu
3. El registre d'instruccions (IR)
4. El registre comptador de programa (PC) per adreçar la memòria
5. Banc de registres. 8 registres de propòsit general on R0 sempre val 0.

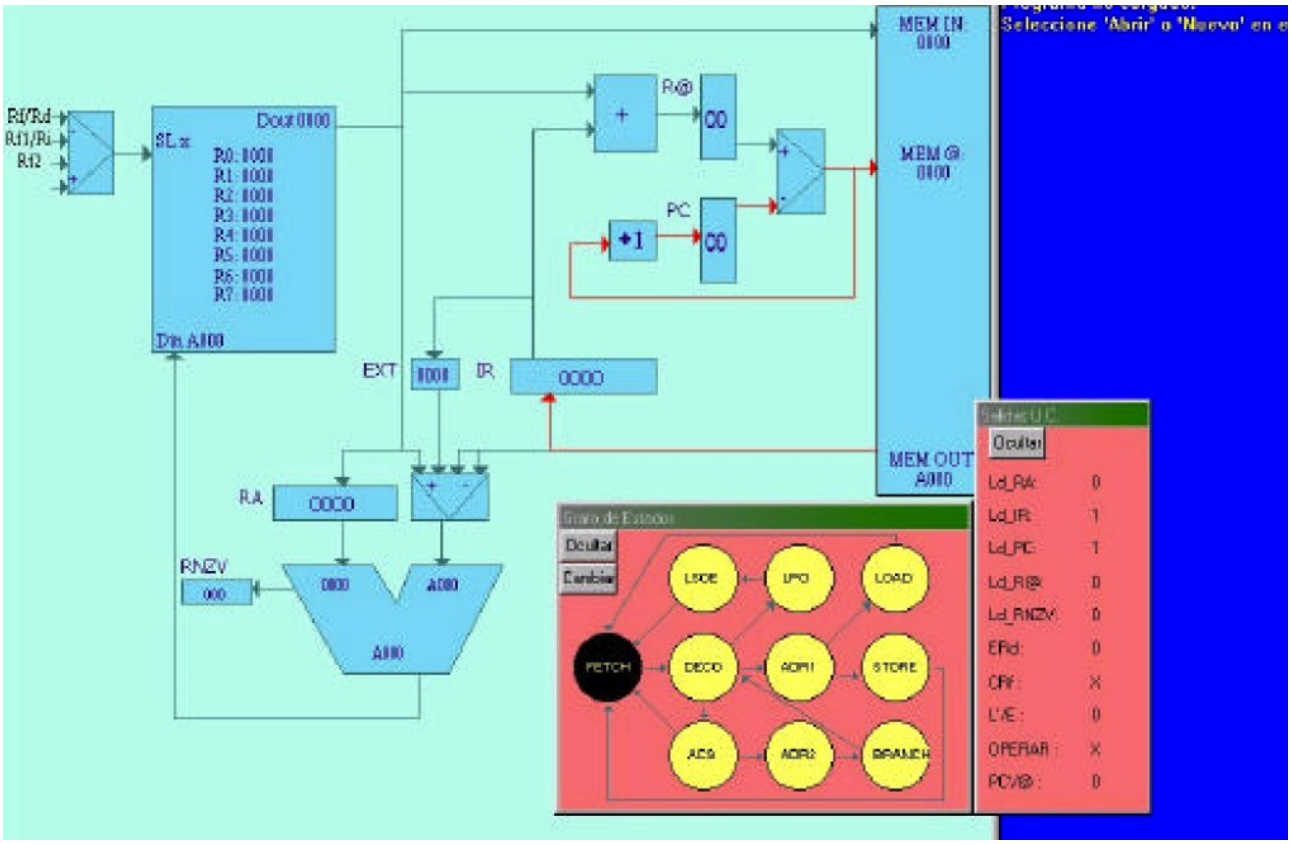


Figura 1. Diagrama del simulador SiMR

Tenim a més un banc de memòria (zona blava més fosca) on es carrega el programa a executar, una

sèrie de multiplexors per seleccionar els registres, entrada en ALU i sortida cap a memòria i una unitat de control on es mostra l'execució de les instruccions.

El professor de pràctiques us explicarà detalladament el funcionament d'aquest simulador.

A nivell de software tenim parts ben diferenciades: (i) les directives de programació, (ii) les etiquetes, (iii) els comentaris i, finalment (iv) el set d'instruccions.

i) Directives de programació: Les directives, també anomenades pseudo-operacions o pseudo-ops són instruccions que s'executen en temps d'ensamblat i no per la CPU en temps d'execució. Permeten fer l'ensamblat del programa depenent de paràmetres introduïts pel programador, de tal manera que el codi pugui ser ensamblat de diferents formes tenint en compte diferents aplicacions. Indiquen per tant a l'ensamblador detalls necessaris per portar a bon terme el procés d'ensamblatge. Per exemple, una directiva pot indicar a l'ensamblador la posició de memòria on ha d'estar localitzat el programa. Tenim 4 directives

    1. Directiva .dw -> Serveix per assignar a una determinada posició de memòria un determinat contingut .dw 7,4
    2. Directiva .rw -> Serveix per reservar n espais de memòria. .sw 3 reservaria 3 espais de memòria.
    3. Directiva .begin -> Indica on comença la memòria de programa.
    4. Directiva .end -> Indica on acaba la memòria de programa.

ii) Etiquetes. Serveixen per apuntar de forma més amigable a una determinada posició del programa.

    Etiqueta: .dw 7 -> Indica que en l'adreça de memòria Etiqueta tinc un 7 guardat

iii) Comentaris. A diferència de altres llenguatges de programació com C o Java, en ensamblador els comentaris s'indiquen amb un ";"

iv) El conjunt d'instruccions: Una instrucció en ensamblador és una representació simbòlica de codi màquina binari necessària per realitzar una sèrie de operacions per part de la CPU.

    ADD R1,R2,R3 -> 11 011 001 010 00 100 -> R1 + R2 => R3

    El conjunt d'instruccions d'un processador ens reflexa directament la seva arquitectura. La implementació d'un algorisme per tal de solucionar un problema implica la realització d'un programa que es realitzarà tenint en compte les instruccions que pot interpretar l'ensamblador. La majoria dels processadors tenen el mateix tipus de grups d'instruccions, tot i que no necessàriament han de tenir el mateix format ni el mateix nombre d'instruccions per a cada grup. De fet, cada arquitectura té el seu propi llenguatge màquina i en conseqüència el seu propi conjunt d'instruccions.

Exemple de tot plegat:

```
; Aquest és el primer programa de Introducció als Ordinadors
; Autor: alumne
memoriaDades: .dw 7,4,3,2        ;aquest simulador sempre inicia en la posició 0 de memòria
                                 ;per tant, memoriaDades equival a la posició 0 de memòria, on
```

```
                                ;tinc un 7, a la posició 1 tinc un 4, a la posició 2 tinc un 3,...
reservaMemoria: .rw 4           ;reservo 4 posicions de memòria
.begin iniciPrograma
iniciPrograma:
                ;TODO
.end
```

El programa pot escriure's en qualsevol editor de text, però s'ha de guardar amb l'extensió .asm. Si feu servir el simulador compatible amb la versió windows XP en endavant, haureu de cridar l'executable

POSTEN nomPrograma.asm

això crearà un fitxer compilat que podrà executar-se amb la màquina virtual.

Si teniu instal·lat Wine, podeu executar la versió SIMR3.0. Aquesta versió incorpora un editor propi que genera el fitxer executable. A pràctiques treballarem amb la primera versió.

## Instruccions de SIMR

Dividim el conjunt d'instruccions en els seguents subgrups:

1. Instruccions aritmètic – lògiques
2. Instruccions d'accés a memòria
3. Instruccions de salt

**Instruccions Aritmètic – lògiques**

| | |
|---|---|
| ADD R1, R2, R3 | ; Suma el contingut de R1 + el contingut de R2 i ho guarda en R3 |
| SUB R1,R2,R3 | ; Resta el contingut de R1 – el contingut de R2 i ho guarda en R3 |
| ADDI R1, #nombre, R2 | ; Suma el contingut de R1 + el nombre i ho guarda en R2 |
| SUBI R1, #nombre, R2 | ; Resta el contingut de R1 – el nombre i ho guarda en R2 |
| AND R1, R2, R3 | ; Fa una AND dels continguts de R1 i R2 i ho guarda en R3 |
| ASR R1,R2 | ; Fa un desplaçament de 1 bit cap a la dreta de R1 i ho guarda en R2 |

**Instruccions d'accés a memòria**

on valor és un nombre o una etiqueta. Per exemple el cas

| | |
|---|---|
| LOAD valor(R2), R1 | LOAD AdreçaMemòria, desaria el contingut que hi ha a la posició de memòria valor + contingut de R2 en el registre R1 |
| STORE R1, valor(R2) | Guarda el contingut de R1 en la posició de memòria valor + contingut de R2 |

**Exemples:**

valors inicials R4 = 4, R1 = 3 @7 = 10

LOAD 3(R4), R1

A R1 guardem el contingut de 3 + 4 = 7. A l'adreça 7 tenim un 10 => R1 <= 10

STORE R1, 5(R0)

Ara R1 guarda un 10, guardem en la posició de memòria 5 + contingut de R0 (que sempre és 0) el 10 => @5 <=10

**Instruccions de salt**

| | |
|---|---|
| BR posicioMemoria | salt incondicional a la posició de memòria posicioMemoria |
| BEQ posicioMemoria | si el bit de zero del registre d'estat esta a 1, salta a la posició de memòria posicioMemoria |
| BG posicioMemoria | si la operació anterior dona un valor positiu, salta a la posició de memòria posicioMemoria |
| BL posicioMemoria | si la operació anterior dona un valor negatiu, salta a la posició de memòria posicioMemoria |
| BLE posicioMemoria | si la operació anterior és zero o negativa, salta a la posició de memòria posicioMemoria |
| BGE posicioMemoria | si la operació anterior és zero o positiva, salta a la posició de memòria posicioMemoria |

## *Exercicis guiats*

***1.*** Indiqueu quines de les següents instruccions són incorrectes segons les especificacions de la Màquina Rudimentària (MR), expliqueu el perquè en cada cas:

| | Instruccions | Correcte/Incorrecte | Motiu |
|---|---|---|---|
| a | LOAD R1(R0), R1 | incorrecte | |
| b | STORE R1,R1(3) | incorrecte | |
| c | BG 6(R1) | incorrecte | |
| d | ADDI R2, #11, 5(R3) | incorrecte | ;ADDI R2, #11, R3 |
| e | SUB R0,R2,R3 | correcte | |
| f | LOAD 3(R0),R1 | correcte | |

***2.*** Suposeu que la màquina rudimentària té els següents valors en alguns registres i posicions de memòria (recordeu que amb la lletra "h" s'indica que el número està en format hexadecimal):

-Registres: R0=0000h, R1=0002h, R2=A5E3h, R3=F412h

-Bits de condició (recordeu que N és el bit de Negatiu, Z és el bit de resultat igual a zero): N=0, Z=1

-Memòria: M[0Ch]=F45Ah i M[0Dh]=0033h

Indiqueu què fa cadascuna de les instruccions següents, cal explicitar totes les alteracions que es produeixen a la memòria, bits de condició, registres i valor final. Per cada una de les instruccions sempre partiu de les condicions inicials descrites més a dalt.

| | Instruccions | R0 | R1 | R2 | R3 | N | Z | M[0Ch] | M[13h] | PC |
|---|---|---|---|---|---|---|---|---|---|---|
| a | LOAD 12(R0),R1 | igual | F45Ah | igual | igual | 1 | 0 | igual | igual | +1 |
| b | STORE R1,1(R3) | igual | igual | igual | igual | 1 | 0 | igual | igual | +1 |
| c | BR 33 | "" | "" | "" | "" | "" | "" | "" | "" | 34 |
| d | ADDI R2,#11,R3 | "" | "" | "" | R2+11 | 1 | 0 | "" | "" | +1 |
| e | SUB R0,R2,R3 | "" | "" | "" | R2 | 0 | 0 | "" | "" | +1 |
| f | ASR R1,R1 | "" | R1/2 | "" | "" | 0 | 0 | "" | "" | +1 |

Breu descripció del que fa cada instrucció:

***a.***

***b.***

***c.***

***d.***

***e.***

***f.***

***3.*** Escriviu en llenguatge màquina (segons les especificacions de la MR) el següent programa, feu-ho en hexadecimal i binari:

| @ | M[@] | LLENGUATGE MÀQUINA | LLENG. MAQ. (hex) |
|---|---|---|---|
| 05h | SUB R1,R1,R1 | | |
| 06h | ADD R0,R0,R2 | | |
| 07h | SUBI R1, #4,R0 | | |
| 08h | BEQ 13 | | |
| 09h | LOAD 0(R1),R3 | | |
| 0Ah | ADD R3,R2,R2 | | |
| 0Bh | ADDI R1,#1,R1 | | |
| 0Ch | BR 07 | | |
| 0Dh | STORE R2,4(R0) | | |

## Realització Pràctica Guiada

***1.*** Escriure les instruccions de l'apartat 1 de l'estudi previ en un programa al SiMR (no cal que implementi cap funcionalitat) i intenteu compilar-lo. Podreu comprovar com les instruccions

incorrectes us donen error alhora de compilar. Feu les modificacions adequades per a que no us doni cap error en la compilació (recordeu que no es demana cap funcionalitat).

La forma de crear un programa amb el SiMR (en la seva versió 3.0) és el següent:

1. Executar el programa SiMR3.0.exe
2. Continuar sense identificació
3. Fer “Archivo _ Nuevo”
4. Escriure el programa en llenguatge Assemblador
5. Alhora de desar el programa donar-li l’extensió *.asm

Si feu servir la versió SiMR 2.0 (la que hi ha disponible en pràctiques) feu el següent:

Obriu l'editor de text de windows

Escriure el programa en llenguatge assemblador

Alhora de desar el programa poseu l'extensió *.asm

Compileu el programa fent POSTEN nomPrograma.asm

Un exemple de com escriure les instruccions anteriors en un programa és el següent:

```
.begin inici
inici:
        LOAD R1(R0), R1
        STORE R1, R1(3)
        BG 6(R1)
        ADDI R2, #11, 5(R3)
        SUB R0, R2, R3
        LOAD 3(R0), R1
.end
```

Per “compilar” el programa en SiMR 3.0 només cal fer “Compilador _ Compilar”. No és una compilació de llenguatge d’alt nivell a codi màquina, sinó ensamblar.

***2.*** Pel programa de l’apartat 3 de l’estudi guiat, les posicions de memòria que van des de la 00h a la 03h, ambdues incloses, contenen una seqüència del quatre números 3, 5, 2, 8, emmagatzemats de forma consecutiva. Aquesta seqüència de valors no està especificada al programa anterior. La posició 00h conte el primer element, la posició 01h el segon element, etc.. El programa descrit anteriorment es troba emmagatzemat a partir de la posició de memòria 05h.

Partint d’aquesta descripció responeu a les següents preguntes:

a) En quina posició de memòria escriu aquest programa el resultat?

b) Què fa el programa?

c) Quina sentència de control de flux (en llenguatge d’alt nivell) implementa el

programa?

d) Quina és la funció d’R1 al programa? I la d’R0?

Recordeu que aquest és el programa:

| @ | M[@] |
|---|---|
| 05h | SUB R1,R1,R1 |
| 06h | ADD R0,R0,R2 |
| 07h | SUBI R1, #4,R0 |
| 08h | BEQ 13 |
| 09h | LOAD 0(R1),R3 |
| 0Ah | ADD R3,R2,R2 |
| 0Bh | ADDI R1,#1,R1 |
| 0Ch | BR 07 |
| 0Dh | STORE R2,4(R0) |

***3.*** Realitzeu les següents accions sobre el Simulador de la màquina rudimentària (SiMR) i expliqueu-ne els resultats:

e) Arranqueu el simulador de la MR y carregueu el programa pract1.asm. Compileu-lo. Carregueu ara el programa pract1.cod. Estabiu una relació entre el codi assemblador del programa presentat pel simulador i el de l'apartat anterior.

f) Observeu que inicialment PC=05h (el PC apunta a la primera instrucció executable del programa). Esbrineu com ho fa.

g) Executeu el programa, instrucció per instrucció, observant el resultat d'executar cada una de les instruccions:

-Abans d'executar cada instrucció prediu les variacions que es produiran al banc de registres i a la memòria.

-Comproveu que el resultar de l'execució del programa coincideix amb el que havíeu previst.

-Per cada instruccions anoteu tots els canvis.

## Informe

Realitzeu un informe de la pràctica explicant el funcionament del simulador utilitzat.

## *Pràctica 2: Exercicis amb la Màquina SIMR*

Aquesta pràctica és novament una sessió guiada pel professor.

El professor recordarà el conjunt d'instruccions, el funcionament del simulador i l'adreçament a memòria.

## Objectius

L'objectiu d'aquesta pràctica és familiaritzar-nos amb el funcionament dels registres que hi ha a la CPU. Per tal d'assolir aquest objectiu, farem servir el simulador SIMR.

Com s'ha explicat a teoria, dividim el conjunt de registres fonamentalment en dos tipus:

- Registres de propòsit general
- Registres específics

Els registres de propòsit general es troben al banc de registres. Són 8 registres [R0 : R7], el registre R0 sempre val 0, mentre que la resta, el seu contingut és variable. Així una bona forma d'inicialitzar un registre (posar a 0 el seu contingut) serà per exemple:

ADD R0,R0, R1 => Suma el contingut de R0 + el contingut de R0 i es desa en R1

Els registres de propòsit específic són:

- El registre Acumulador, que en el nostre simulador el tenim en una de les entrades de la ALU
- El registre d'estat, que el tenim connectat a la ALU per detectar valors negatius, zero,i OV
- El registre d'Instruccions, on es guarda la instrucció a executar
- El registre Program Counter, on tenim l'adreça de la següent instrucció a executar.
- Un registre específic per guardar les adreces en cas de salt, al costat del PC

## Exercici 1 (Guiat)

Carrega el següent programa en el simulador SIMR

```
Dades: .dw 4, 5, 6, 7          ;aquí tenim les dades per operar amb els registres
.begin inici                   ;directiva d'inici de programa
inci:
        SUB R7, R7, R7
        LOAD Dades(R0), R1
        ADDI R7, #1, R7
        LOAD 0(R7), R2
        ADDI R7, #1, R7
        LOAD 0(R7), R3
        ADDI R7, #1, R7
```

```
        LOAD 0(R7), R4
loop:
        ADD R1, R2, R5
        ADD R3, R4, R6
        SUBI R3,#1, R3
        BG loop
.end                                ;directiva final de programa
```

## Exercici 2 (Guiat)

Feu un programa similar al de l'exercici 1. Carregueu al registre 1 el valor 0000110000001011b. Carregueu al registre 2 el valor 0000000000010001b. Feu l'operació R7<= R7+R1 y decrementeu el valor de R2. Feu això en un bucle fins que el valor contingut en R2 sigui 0.

## Exercici 3 ( a fer per l'alumne)

Ens demanen calcular un algoritme que ens faci el següent: Donades dues entrades enmagatzemades en les posicions de memoria A i B, fer la comparació. Si A>B calculem la suma. Si B>A fem la diferència (B-A) i si són iguals que finalitzi l'algoritme.

## Informe

Expliqueu-ne la feina realitzada en aquesta pràctica.

A l'exercici 1:

i) Un cop acabat el programa quant val el contingut de R5?

ii) Un cop acabat el programa quant val el contingut de R6?

iii) Un cop acabat el programa quant val el contingut de R3?

A l'exercici 2:

i) Quant val el contingut de R7 quan acaba el programa?

ii) Quantes vegades s'executa el codi?

iii) Al final del programa, just abans del .end, poseu la instrcció STORE R7, 0(R2). En quina posición de memòria es guarda el resultat?

A l'exercici 3:

i) Expliqueu el funcionament del programa

ii) Feu que el resultat es guardi a la posició de memòria 22h

## *Pràctica 3: Operacions i bucles amb SIMR*

(1 sessió al laboratori + 1 treball a casa )

## Objectius

Aquesta pràctica té com a principal objectiu la utilització de sentències de control de flux, generació de bucles i salts condicionals i incondicionals. Per altra banda, un altre objectiu és solidificar els coneixements adquirits per part de l'alumne en les pràctiques anteriors.

## Problema 1

Executa el següent codi, pren les notes pertinents per tal de respondre les qüestions de l'informe. Pren especial atenció al funcionament dels bucles amb les sentències de salt associades.

```
Contador: .dW 5
Valor1: .dW 6
Resultat: .rW 1
.begin inici
Inici:
ADD R0,R0,R1
SUB R0,R0,R2
LOAD Contador(R0), R3
LOAD Valor1(R1), R2
Loop:
ADD R1, R2,R1
SUBI R3, #1,R3
BG Loop
STORE R1, Resultat(R0)
.end
```

## Problema 2

A partir del programa que hi ha implementat al problema 1, feu un programa que permeti fer la multiplicació de dos números sencers, tant positius com negatius. Considereu la opció de tenir CARRY i OVERFLOW.

L'algorisme de la multiplicació es pot implementar fent servir sumes iteratives. Així, si volem multiplicar per exemple 3x4 faríem:

```
ADDI R0, 4, R1
ADDI R0, 3, R2
loop:
    ADD R1, R3, R3
    SUBI R2, 1, R2
    BG loop
```

## Problema 3

Feu un programa que calculi el Ca1 del contingut d'una determinada posició de memòria.

## Informe

Expliqueu-ne el funcionament de la pràctica.

Problema 1

i) Quin és el valor final que hi ha a R1?

ii) Quin bit s'activa per surtir del bucle?

iii) Quina operació estem fent en aquest algoritme?

iv) A que equival Valor1?

Expliqueu detalladament els problemes 2 i 3

## *Pràctica 4: Microinstruccions en SIMR*

(1 sessió)

Explicació prèvia per part del professor del concepte de microinstruccions.

## Objectiu

L'objectiu de la pràctica és visualitzar els diferents passos que ha de fer un microprocessador per tal d'executar una instrucció.

## Introducció

El número de cicles dividit per la freqüència de funcionament del processador ens defineix el temps d'execució de la instrucció. Les diferents accions que es realitzen en cada cicle forma el que és coneix com microinstrucció. Com a norma general, les diferents instruccions que conformen el conjunt d'instruccions varien des de aquelles que s'executen en pocs cicles de rellotge i ocupen poc espai en memòria i aquelles instruccions grans, que s'executen en un número relativament elevat de cicles i que ocupen molt espai en memòria. En el cas particular del simulador SIMR, totes les instruccions tenen la mateixa longitud, 16 bits. L'execució varia entre 4 cicles per instruccions AL i d'accés a memòria i 3 ó 5 les instruccions de salt.

## Exercici Guiat

Realitzeu amb l'ajut del professor el següent exercici:

Executeu el següent codi microinstrucció a microinstrucció seguint les instruccions del professor:

```
.begin start
start:
        ADD R0,R0,R3
        ADD R0,R0,R7
        ADDI R0, #4, R2
        ADD R2, R3, R3
        SUBI R7, #1,R7
        BG salto
salto:  STORE R3, guardaResultat(R0)
        .end
```

## Realització Pràctica

Executa el següent programa microinstrucció a microinstrucció:

```
valorDada: .dw 7
guardaResultat: .rw 1
        .begin start
start:  LOAD valorDada(R0), R1
```

```
        ADDI R0, #9, R2
        ADD R0, R0, R3
loop:   ADD R2, R3, R3
        SUBI R7, #1,R7
        BG loop
        STORE R3, guardaResultat(R0)
        .end
```

## Informe

Explica detalladament la pràctica realitzada. Fes els diagrames necessaris per entendre i mostrar el cicle d'execució dels diferents tipus d'instruccions.

Respon les següents qüestions:

i) Quan es produeix el salt, quants cicles triga en executar-se la instrucció BG loop?

ii) Quan NO es produeix el salt, quants cicles triga en executar-se la instrucció BG loop?

iii) Que bits del bus de control s'activen en el primer cicle de la instrucció ADD R2, R3, R3

iv) És igual la resposta del processador en el primer cicle de la instrucció ADD R2,R3,R3 que en el primer cicle de la instrucció LOAD valorDada(R0),R1?

v) Quan es produeix el salt (instrucció BG loop) , quina entrada del multiplexor s'activa com a sortida per apuntar a una determinada posició de la memòria? A quina posició apunta?

vi) Quan no es produeix el salt (instrucció BG loop) , quina entrada del multiplexor s'activa com a sortida per apuntar a una determinada posició de la memòria? A quina posició apunta?