PRÀCTIQUES DE L'ASSIGNATURA

INTRODUCCIÓ ALS ORDINADORS

UNIVERSITAT DE BARCELONA

Índice de contenido

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 opcional 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 Linux. 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.

INTRODUCCIÓ TEÒRICA. ESTRUCTURA D'UN ORDINADOR

DIAGRAMA DE BLOCS D'UN ORDINADOR

Un ordinador és un dispositiu electrònic basat en un processador, un sistema de memòria, i uns dispositius d'entrada sortida que permeten la interacció amb l'usuari. Aquesta màquina processa les dades a partir de un grup d'instruccions, el programa. És, en definitiva, una dualitat entre hardware (part física) i software (part lògica) que interactuen entre si per tal d'assolir una funcionalitat determinada.

L'ordinador segueix una arquitectura comú. És a dir, té uns determinats atributs que són visibles al programador i que li permeten assolir unes determinades tasques. En funció de l'arquitectura tindrem un conjunt d'instruccions, el nombre de bits per realitzar una determinada operació, les tècniques d'adreçar les dades a memòria, etc. La forma en com s'implementa això internament és el que es coneix com organització de l'ordinador.

- **Arquitectura:** consisteix en aquells atributs que són visibles al programador
  - Conjunt d'instruccions
  - Nombre de bits usat per representació de dades
  - Mecanismes Entrada/Sortida
  - Tècniques d'adreça
  - P.ex.:Hi ha instrucció de multiplicar?
- **Organització:** consisteix en com s'han implementat, típicament amagats al programador
  - Senyals de control, interfícies, tecnologia de memòria
  - P.Ex.: Hi ha unitat de multiplicació per HW o es fa per addició repetida (algoritme)?

L'estructura típica d'un processador és la proposada per Von Neumann. Els ordinadors que implementen aquesta arquitectura consten de cinc parts: La unitat aritmètic-lògica o ALU, la unitat de control, la memòria, els dispositius d'entrada-sortida i els busos de interconnexió, que proporcionen un medi de transport de les dades entre les diferents parts que constitueixen el computador. Un ordinador amb aquesta arquitectura realitza o emula els següents passos de manera seqüencial:

1.Encén l'ordinador i obté la següent instrucció des de la memòria en l'adreça indicada pel contador de programa i la guarda en el registre d'instruccions.

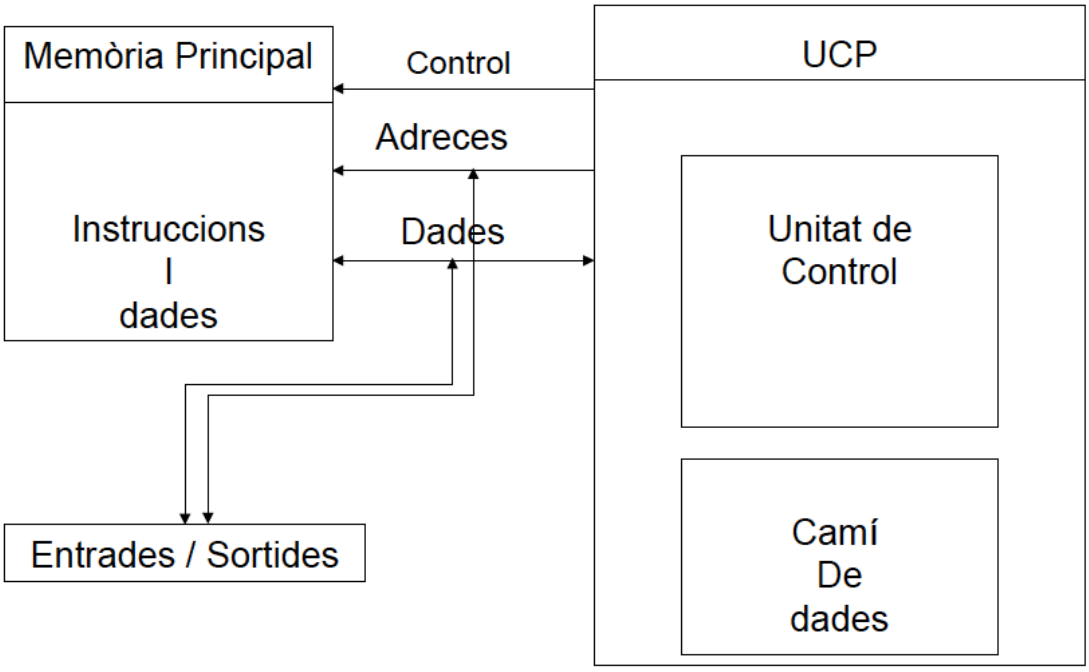
2.Augmenta el registre contador de programa, per apuntar a la següent instrucció.

3.Descodifica la instrucció mitjançant la unitat de control. Aquesta s'encarrega de coordinar la resta de components de l'ordinador per realitzar les operacions necessàries per tal d'executar la instrucció

4.S'executa la instrucció.

La figura 1 mostra aquest tipus d'arquitectura. A la memòria principal tenim el programa i la memòria assignada a les dades associades a l'execució del programa.

Figura 1. Estructura simplificada de l'arquitectura de Von Neumann



Tal i com s'observa a la figura 1, el bus d'adreces es unidireccional, ja que els generadors d'adreces apunten a la memòria principal i als dispositius d'entrada-sortida. A la CPU, no hi han adreces que adreçar per els altres dispositius. El bus de dades és bidireccional, per tal de permetre de llegir dades e instruccions per part de la CPU i escriure dades en la memòria o en els dispositius d'entrada-sortida. El bus de control ens serveix per identificar l'acció a executar. Per exemple, si el que volem és fer una escriptura a la memòria, des del bus de control seleccionarem la memòria i habilitarem el bit d'escriptura des del bus de control, a través del bus d'adreces indicarem la posició on volem guardar la dada i aquesta enviarà pel bus de dades.

Un exemple d'arquitectura von Neumann és la utilitzada per Intel en les seves famílies. Per exemple, a la figura 2 mostrem l'arquitectura del 8086.

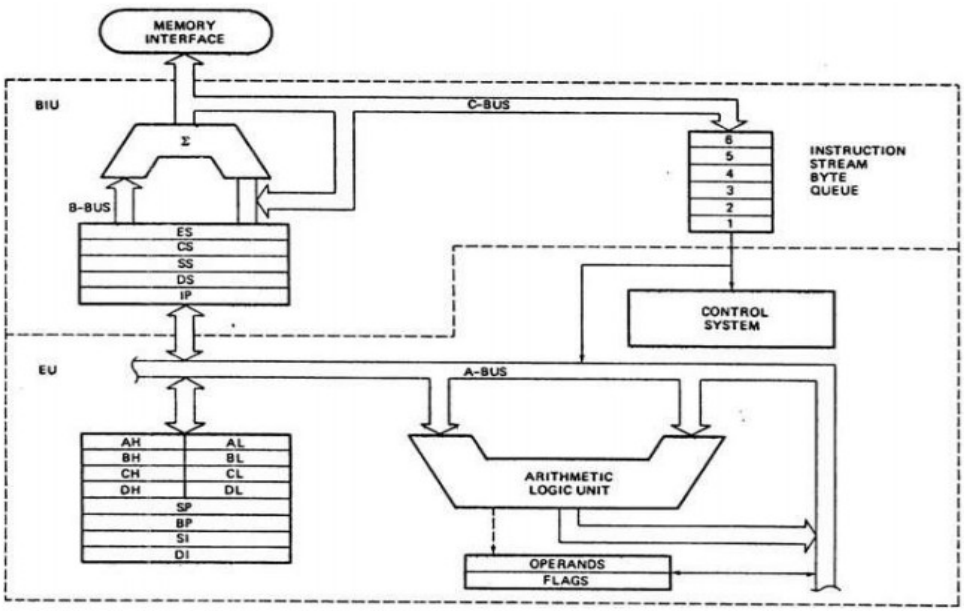


Figura 2. Arquitectura Intel 8086

Com s'ha comentat, aquesta arquitectura, tot i ser la més utilitzada, no és la única. Existeixen nombrosos processadors que implementen una arquitectura diferent, on el que tenim són dos memòries: una per dades i l'altra associada a la memòria de programa. Aquesta estructura es coneix

com arquitectura Harvard.

Originalment, el terme arquitectura Harvard feia referència a les arquitectures de computadors que utilitzaven dispositius emmagatzemament físicament separats per a les instruccions i les dades, en oposició a l'arquitectura de von Neumann. Aquest terme prové de la computadora Harvard Mark I, que emmagatzema les instruccions en cintes perforades i les dades en interruptors. La figura 3 mostra el diagrama de blocs simplificat de una arquitectura Harvard.

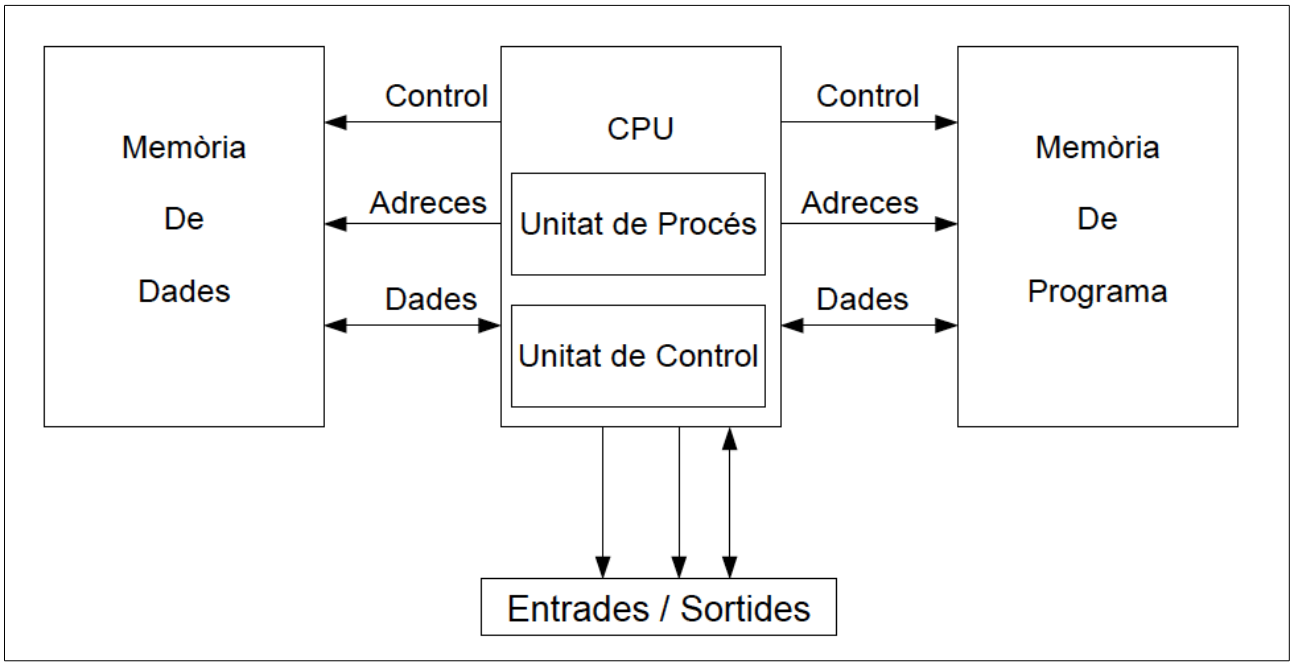


Figura 3. Diagrama de blocs simplificat de una arquitectura Harvard.

La figura 4 mostra un exemple de processador implementant aquest tipus d'arquitectura: El microprocessador de la casa Microchip de 8 bits 16F84A. La memòria de programa es una memòria Flash de múltiples lectures i escriptures, mentre que la memòria de dades és una memòria EEPROM. En aquest cas, les dues memòries s'integren en el mateix xip.

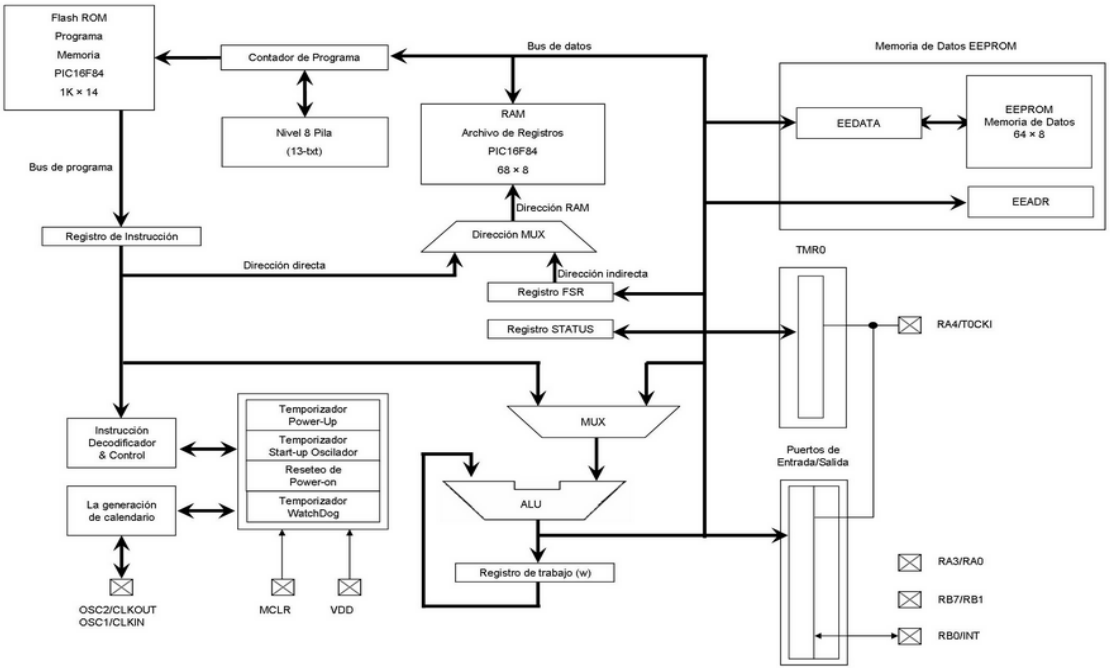


Figura 4. Diagrama de blocs arquitectura Harvard.

ESTRUCTURA BÀSICA DE LA CPU

Un processador consta dels següents blocs:

1. El camí de dades, que inclou, la Unitat Aritmètic Lògica (ALU), els registres de propòsit general, el registre acumulador, normalment connectat a la sortida de la ALU i les línies de interconnexió

2. La unitat de control, que s'encarrega de gestionar tot el funcionament del processador així com l'accés a la memòria principal

3. El conjunt de registres de propòsit específic, que es troben majoritàriament al camí de dades, però que poden estar associats a les altres unitats

4. Els busos d'interconnexió, entre els que tenim el bus de dades, el bus d'adreces i el bus de control

5. Els blocs de memòria cache, on tindrem les instruccions que s'han d'executar i les dades més importants.

El propòsit d'aquestes pràctiques inicials és justament entendre el funcionament de la CPU i la seva connexió amb la memòria principal. Per assolir aquest objectiu treballarem amb dos simuladors. El SiMR i el Ripes_RiscV. Tots dos tenen instruccions de mida constant. SiMR de 16 bits mentre que Ripes de 32 bits. En canvi, l'estructura de la CPU és diferent. SiMR segueix una estructura del tipus Von Neumann, amb una unitat de memòria on es guarden tant els programes con les dades. Ripes té una estructura Harvard, amb la memòria de dades per una banda i la memòria de programa per l'altre. Això ens permetrà analitzar i veure en la pràctica conceptes com el 'Pipeline' o execució pseudo-paral·lela de les instruccions. Per entendre el funcinament posarem l'exemple d'una bugaderia.

SiMR treballa de forma totalment seqüencial. La roba (el programa) està a la cistella inicial. Agafo la primera peça de roba i la fico a la rentadora, un cop acabada, fico la peça a l`assecadora, verifico que està neta i seca i la guardo un una segona cistella. Un cop acabat agafo la segona peça i la fico a la rentadora, després passo a l'assecadora, torno a verificar que estigui neta i seca i la guardo a la segona cistella... Aquest és el mode de treball sense pipeline.

Ripes treballa seguint una execució amb pipeline. Seguint amb l'exemple, primer agafa la primera peça de roba bruta i la fica en la rentadora, un cop acabada aquesta, fico la peça a l'assecadora, i com que la rentadora està buida, fico la segona peça de roba. Un cop acabat el procés de rentar i assecat, inspecciono visualment la peça rentada i secada, mentrestant, fico la segona peça ja rentada a l'assecadora i introdueixo una tercera peça a la rentadora... Aquest procés, implica fer servir més recursos del sistema en el mateix interval de temps, minimitzant el temps d'execució i millorant les prestacions de l'ordinador. Avui dia tots els ordinadors treballen d'aquesta manera.

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.

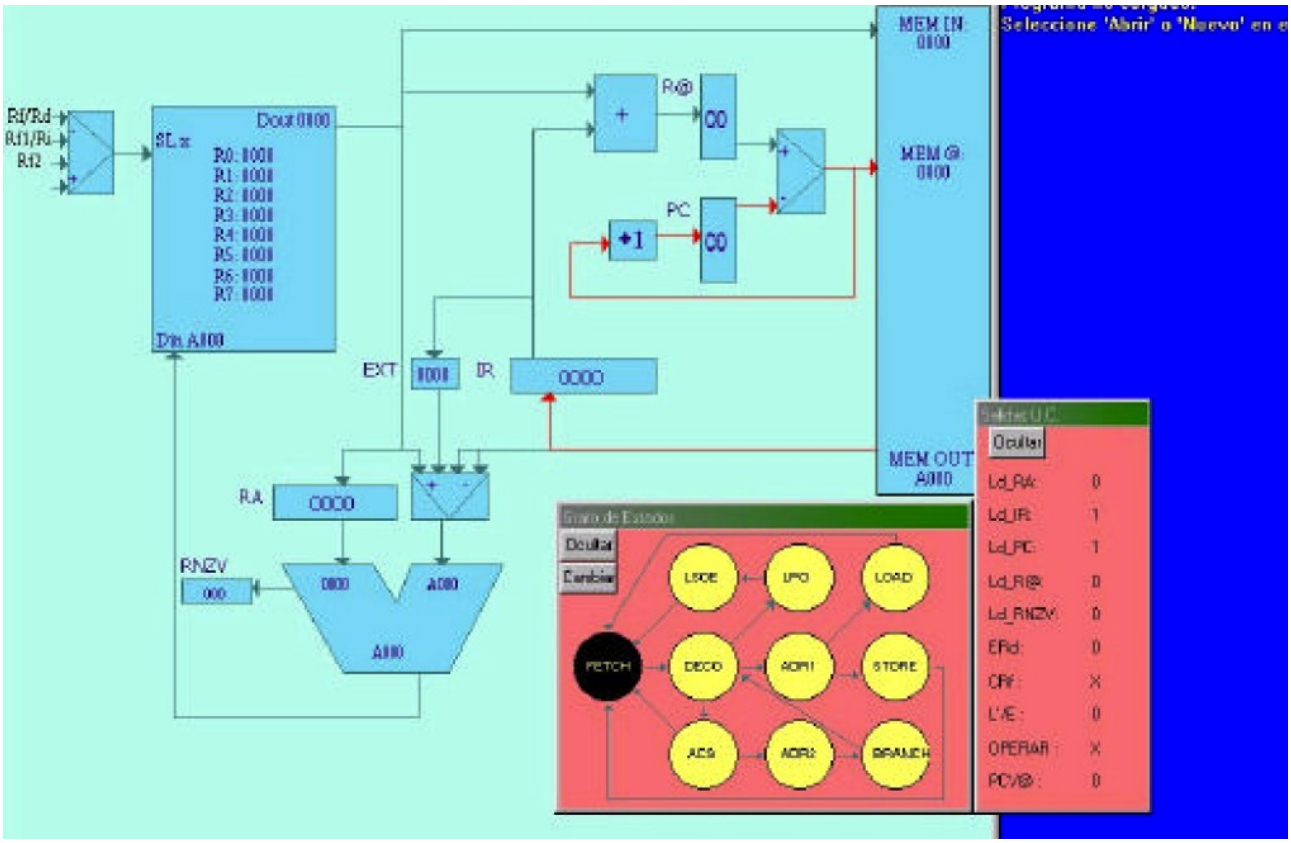


Figura 1. Diagrama del simulador SiMR

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		
b	STORE R1,R1(3)		
c	BG 6(R1)		
d	ADDI R2, #11, 5(R3)		
e	SUB R0,R2,R3		
f	LOAD 3(R0),R1		

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									
b	STORE R1,1(R3)									
c	BR 33									
d	ADDI R2,#11,R3									
e	SUB R0,R2,R3									
f	ASR R1,R1									

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. Establiu 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.

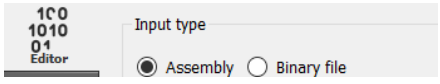
Treball a fer a casa

1.- Obre l'editor de text predefinit i escriu el següent programa:

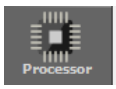
```
.data
#arguments i valors...
dataByte0: .byte 4
dataByte1: .byte 55
dataByte2: .byte 255
dataByte3: .byte 31
data1: .word 1
data2: .word 0
data3: .word 4

.text
main:
        lw a0, data1
        lb t0, dataByte0
        sw a0, data2(zero)
        lw a1, data3
loop:
        beq a1, a0, salta
        sub a1, a1, a0
        j loop
salta:
        lw a3, data2
end:
        j end
```

2.- Guarda el programa amb el nom Practica1_riscv.asm (sobre tot és important l'extensió). Obre el simulador Ripes. Posiciona a la part de l'editor.

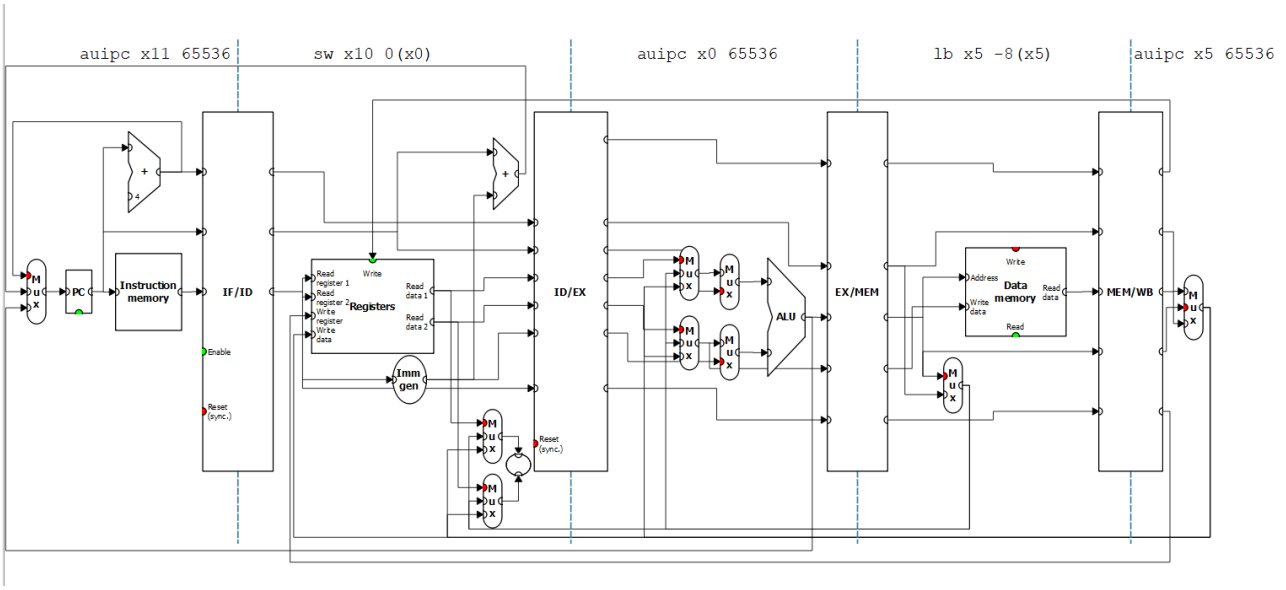


3.- Clica File i selecciona l'opció Load Assembly File. Veureu el vostre codi a la part esquerra de la pantalla (source code) i el codi desensamblat a la part dreta.

4.- Clica a la part del processador 

```
auipc x10 65536
lw x10 16(x10)
auipc x5 65536
lb x5 -8(x5)
auipc x0 65536
sw x10 0(x0)
auipc x11 65536
lw x11 0(x11)
beq x11 x10 12
sub x11 x11 x10
jal x0 32
auipc x13 65536
lw x13 -24(x13)
jal x0 52
```

5.- Executa el programa pas a pas. El programa que s'executarà és el desensamblat. El que tens a la figura superior. La primera instrucció el que fa es incrementar el registre PC per carregar la instrucció des de la memòria de programa. El processador amb el que esteu simulant és un processador amb 5 pipelines. Això vol dir que, en un cicle de rellotge està executant 5 fases de 5 instruccions diferents.



Analitza el comportament del programa.

Què fa? Reescriu el codi (sempre que es pugui) perquè s'executi en el SiMR.

Quines són les directives utilitzades en Ripes? Compara-les amb les de SiMR.

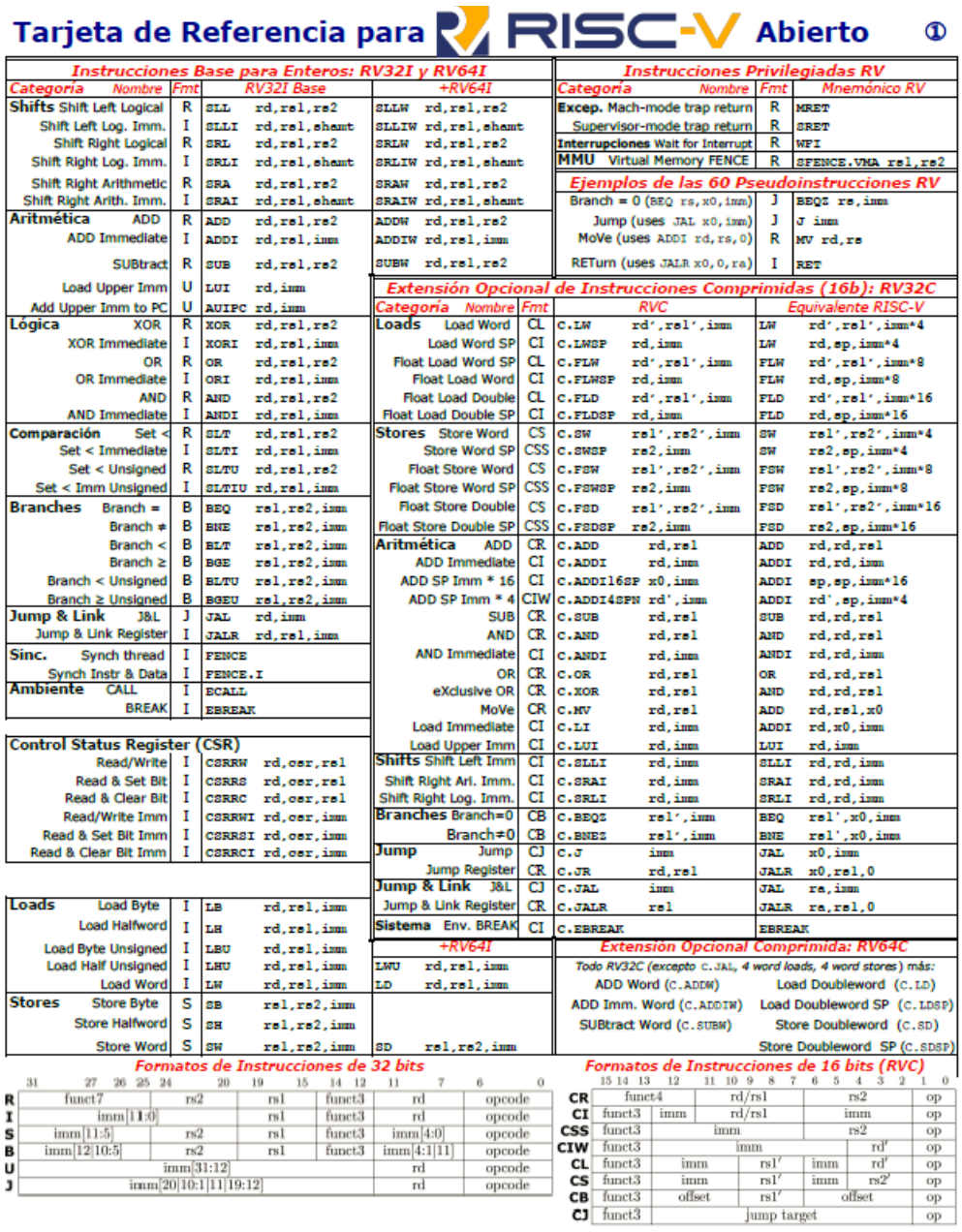
Ripes té 32 registres, entre els que tenim el primer x0, anomenat zero i que només és de lectura i té el valor 0. Tota la resta són accessibles tant per lectura com per escriptura.

Les instruccions lb i lw corresponen a fer un LOAD. En particular, lb és load byte i lw és load word. Un word correspon a 8 Bytes. La instrucció sw correspon a fer un STORE. En aquest cas tenim un store word.

Troba les similituds i diferències entre les instruccions emprades en SiMR i les instruccions emprades en Ripes. Feu els diferents programes implementats en SiMR amb Ripes.

La següent figura mostra el conjunt de registres i les instruccions .

Registro	Nombre ABI	Descripción
x0	zero	Alambrado a cero
x1	ra	Dirección de retorno
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Link register temporal/alterno
x6–7	t1–2	Temporales
x8	s0/fp	Saved register/frame pointer
x9	s1	Saved register
x10–11	a0–1	Argumentos de función/valores de retorno
x12–17	a2–7	Argumentos de función
x18–27	s2–11	Saved registers
x28–31	t3–6	Temporales
f0–7	ft0–7	Temporales, FP
f8–9	fs0–1	Saved registers, FP
f10–11	fa0–1	Argumentos/valores de retorno, FP
f12–17	fa2–7	Argumentos, FP
f18–27	fs2–11	Saved registers, FP
f28–31	ft8–11	Temporales, FP



Informe

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

Pràctica 2: Exercicis amb la Màquina SIMR i el simulador Ripes

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 els simuladors RIPES i 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 32 registres [X0 : X32] de propòsit general per el RISC-V i 8 registres [R0 : R7], el registre 0 sempre val 0, per tots dos casos, 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

Com seria aquesta instrucció amb el simulador Ripes?

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 singulars com resultats 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 RIPES.

```
.data
Dada1: .word 9
Dada2: .word 3
Dada3: .word 4
Dada4: .word 1
Reserva: .word 0
.text                                   #directiva d'inici de programa
main:
        SUB A0, A0, A0      #posem A0 = 0. A0 actuarà com a comtador
        LW A1, Dada1        #carreguem el contingut de @Dada1 en A1
        lw A2, Dada2
```

```
        lw A3, Dada3
        lw A4, Dada4
loop:
        add a5,a1,a2
        add a6,a4,a2
        addi a3, a3,-1
        bge a3, zero, loop
        sw a5, a0(Reserva)
        addi a0, a0,1
        sw a6, a0(Reserva)
end:    nop             #final de programa
```

Feu el següent exercici a casa utilitzant el simulador SIMR. Compareu-lo amb el fet a classe. Indiqueu diferències i similituds.

```
.begin inici                    ;directiva d'inici de programa
inci:
        SUB R7, R7, R7          ;posem R7 = 0. R7 actua com a comtador
        LOAD Dades(R7), R1      ;R1 <=4
        ADDI R7, #1, R7         ;R7 <=1
        LOAD Dades(R7), R2      ;R2 <=5
        ADDI R7, #1, R7         ;incrementem el contingut de R7, R7 <=2
        LOAD Dades(R7), R3      ;R3 <=6
        ADDI R7, #1, R7         ;incrementem de nou R7, R7<=3
        LOAD Dades(R7), R4              ;R4 <=7
loop:
        ;****************************************
        ; operacions entre registres bàsiques
        ;****************************************
        ADD R1, R2, R5
        ADD R3, R4, R6
        SUBI R3,#1, R3
        BG loop
.end                            ;directiva final de programa
Dades: .dw 4, 5, 6, 7           ;aquí tenim les dades per operar amb els registres
```

Quines operacions fan aquests codis?

Exercici 2 (Guiat)

Feu uns programes similars als de l'exercici 1. Inicialitzeu primer el contingut de A0. Carregueu al A1 el valor 0000110000001011b. Carregueu al registre A2 el valor 0000000000010001b. Feu l'operació A0<= A0+A1 i decrementeu el valor de A2. Feu això en un bucle fins que el valor contingut en A2 sigui 0.

Exercici 3 (a fer per l'alumne)

Ens demanen calcular un algoritme que ens faci el següent: Donades dues entrades emmagatzemades en les posicions de memòria 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 multipliqui el seu valor. Carregueu diferents valors en memòria per tal de comprovar el funcionament del programa. Quina instrucció feu servir per fer la multiplicació? Què és més eficient, una instrucció directament que faci aquesta operació o el càlcul iteratiu? Raoneu la resposta.

Informe

Expliqueu-ne la feina realitzada en aquesta pràctica.

A l'exercici 1:

- Un cop acabat el programa quant val el contingut de A5?
- Un cop acabat el programa quant val el contingut de A6?
- Un cop acabat el programa quant val el contingut de A3?

A l'exercici 2:

- Quant val el contingut de A0 quan acaba el programa?
- Quantes vegades s'executa el codi?
- Al final del programa, just abans del .end, poseu la instrucció `sw A5, A0(Reserva)`. En quina posición de memòria es guarda el resultat? En quina posició de memòria es guarda el contingut del registre A6?

A l'exercici 3:

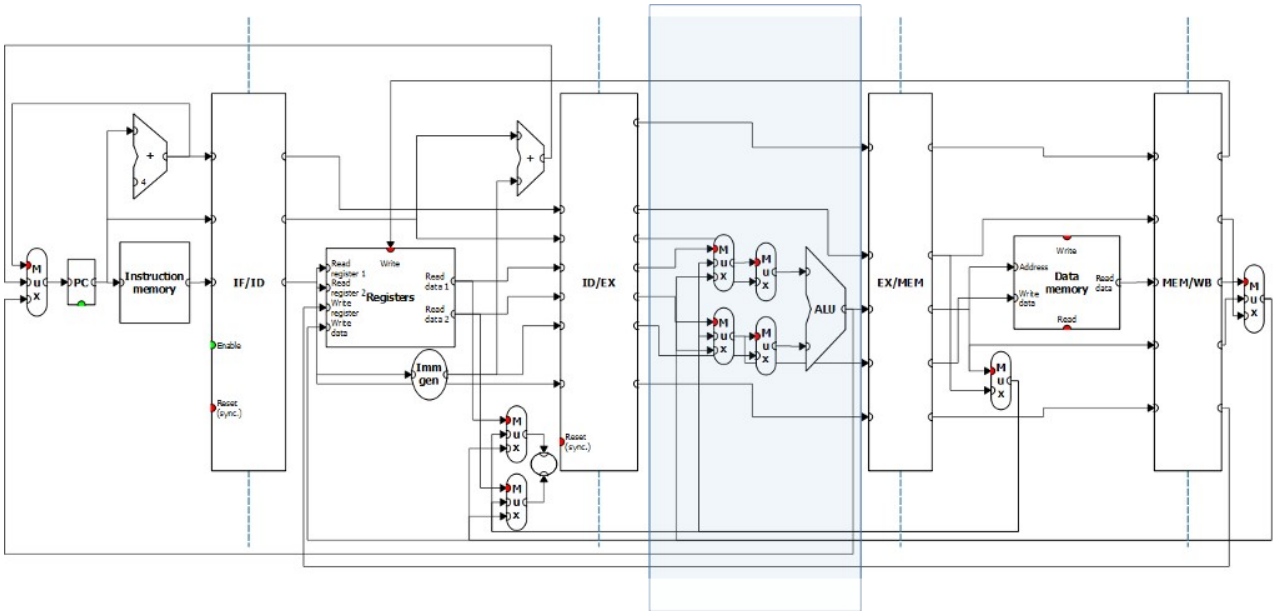
- Expliqueu el funcionament del programa
- Feu que el resultat es guardi a la posició de memòria 22h

Pràctica 3: Operacions i bucles amb RISC-V i SIMR

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.

Estructura de Ripes V



La zona marcada és el lloc on es realitza la comprovació per decidir si es produeix un salt o no. Les diferents opcions de salt venen donades per la taula presentada en el següent punt, i on el que es fa és comparar entre dos entrades, associades a dos registres.

Instruccions de salt

Instrucció	**Operació**	**Equivalent**
`beq rs, rt, destí`	Salta a "destí" si el registre rs = rt	
`bne rs, rt, destí`	Salta a "destí" si el registre rs ≠ rt	
`blt rs, rt, destí`	Salta a "destí" si el registre rs < rt	
`bge rs, rt, destí`	Salta a "destí" si el registre rs ≥ rt	
`bltu rs, rt, destí`	Salta a "destí" si el registre rs < rt (*)	
`bgeu rs, rt, destí`	Salta a "destí" si el registre rs ≥ rt (*)	
`beqz rs, destí`	Salta a "destí" si el registre rs = 0	`beq rs, zero, destí`
`bnez rs, destí`	Salta a "destí" si el registre rs ≠ 0	`bne rs, zero, destí`
`blez rs, destí`	Salta a "destí" si el registre rs ≤ 0	`bge zero, rt, destí`
`bgez rs, destí`	Salta a "destí" si el registre rs ≥ 0	`bge rs, zero, destí`
`bltz rs, destí`	Salta a "destí" si el registre rs < 0	`blt rs, zero, destí`
`bgtz rs, destí`	Salta a "destí" si el registre rs > 0	`blt zero, rs, destí`
`bgt rs, rt, destí`	Salta a "destí" si el registre rs > rt	`blt rt, rs, destí`
`ble rs, rt, destí`	Salta a "destí" si el registre rs ≤ rt	`bge rt, rs, destí`
`bgtu rs, rt, destí`	Salta a "destí" si el registre rs > rt (*)	`bltu rt, rs, destí`
`bleu rs, rt, destí`	Salta a "destí" si el registre rs ≤ rt (*)	`bgeu rt, rs, destí`
`j destí`	Salta a "destí" incondicionalment	

* Comparacions sense complement a dos, considerant només números positius.

Problema 1
Els salts ens permeten executar diferents blocs de codi i així podem implementar estructures de control de flux.

Exemple **if**: distància entre dos números enters

Per calcular la distància entre dos números enters, farem la resta i llavors el valor absolut.

$$|a - b|$$

Exemples:

$$|5 - (-6)| = 11$$

o

$$|-6 - 5| = 11$$

El programa següent en C calcula aquesta distància. Com es faria en RISC V?

Alt nivell (C)	RISC V

`int a = 5;`
`int b = -6;`
`int resultat;`
 	
`if (a >= b)`
`resultat = a - b;`
`else`
`resultat = b - a;`
	`.data`
	`a: .word 5`
	`b: .word -6`
	`resultat: .word 0`
	`.text`
	`la a0, a`
	`lw a1, 0(a0)`
	`lw a2, 4(a0)`
	`# condició`
	`cert:`
	`# branca certa`
	`fals:`
	`# branca falsa`
	`end:`
	`sw a3, 8(a0)`

Nota important: Fixa't que la condició per al codi d'alt nivell ($a \geq b$) és la contrària que fem servir en el codi en llenguatge assemblador.

Això passa perquè en el codi d'alt nivell la condició indica que s'executa la branca certa ($a \geq b$). En canvi, en el codi màquina de l'exemple anterior indica que saltem a la branca falsa ($a < b$).

També és important recordar que després d'executar la branca certa cal saltar-se el bloc de la branca falsa.

Problema 2
Exemple **while**: Fibonacci

La successió de Fibonacci és una successió matemàtica de nombres naturals tal que cada un dels seus termes és igual a la suma dels dos anteriors.

$F_n = F_{n-1} + F_{n-2}$

sempre es parteix dels valors inicials

$F_0 = 0, F_1 = 1$

i es a partir del elements F_0 i F_1 que es genera la resta d'elements. Exemples:

$F_2 = 1$

$F_3 = 2$

$F_4 = 3$

…

$F_9 = 34$

$F_{10} = 55$

El programa següent calcula el terme de la série de Fibonacci indicat per la variable "comptador" ($F_{comptador}$), i el desa a la variable "resultat". **Com es faria en RISC V?**

Alt nivell (C)	RISC V

`int comptador = 10;`
`int resultat;`
 	
`int a = 0;`
`int b = 1;`
`while (comptador > 0) {`
`  int t = a + b;`
`  a = b;`
`  b = t;`
`  comptador--;`
`}`
`resultat = a;`
	`.data`
	`comptador: .word 10`
	`resultat: .word 0`
	`.text`
	`  la a0, comptador`
	`  lw a0, 0(a0)`
	`  addi a1, zero, 0`
	`  addi a2, zero, 1`
	`loop:`
	`  # condició`
	`  # cos del bucle`
	`end:`
	`  la a0, resultat`
	`  sw a1, 0(a0)`

Nota important: De nou, la condició per al codi d'alt nivell (comptador > 0) és la contrària que fem servir en el codi en llenguatge assemblador.

En aquest cas, el codi d'alt nivell avalua si es manté en el bucle (comptador > 0), mentre que el codi en llenguatge assemblador salta si surt del bucle (comptador = 0).

A més, al final del bucle hem de tornar al principi.

Problema 3
Per fer a casa: implementa les parts que falten del programa assemblador.

Exemple **while amb if**: màxim comú divisor

El màxim comú divisor (mcd) de dos o més nombres enters positius és el major divisor possible de tots ells.

mcd(a,b)

Exemples:

mcd(252,105) = 21

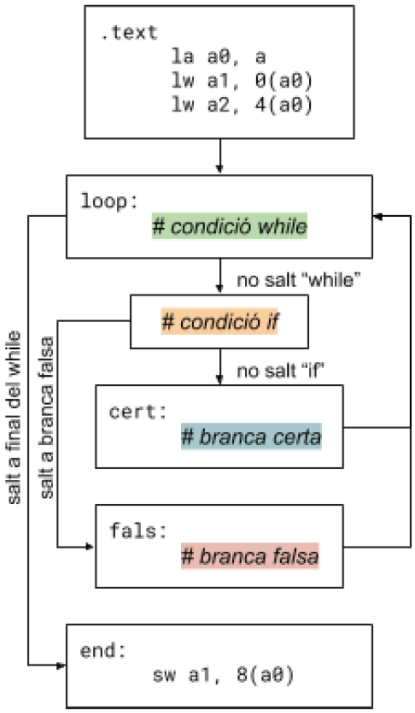
mcd(13,31) = 1

A continuació hi ha la implementació en C usant l'algorisme d'Euclides. Dins d'un bucle "while", a cada iteració es fa una comparació en un "if". **Com es faria en RISC V?**

Tingues en compte el flux del programa assemblador, i el que hem explicat en els exercicis anteriors sobre com avaluem de forma diferent les condicions en llenguatge d'alt nivell i en llenguatge màquina, i que algunes estructures de flux necessiten també salts incondicionals.

Alt nivell (C)	RISC V

`int a = 252;`
`int b = 105;`
`int resultat;`
 	
`while (a != b) {`
`  if (a > b)`
`  a = a - b;`
`  else`
`  b = b - a;`
`}`
`resultat = a;`
	`.data`
	`a: .word 252`
	`b: .word 105`
	`resultat: .word 0`
	`.text`
	`  la a0, a`
	`  lw a1, 0(a0)`
	`  lw a2, 4(a0)`
	`loop:`
	`  # condició while`
	`  # condició if`
	`cert:`
	`  # branca certa`
	`fals:`
	`  # branca falsa`
	`end:`
	`  sw a1, 8(a0)`



Informe
Exercici 1

1. Quines instruccions de salt condicional hem fet servir? En quin cas salten?
2. Què fan les instruccions de salt condicional quan la condició no es compleix?
3. Per què hem utilitzat les instruccions de salt incondicional?

Exercici 2

1. Quin tipus d'estructura de control de flux indica normalment un salt cap enrere?
2. Omple la següent taula executant el programa.

Valor inicial	Valor a "resultat"	# cicles	# instruccions
F_0			
F_1			
F_2			
F_3			
F_4			
F_5			
F_6			
F_{25}			
F_{46}			
F_{47}			

3. Què passa amb el resultat F_{47}?

Exercici 3

Descriu el programa que has implementat. Quins salts has usat? Quins són condicionals i quins són incondicionals, i per què?

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?