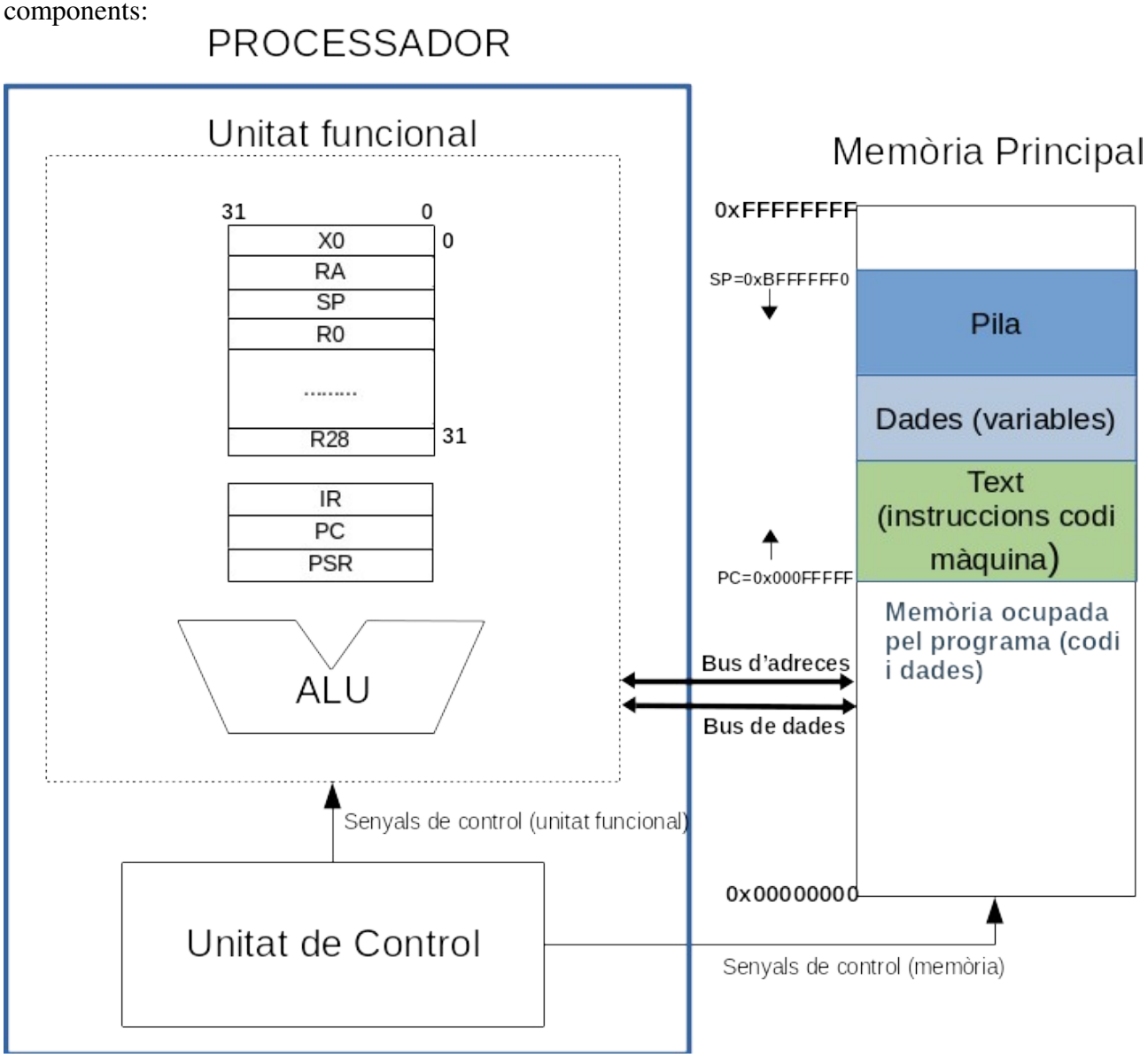
Per a la realització dels problemes d'aquest tema, disposem d'un sistema que conté els següents components:



La arquitectura del processador disposa del següents principals recursos :

- **32 registres de propòsit general** (poden ser utilitzats per la majoria d'instruccions assemblador de la màquina), Els tres primers, a més a més, tenen una funcionalitat dedicada: **X0** es un registre que sempre té el valor de **0**, **RA** es el registre a on es guarda la adreça de retorn d'una funció i **SP** (Stack Pointer) es el apuntador de la pila.
- Un registre de estat (**PSR**, Program Status Register) que conté informació del estat de la màquina.
- Un registre comptador de programa (**PC** Program Counter) que conté l'adreça de memòria de la següent instrucció a executar.
- Una **ALU** que realitza les operacions aritmètiques (**+, -, *, /, mòdul**), lògiques (**AND,OR, XOR**) i de desplaçament de bits (**Shift**).
- Un registre d'instrucció **(IR)** que Emmagatzema la instrucció que s'està executant actualment.

Les dades son del tipus **sencer (4 bytes)** i poden ser **amb i sense signe**. Els nombres amb signe es representen en **C2**. Cada cop que es demana o s'escriu una dada sencera a la memòria, s'escriuen o llegeixen els seus 4 bytes a partir de la adreça de memòria a on comença aquesta dada. Per exemple, si una dada comença a l'adreça 1000h de memòria, aquesta ocuparà les adreces consecutives compreses entre la 1000h i la 1003h.
A continuació es mostren les instruccions màquina d'aquesta arquitectura, agrupades per funcionalitat.

Nomenclatura utilitzada:

- **imm:** Valor immediat (**sencer amb signe**)
- **símbol:** Nom d'una variable
- **[mem (adreça_mem)]:** Accés al contingut d'una direcció de memòria (per escriure o llegir una dada) indicada per **adreça_mem.** Per exemple, si **adreça_mem** val 1000h, llavors **[mem (1000)]** vol dir accedir al contingut de l'adreça de memòria 1000h.
- La '<-' indica el sentit del moviment de les dades. Per exemple **R0 <- R1**, significa copiar el contingut del registre R1 al R0.

a) Lectura/escriptura memòria :

- **LLA *Rd, símbol:*** Emmagatzema en el registre **Rd** la adreça de memòria de la variable amb el nom passat com el seu segon operand ***símbol.***
- **LW *Rd, desplaçament(Rm):*** Llegir una dada (tipus sencer) de la memòria.
- ***SW Rd, desplaçament(Rm)*** Escriure una dada (tipus sencer) a memòria.
  Tipus de adreçament d'aquestes instruccions ***:***
  - **Indexat sobre registre:** Format operant: ***desplaçament(Rm).***
    Operació per **LW: Rd <- [mem (Rm+desplaçament)].**
    Operació per **SW: [mem (Rm+desplaçament)] <- Rd.**
    Si ***desplaçament*** es igual a **0**, llavors tenim un adreçament **directe a registre ([mem (Rm+0)] es [mem (Rm)])** .

b) Lectura d'un valor immediat

- **LI *Rd, imm*:** Guarda el valor immediat ***imm*** en el registre ***Rd*** (**Rd <- imm)**

c) Moviment:

- **MV *Rd, Rm* :** Copia el contingut del registre ***Rm*** en el registre ***Rd*** (**Rd <- Rm)**

d) Aritmètiques, lògiques i de desplaçament:
La seva codificació pot ser ***CODI_OP Rd, Rm, Rs*** *o* ***CODI_OPI Rd, Rm, imm (I de valor immediat)*** i les operacions que realitzen respectivament son: ***Rd <- Rm oper Rs o Rd <- Rm oper imm,*** a on ***oper*** es una de les operacions que realitza la ALU. Els codis d'operació (***CODI_OP o CODI_OPI)*** de cada instrucció son:

- **ADD/ADDI:** Suma
- **SUB:** Resta
- **MUL** : Multiplicació (extensió arquitectura)
- **DIV:** Divisió (extensió arquitectura)
- **SRA/SRAI :** Desplaçament aritmètic a la dreta
- **SLL/SLLI :** Desplaçament lògic a l'esquerra
- **SRL/SRLI:** Desplaçament lògic a la dreta
- **AND/ANDI:** And bit a bit

- **OR/ORI:** Or bit a bit
- **XOR/XORI:** Xor bit a bit

e) Salts condicionals:

Forma general **B*cond* RX, RY *etiqueta*** a on el sufix ***cond*** indica la condició aplicada als dos registres passats com operands: ***RX cond RY*** (per exemple si ***cond*** es **EQ,** llavors es compara la igualtat dels dos registres passats com operands, ***RX == RY?***). Si es compleix la condició indicada pel sufix ***cond***, el registre PC s'actualitzarà amb l'adreça de memòria indicada per ***PC+etiqueta*** (l'execució del programa saltarà a l'adreça ***PC+etiqueta***). En cas contrari, no es compleix la condició, el registre PC no es modificarà i es continuarà la execució del programa en seqüencia. Cal recordar que en l'execució estàndard d'un programa, el registre PC conté l'adreça de la següent instrucció.

EQ: Igual
NE: No Igual
LE: Més petit o igual (amb signe)
LEU: Més petit o igual (sense signe)
LT: Més petit (amb signe)
LTU: Més petit (sense signe)
GE: Més gran o igual (amb signe)
GEU: Més gran o igual (sense signe)
GT: Més gran (amb signe)
GTU: Més gran (sense signe)

f) Salt incondicional:

- **J etiqueta:** El registre PC s'actualitzarà amb l'adreça de memòria indicada per ***PC+etiqueta*** (l'execució del programa saltarà a l'adreça ***PC+etiqueta***)

g) Crida i retorn de subrutina:

- **JAL Rd, etiqueta_subrutina:** Guarda al l'adreça de la següent instrucció (**PC + 4**) al registre Rd (la del retorn de la subrutina) i actualitza el registre PC amb l'adreça de memòria, **etiqueta_subrutina,** passada com a segon operand **(PC <- etiqueta_subrutina,** aquesta adreça indica la 1ª instrucció de la subrutina) .
- **JALR Rd, desplaçament(Rm)**: Igual que l'anterior instrucció, però actualitza el registre PC amb l'adreça de memòria obtinguda de fer la següent operació (amb els components del segon operand): **Rm** + **desplaçament (PC <- Rm** + **desplaçament).**
- **RET (Retorn de subrutina)** Situa al registre PC l'adreça de memòria de la instrucció de retorn de la subrutina situada al en un registre especial (per continuar l'execució del programa després de la crida a subrutina). El registre que conté l'adreça de retorn es decidit amb les instruccions **JAL o JALR** .

f) Pila:

- Situar el contingut del registre **Rd** al cap de pila (**push Rd**). Operacions:
  1) SW Rd, 0(SP) **; [mem (SP)] <- Rd**
  2) ADDI SP, SP, -4 **; SP <- SP - 4**
- Treure el contingut del cap de la pila i situar-lo en el registre **Rd (pop Rd).** Operacions:
  1) ADDI SP,SP,4 **;SP <- SP + 4**
  2) LW Rd, 0(SP) **;Rd <- [mem(SP)].**

Exemple:

Si a l'adreça de memòria 1000h (hexadecimal) tenim una variable sencera anomenada **var_a** amb el valor 0x0000000A, llavors:

salt:

```
LLA R2, var_a     ; Guarda l'adreça de var_a en R2, R2 conté 1000h
LW  R3, 0(R2)     ; Guarda el contingut de l'adreça emmagatzemada en el  registre R2 (que
                  ; conte  1000h) en R3, per tant R3 conté 0x0000000A.
ADD R3, R3, R2    ; R3 conté el resultat de R3 + R2, R3 <- 0x0000000A + 0x1000,
                  ; R3 <- 0x0000100A
LI R4,0           ; Posa el valor immediat 0 a R4
BLE R3,R4, salt   : Salta a la instrucció de la etiqueta salt si R3 <= 0  (amb signe)
SW R3,0(R2)       ; Emmagatzema el contingut del registre R3 en l'adreça de memòria
                  ; continguda  en el registre R2, per tant, l'adreça 1000h ara contindrà
                  ; 0x0000100A
```

Problema 1.

L'objectiu d'aquest problema consisteix en traduir fragments de codi en C a llenguatge assemblador.

En els codis escrits en C no declarem les variables que es fan servir. Suposem que aquestes variables ja les tenim declarades i en tots els casos són nombres **enters sense signe de 32 bits** codificats en complement a dos. Tampoc heu de reservar espai en memòria ni inicialitzar les variables en la memòria de l'assemblador. Per referir-nos a l'adreça de memòria d'una variable que surt en el codi en C utilitzem el mateix nom de la variable en C. Per exemple, per traduir la següent sentència en C,

```
A = A + B;
```

podem fer servir el codi següent:

```
LLA R1, A
LW R2, 0(R1)
LLA R3, B
LW R3, 0(R3)
ADD R3, R2, R3
SW R3, 0(R1)
```

Traduïu a llenguatge assemblador cadascun dels següents fragments de codi.

a) A=B+C*D;

b) while (A<B)

```
{
    C = C + C;
    A = A + 1;
}
```

Problema 2.

Escriure un fragment de codi en llenguatge assemblador que realitzi la funcionalitat que es descriu en cadascun dels següents apartats:

a) Emmagatzemi en R1 un 1 si el contingut de R2 és més gran o igual que el de R3 i a més el de R4 és més petit que el de R5. Si no passa tot l'anterior, emmagatzemi un 0. Operem amb **nombres amb signe.**

b) Emmagatzemi en R1 un 1 si es compleix una i només una de les dues condicions següents, Operem amb **nombres sense signe** :

- El contingut de R2 es més gran o igual que el de R3
- El contingut de R4 es més petit que el de R5.

En qualsevol altre cas emmagatzemeu en R1 un 0.

Problema 3.

A partir del següent programa escrit en el llenguatge d'alt nivell C, que calcula la suma dels elements d'un vector, escriure un programa en assemblador que implementi la mateixa funcionalitat. Considereu que no ens interessa el valor final de les variables **i** i **partial_sum**, . Per això, per optimitzar el codi heu de fer servir registres del processador per emmagatzemar els seus valors.
Suposeu que al programa escrit en C s'ha definit la variable V com a vector d'enters de 32 bits i la variable sum com a nombre enter de 32 bits i que es troben emmagatzemats a partir de les adreces simbòliques de memòria V i sum, respectivament.

```
int N = 6;
int sum = 0;
int partial_sum = 0;
int i = 0;
while ( i < N) {
        partial_sum = partial_sum + V[i];
        i = i + 1;
}
sum = partial_sum;
```

Problema 4.

Donats els següents fragments de codi situats en la memòria principal, mostrar com va variant el contingut de les posicions de memòria dedicades a les dades i la pila, així com dels registres del processador: **PC, Ri (i=0,..28), SP, RA i IR** després de l'execució de cadascuna de les instruccions d'aquests fragments de codi. La columna. DIR ha d'indicar l'adreça de memòria efectiva utilitzada pel mode d'adreçament de cada instrucció. La primera fila mostra l'estat inicial de la memòria i els registres. El **registre PC** ha de contenir la següent instrucció a executar. Totes les adreces i dades estan en decimal.

Universitat Autònoma de Barcelona

CODIS EN MEMÒRIA PRINCIPAL

ADRECES (de 4 en 4)	INSTRUCCIONS
12,..,15	LI R1, 60
16,..,19	SW R2,0(SP)
20,..,23	ADDI SP,SP.-4
24,..,27	JAL RA, 40
28,..,31	ADDI SP,SP.4
32,..,35	LW R2,0(SP)
.....	
40,..,43	LW R3, 4(R3)
44,..,47	SUB R2, R1,R3
48,..,51	SW R2, 8(R1)
52,..,55	RET

Adreces de memòria (de 4 en 4) dades i pila						Registres processador							
60,..,63	**64,..,67**	**68,..,71**	**72,..,75**	**76,..,79**	**80,..,83**	**PC**	**IR**	**R1**	**R2**	**R3**	**SP**	**RA**	**DIR**
4	8	4	0	0	8			8	4	64	80		- - -

Problema 5.

a) Repetir les mateixes accions que amb el exercici anterior, però amb els següents fragments de codi

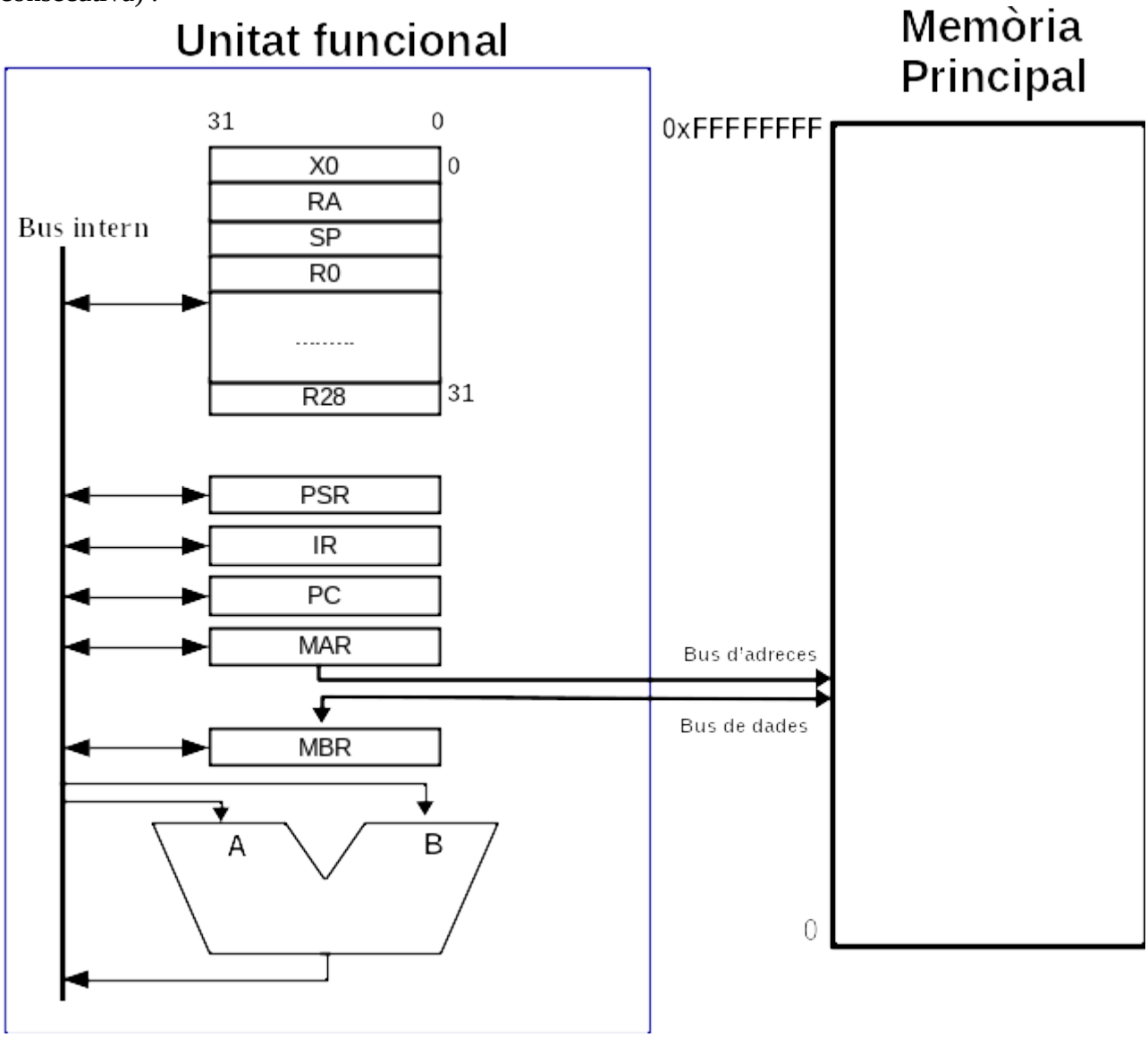
CODIS EN MEMÒRIA PRINCIPAL

ADRECES (de 4 en 4)	INSTRUCCIONS
12,..,15	LI R1, 76
16,..,19	JAL RA, 32
20,..,23	SW R3, 4(R1)
.....	
32,..,35	SW RA,0(SP)
36,..,39	ADD R2, R2,R1
40,..,43	JAL RA 52
44,..,47	LW RA,0(SP)
48,..,51	RET
52,..,55	SW R1,-4(SP)
56,..,59	MUL R2, R3,R2
60,..,63	LW R1,-4(SP)
64,..,67	RET

Adreces de memòria (de 4 en 4) dades i pila						Registres processador							
72,..,75	**76,..,79**	**80,..,83**	**84,..,87**	**88,..,91**	**92,..,95**	**PC**	**IR**	**R1**	**R2**	**R3**	**SP**	**RA**	**DIR**
4	8	4	0	0	8			8	4	2	88		- - -

Problema 6.

A continuació es mostra un esquema ampliat de la **unitat funcional** del processador del sistema (a on també es mostren el bus intern i els registres MAR i MBR per operar amb la memòria)
Suposar que els **busos de dades i de direccions del sistema són de 32 bits cadascun** i recordeu que **cada accés a memòria llegeix o escriu dades de 4 bytes** (s'accedeix a 4 posicions de memòria consecutiva) .



Les principals fases o etapes implicades en l'execució d'una instrucció són:

1) Cerca de la instrucció (Fetch): Portar la instrucció situada en la **memòria,** al registre d'instruccions (**IR**)

2) Descodificació de la instrucció (Decode): Esbrinar quina instrucció es vol executar.

3) Execució de la instrucció (Execute): Executar l'operació indicada per la instrucció.

Amb les instruccions dels bucles dels diferents codis de l'apartat b) de l'exercici 1, contesteu:

a) Mostreu les **operacions elementals** de les següents instruccions :

a.1) LW R5, 0(R4)

a.2) ADD R2, R2,1

Problema 7.

Considerant l'esquema de l'exercici anterior i amb les instruccions dels bucles dels diferents codis de l'apartat b) de l'exercici 1: Quants accessos a memòria produiran les execucions d'aquests 2 bucles?. Si el temps d'execució d'un accés a memòria implica 10 cicles del processador, Quin serà el temps que implicaran el total d'aquests accessos a memòria?. Considereu que A=0 i b=10