Architecture des ordinateurs 1

Jean-Luc Scharbarg - ENSEEIHT - Dpt. SdN

Janvier 2021

Organisation de l'enseignement

- Volume horaire
  - Cours : 3 $\times$ 1 heure 45
  - TD : 3 $\times$ 1 heure 45
  - TP : 6 $\times$ 1 heure 45
  - Contrôle : 1 $\times$ 1 heure
- Objectif
  - Introduction à l'exécution d'un programme écrit dans un langage de haut niveau sur une architecture matérielle
  - Introduction à l'échange d'informations entre un processeur et son environnement

D'un algorithme à son exécution

- Algorithme à exécuter

Calcul du PGCD de x et y

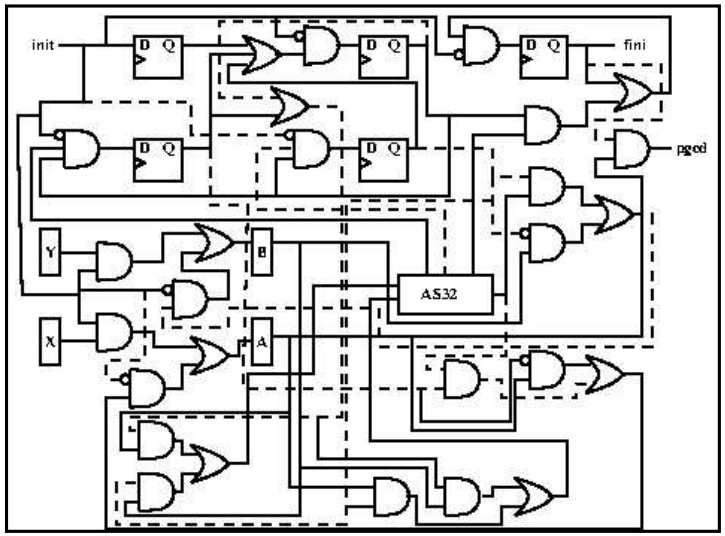
```
tq x != y faire
  si x > y alors
    x <– x–y;
  sinon
    y <– y–x;
  fsi;
ftq;
pgcd <– x;
```

x	y
35	21
14	
	7
7	

- solutions possibles
  - Un circuit complètement dédié
  - Un circuit configurable
  - Un circuit programmable (processeur)

De l'algorithme au circuit complètement dédié

```
module pgcd(clk,rst,init,x[31..0],y[31..0]:fini,pgcd[31..0])
  debut := init on clk reset when rst
  cmpab := /init*(debut+ab+ba) on clk reset when rst
  ab := cmpab*asupb on clk reset when rst
  ba := cmpab*bsupa on clk reset when rst
  trouve := /init*(cmpab*aeqb+trouve) on clk reset when rst
  a[31..0] := init*x[31..0]+/init*(ab*s[31..0]+/ab*a[31..0]) on clk reset when rst
  b[31..0] := init*y[31..0]+/init*(ba*s[31..0]+/ba*b[31..0]) on clk reset when rst
  e1[31..0] = a[31..0]*(cmpab+ab)+b[31..0]*ba
  e2[31..0] = a[31..0]*ba+b[31..0]*(cmpab+ab)
  AS32(e1[31..0],e2[31..0],1,s[31..0],asupb,bsupa,aeqb)
  fini = trouve
  pgcd[31..0] = trouve*a[31..0]
end module
```



Que comprend ce circuit ?

- Un ensemble de bascules pour mémoriser l'état courant

  debut := init on clk reset when rst
  cmpab := /init*(debut+ab+ba) on clk reset when rst
  ab := cmpab*asupb on clk reset when rst
  ba := cmpab*bsupa on clk reset when rst
  trouve := /init*(cmpab*aeqb+trouve) on clk reset when rst

- Des registres pour mémoriser les valeurs courantes des variables de calcul

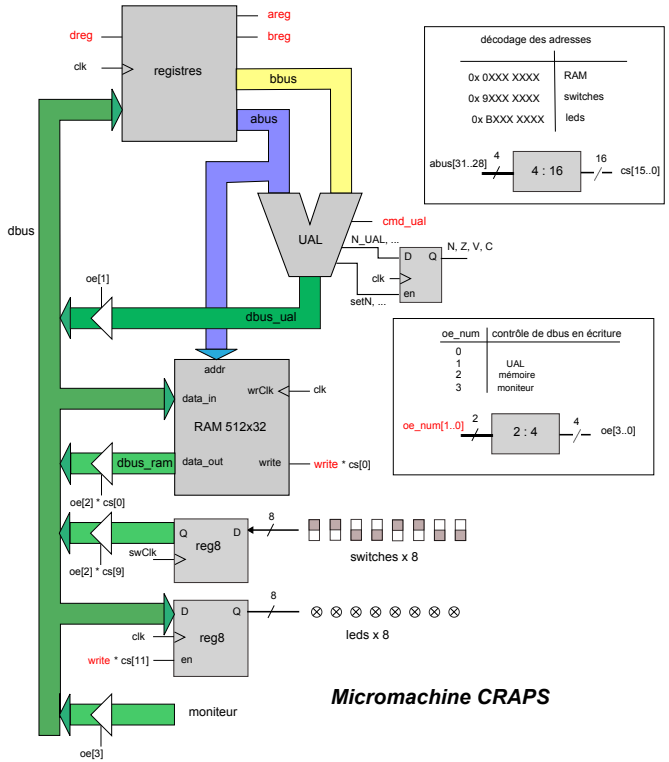
  a[31..0] := init*x[31..0]+/init*(ab*s[31..0]+/ab*a[31..0]) on clk ...
  b[31..0] := init*y[31..0]+/init*(ba*s[31..0]+/ba*b[31..0]) on clk ...

- De la logique combinatoire pour effectuer les calculs

  e1[31..0] = a[31..0]*(cmpab+ab)+b[31..0]*ba
  e2[31..0] = a[31..0]*ba+b[31..0]*(cmpab+ab)
  AS32(e1[31..0],e2[31..0],1,s[31..0],asupb,bsupa,aeqb)

De l'algorithme au processeur

```
1  /* TD1 calcul du PGCD */
2
3          set    78, %r1 // valeur de x
4          set    143, %r2 // valeur de y
5
6  pgcd:   cmp    %r2, %r1 // tant que x <> y
7          bne    skip // x=y ?
8          // x = y : return x
9          mov    %r1, %r3 // val <- x
10         ba     stop
11 skip:   bneg   sup // x>y ?
12         // x < y
13         subcc  %r2, %r1, %r2 // y <- y - x
14         ba     pgcd
15 sup:    // x > y
16         subcc  %r1, %r2, %r1 // x <- x - y
17         ba     pgcd
18 stop:   ba     stop   // arrêt
```



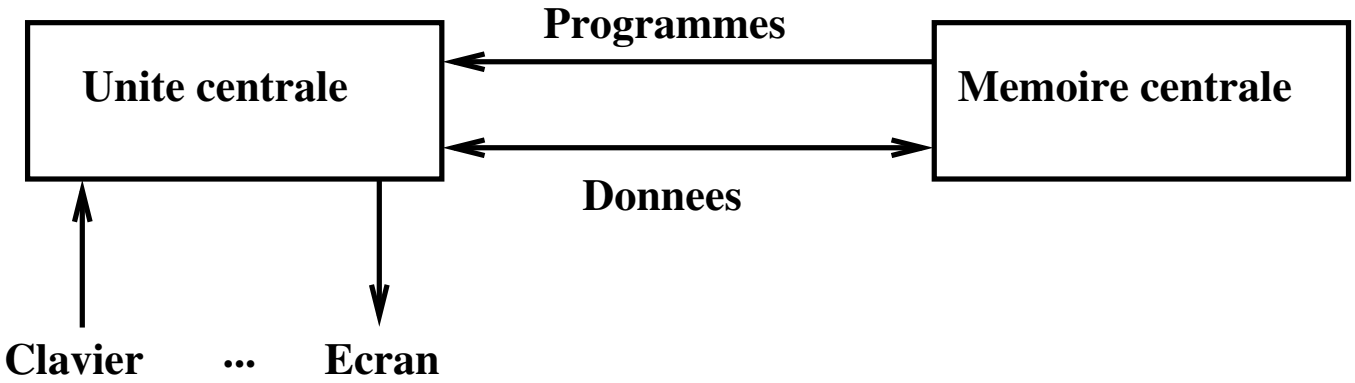
Que comprend un processeur ?

- Un ensemble de bascules pour mémoriser l'état courant du processeur
  - Où en est-il dans l'exécution de l'instruction courante ?
  - ...
- Un ensemble de registres pour mémoriser
  - les valeurs des variables du programme
  - la position dans l'exécution du programme
  - ...
- Un ensemble de circuits de calcul
  - Un additionneur/soustracteur
  - ...

Un ordinateur, c'est quoi ?

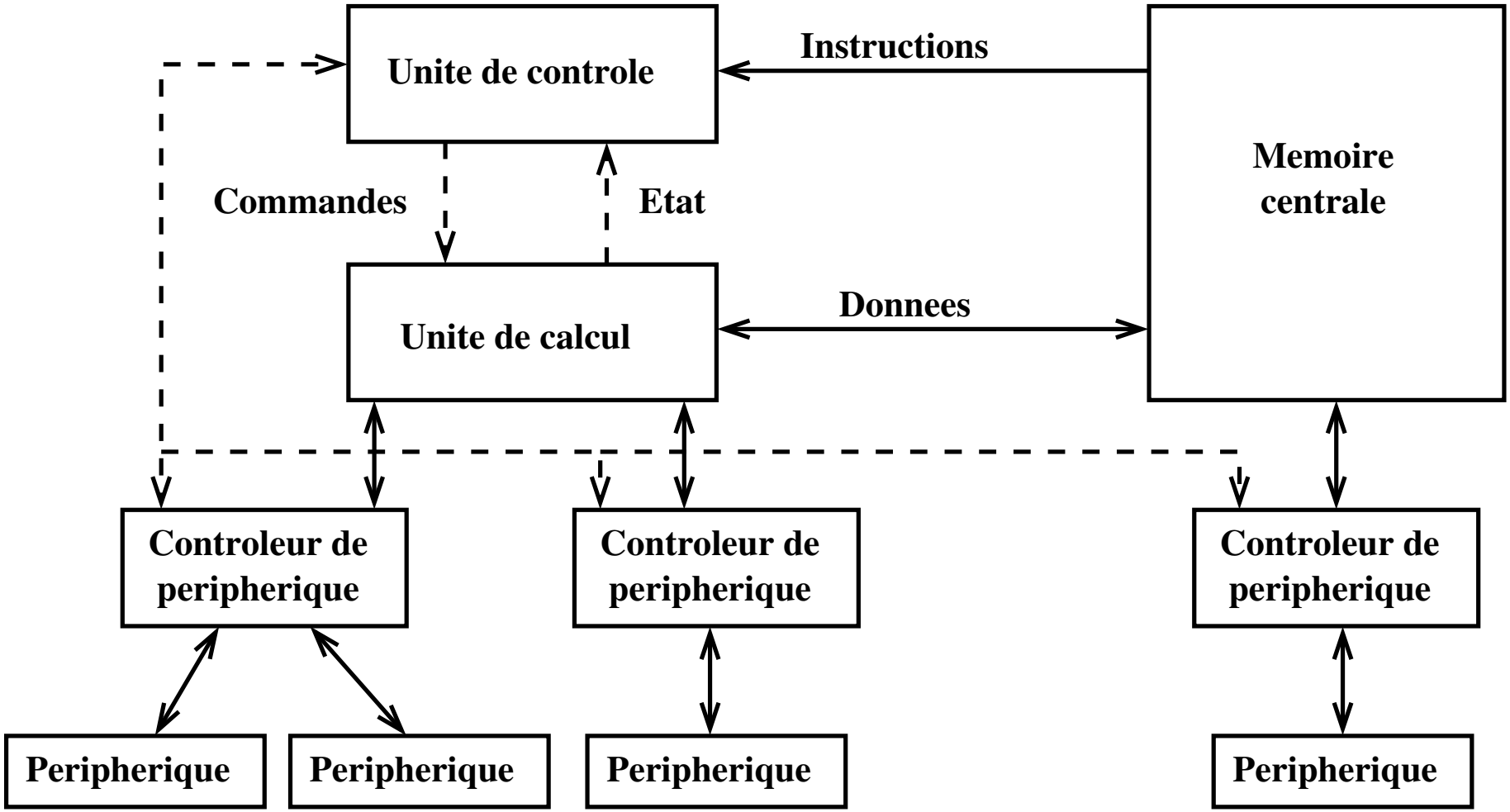
- Un ou plusieurs processeurs qui exécutent des programmes,
- Des moyens pour envoyer des ordres (clavier, souris, ...),
- Des moyens pour récupérer des résultats (écran, ...),
- Des moyens pour stocker de l'information (mémoire, disques, ...)
- Des moyens pour dialoguer avec d'autres dispositifs (interface réseau, ports, ...)

Organisation générale d'un ordinateur



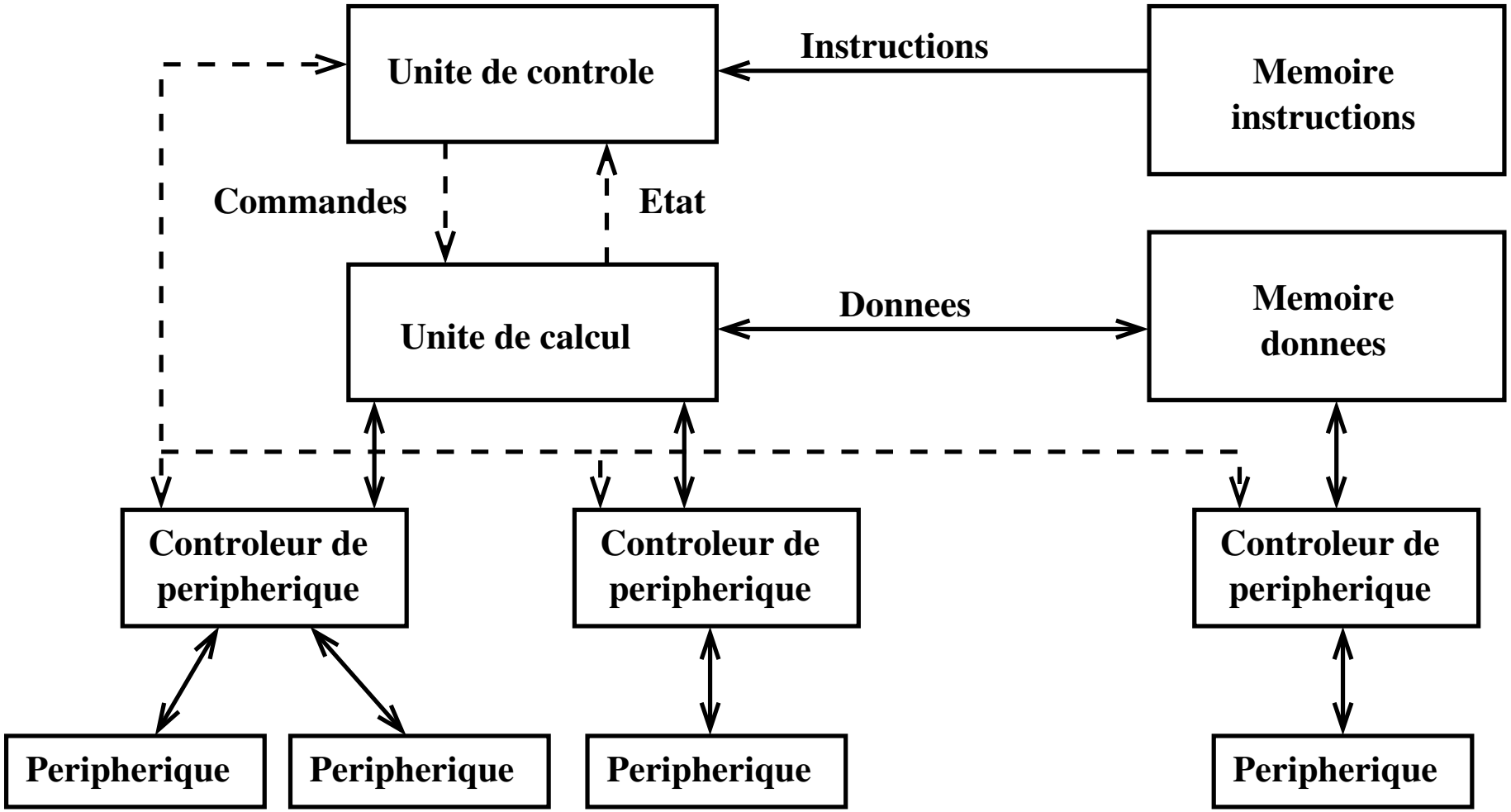
- Exécution de programmes par l'unité centrale
  - Lecture et écriture de données en mémoire centrale
  - Interactions avec l'extérieur
- Programmes et données stockées et véhiculées logiquement sous la forme de chiffres binaires ou bits (binary digits), physiquement sous la forme de signaux électroniques
- Développement d'un programme
  - Ecriture du programme dans un langage de "haut niveau" (e.g. Ada, C, ...)
  - Traduction du programme en langage machine (instructions plus rudimentaires)
  - Exécution du programme en langage machine

Modèle de Von Neuman



- Une mémoire commune aux données et aux programmes

Modèle de Harvard

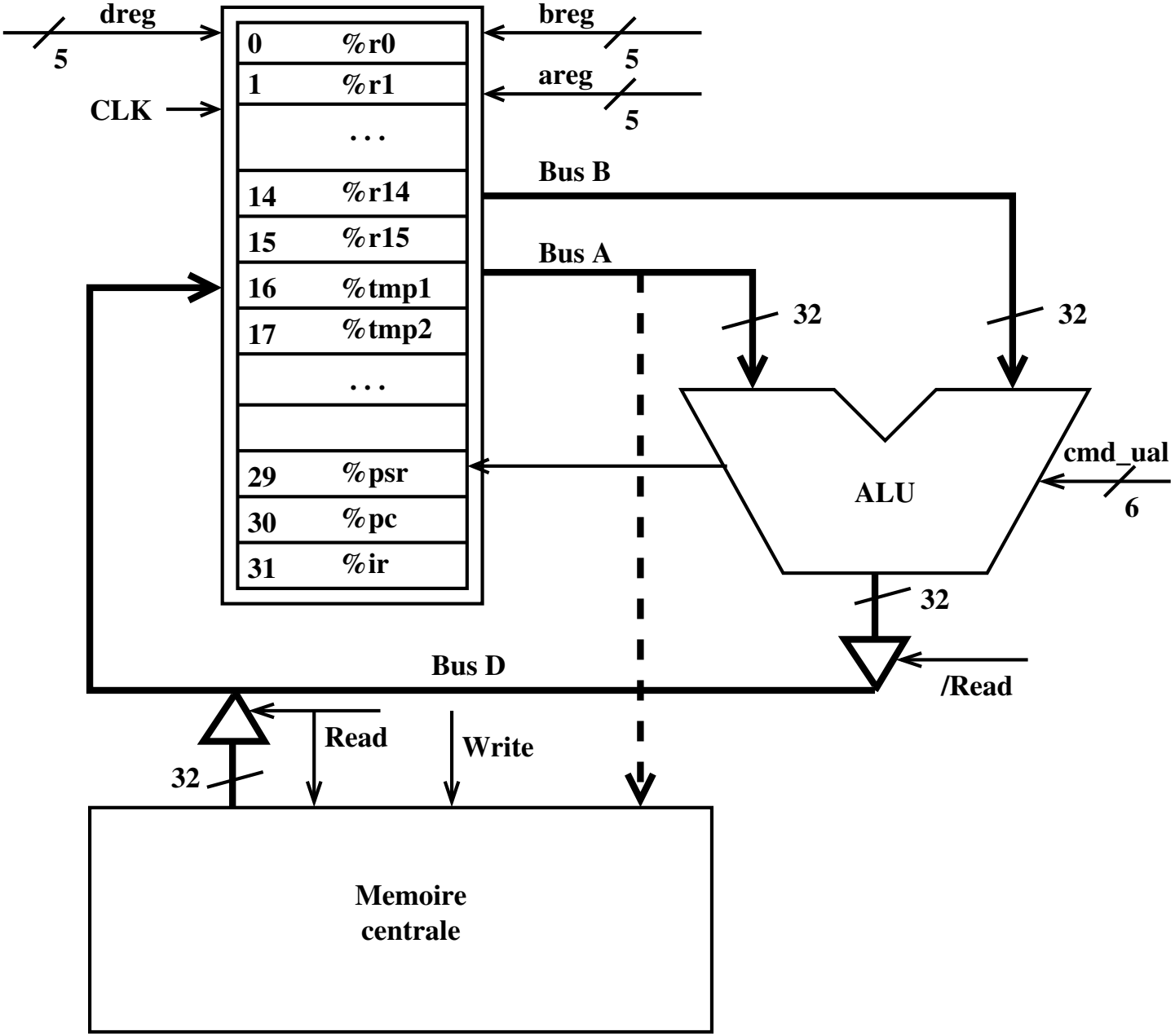


- Deux mémoires séparées pour les données et les programmes

Principe général de fonctionnement d'un processeur

- Unité centrale du processeur
  - Comprend l'unité de contrôle et l'unité de calcul
  - S'occupe de l'interprétation et de l'exécution des programmes
    - ★ Unité de calcul = Unité Arithmétique et Logique + zone de stockage de données temporaires (registres)
    - ★ Unité de contrôle = envoi des ordres à l'unité de calcul pour l'exécution des instructions du programme ⇒ interprétation de chaque instruction
- Mémoire centrale
  - Modèle de Von Neuman : les instructions et les données sont dans la même mémoire ⇒ architecture plus simple, pas d'accès en parallèle
  - Modèle de Harvard : les instructions et les données sont dans deux mémoires distinctes (intérêt : accès simultané aux deux mémoires)
- Accès aux périphériques via des contrôleurs de périphériques
  - Connexion physique du périphérique
  - Synchronisation du périphérique avec l'unité centrale par un protocole assurant que toute donnée échangée est reçue une et une seule fois

L'unité centrale



Principe général de l'exécution d'une instruction

- Adresse mémoire de l'instruction courante dans le Compteur Ordinal (registre PC : Program Counter)
- Code de l'instruction courante dans le registre instructions (registre IR : Instruction Register)
- Etapes pour l'exécution d'une instruction
  - Chargement du code de l'instruction courante dans le registre IR

$$IR \leftarrow Memoire[PC]$$

  - Décodage de l'instruction
  - Lecture éventuelle des opérandes
  - Exécution de l'opération
  - Sauvegarde éventuelle du résultat

Petit exemple introductif

- Mise en œuvre d'un processeur simpliste
  - Une mémoire de 16 mots de 4 bits
  - Un registre de données *ACC* sur 4 bits
  - Un jeu d'instructions très réduit

clr		Mise à 0 du registre *ACC*
ld	#v	Chargement de la valeur *v* dans *ACC*
st	a	Copie du contenu de *ACC* en mémoire à l'adresse *a*
add	a	$ACC \leftarrow ACC+$ contenu du mot mémoire d'adresse *a*
jmp	a	*a* est l'adresse mémoire de l'instruction suivante

- Exemple de programme

	ld	#3	$ACC \leftarrow 3$
	st	8	$MEM[8] \leftarrow ACC$
	add	8	$ACC \leftarrow ACC + MEM[8]$
etiq:	jmp	etiq	Boucle infinie

- Codage des instructions

clr		0000		add	a	0101 $a_3a_2a_1a_0$
ld	#v	0010 $v_3v_2v_1v_0$		jmp	a	0100 $a_3a_2a_1a_0$
st	a	0011 $a_3a_2a_1a_0$				

Exercice 1

- Donner le codage du programme suivant, en supposant que la première instruction se trouve à l'adresse 0 de la mémoire.

	ld	#3	$ACC \leftarrow 3$
	st	8	$MEM[8] \leftarrow ACC$
	add	8	$ACC \leftarrow ACC + MEM[8]$
etiq:	jmp	etiq	Boucle infinie

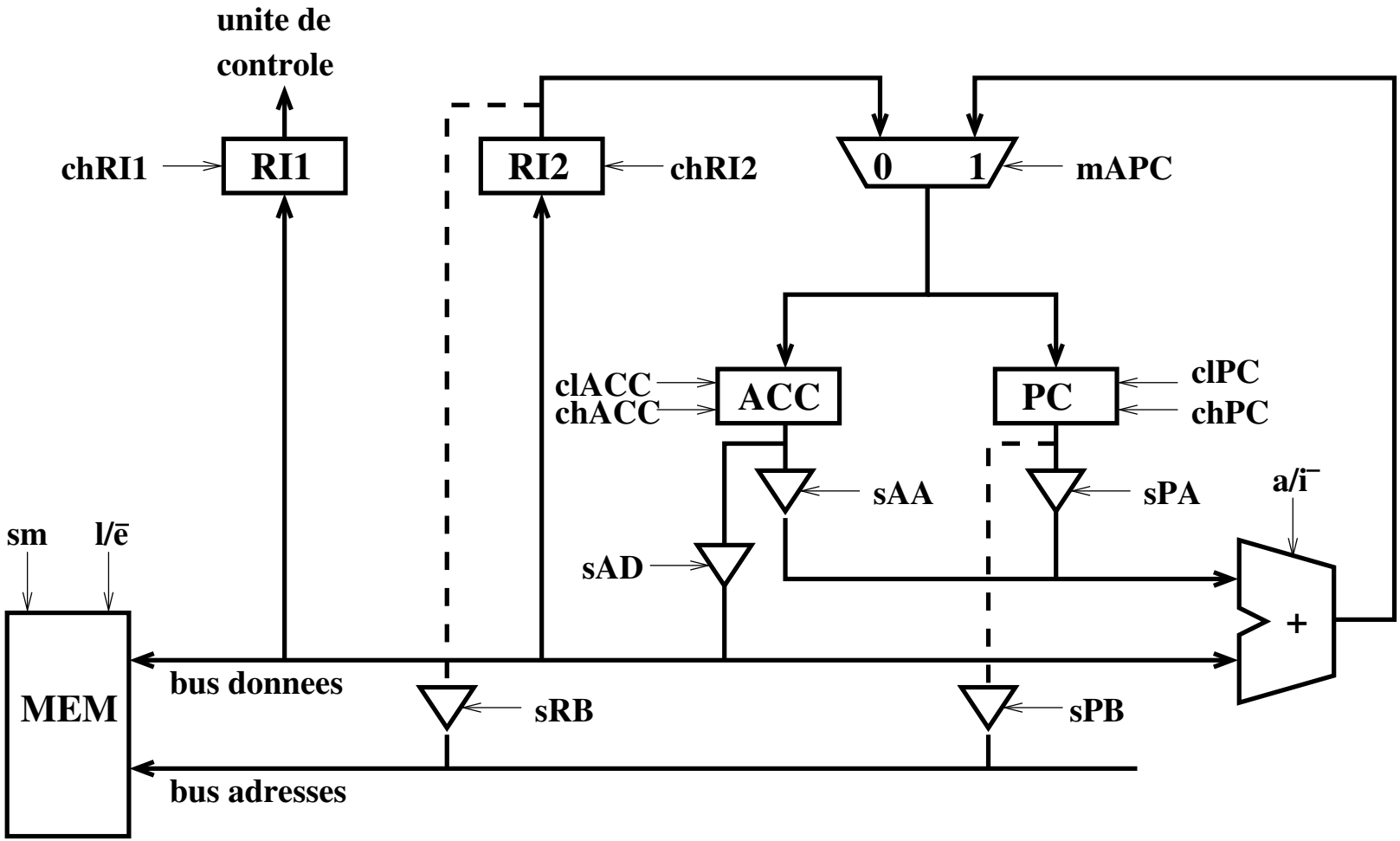
Petit exemple introductif : interprétation des instructions

```
PC ← 0; /* Le programme démarre à l'adresse 0 */
TantQue vrai Faire
    Selon MEM[PC]
        0000₂ : /* Instruction clr */
            ACC ← 0;
            PC ← PC + 1;
        0010₂ : /* Instruction ld */
            ACC ← MEM[PC+1];
            PC ← PC + 2;
        0011₂ : /* Instruction st */
            MEM[MEM[PC+1]] ← ACC;
            PC ← PC + 2;
        0101₂ : /* Instruction add */
            ACC ← ACC + MEM[PC+1];
            PC ← PC + 2;
        0100₂ : /* Instruction jmp */
            PC ← MEM[PC+1];
    FinSelon;
FinTantQue;
```

Petit exemple introductif : unité de calcul

- Principe général de fonctionnement
  - Adresse de l'instruction courante dans un registre PC
  - Code de l'instruction courante (4 ou 8 bits) dans un ou deux registres $RI1$, $RI2$
  - L'unité de calcul charge le code de l'instruction courante, puis exécute l'opération correspondante
- Ressources de l'unité de calcul
  - Un registre PC sur 4 bits
  - Deux registres $RI1$ et $RI2$ chacun sur 4 bits
  - Un registre ACC sur 4 bits
  - Une unité arithmétique capable de faire des additions et des incrémentations
- Mécanisme pour l'échange d'informations avec la mémoire
  - Choix de l'opération à effectuer $\Rightarrow$ deux signaux sm et $l/\overline{e}$
    - $sm = 0$: pas d'opération
    - $sm = 1$ et $l/\overline{e} = 0$: écriture en mémoire
    - $sm = 1$ et $l/\overline{e} = 1$: lecture de la mémoire
  - Donnée lue ou écrite $\Rightarrow$ un bus bidirectionnel
  - Adresse de lecture ou d'écriture $\Rightarrow$ un bus monodirectionnel

Petit exemple introductif : unité de calcul



Petit exemple introductif : traitement des instructions

- Instruction *clr*
  - Chargement du code de l'instruction dans le registre d'instruction
    RI1 $\leftarrow$ **MEM[PC]** : $sPB = 1$, $I/\overline{e} = 1$, $sm = 1$, $chRI1 = 1$
  - Passage au mot suivant du programme
    PC $\leftarrow$ **PC + 1** : $sPA = 1$, $mAPC = 1$, $chPC = 1$
  - Mise à 0 du registre *ACC*
    ACC $\leftarrow$ **0** : $clACC = 1$
- Instruction *ld v*
  - Chargement du code de l'instruction dans le registre d'instruction
    RI1 $\leftarrow$ **MEM[PC]** : $sPB = 1$, $I/\overline{e} = 1$, $sm = 1$, $chRI1 = 1$
  - Passage au mot suivant du programme
    PC $\leftarrow$ **PC + 1** : $sPA = 1$, $mAPC = 1$, $chPC = 1$
  - Chargement de *v* dans le registre d'instruction
    RI2 $\leftarrow$ **MEM[PC]** : $sPB = 1$, $I/\overline{e} = 1$, $sm = 1$, $chRI2 = 1$
  - Passage au mot suivant du programme
    PC $\leftarrow$ **PC + 1** : $sPA = 1$, $mAPC = 1$, $chPC = 1$
  - Copie de *v* dans le registre *ACC*
    ACC $\leftarrow$ **RI2** : $chACC = 1$

Petit exemple introductif : traitement des instructions

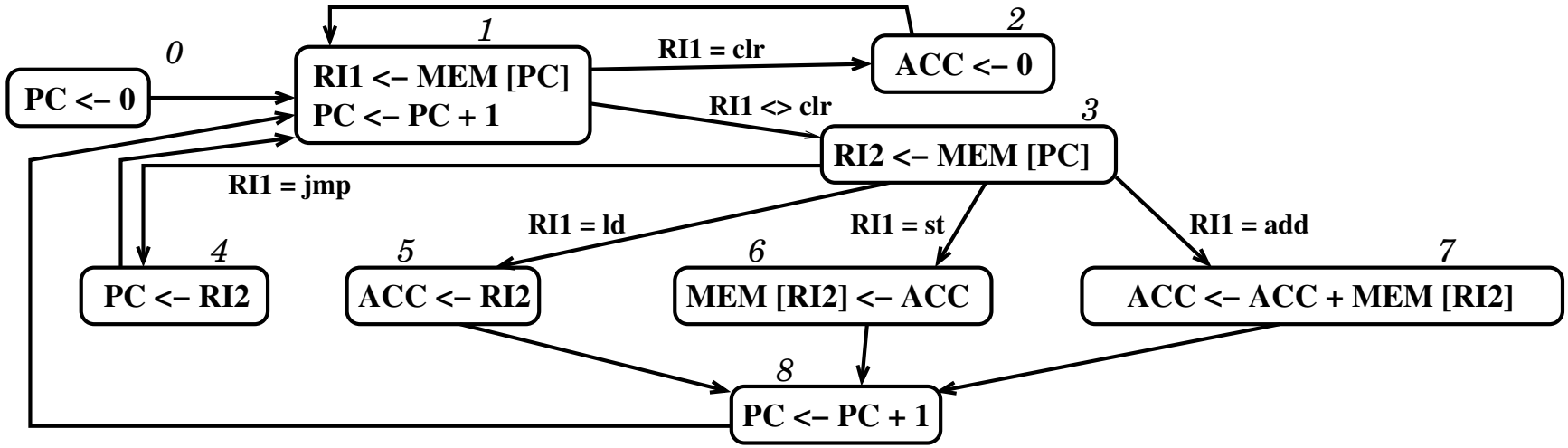
- Instruction *st a*
  - Chargement du code de l'instruction dans le registre d'instruction
    RI1 $\leftarrow$ **MEM[PC]** : $sPB = 1$, $I/\overline{e} = 1$, $sm = 1$, $chRI1 = 1$
  - Passage au mot suivant du programme
    PC $\leftarrow$ **PC + 1** : $sPA = 1$, $mAPC = 1$, $chPC = 1$
  - Chargement de v dans le registre d'instruction
    RI2 $\leftarrow$ **MEM[PC]** : $sPB = 1$, $I/\overline{e} = 1$, $sm = 1$, $chRI2 = 1$
  - Passage au mot suivant du programme
    PC $\leftarrow$ **PC + 1** : $sPA = 1$, $mAPC = 1$, $chPC = 1$
  - Copie du contenu de ACC en mémoire à l'adresse a
    MEM[RI2] $\leftarrow$ **ACC** : $sRB = 1$, $sAD = 1$, $sm = 1$
- Instruction *add a* (début)
  - Chargement du code de l'instruction dans le registre d'instruction
    RI1 $\leftarrow$ **MEM[PC]** : $sPB = 1$, $I/\overline{e} = 1$, $sm = 1$, $chRI1 = 1$
  - Passage au mot suivant du programme
    PC $\leftarrow$ **PC + 1** : $sPA = 1$, $mAPC = 1$, $chPC = 1$
  - Chargement de v dans le registre d'instruction
    RI2 $\leftarrow$ **MEM[PC]** : $sPB = 1$, $I/\overline{e} = 1$, $sm = 1$, $chRI2 = 1$

Petit exemple introductif : traitement des instructions

- Instruction *add a* (fin)
  - Passage au mot suivant du programme
    PC $\leftarrow$ **PC + 1** : $sPA = 1$, $mAPC = 1$, $chPC = 1$
  - Ajout de $MEM[a]$ au contenu de ACC
    ACC $\leftarrow$ **ACC + MEM[RI2]** : $sAA = 1$, $sRB = 1$, $l/\overline{e} = 1$, $sm = 1$, $a/\overline{i} = 1$, $mAPC = 1$, $chACC = 1$
- Instruction *jmp a*
  - Chargement du code de l'instruction dans le registre d'instruction
    RI1 $\leftarrow$ **MEM[PC]** : $sPB = 1$, $l/\overline{e} = 1$, $sm = 1$, $chRI1 = 1$
  - Passage au mot suivant du programme
    PC $\leftarrow$ **PC + 1** : $sPA = 1$, $mAPC = 1$, $chPC = 1$
  - Chargement de v dans le registre d'instruction
    RI2 $\leftarrow$ **MEM[PC]** : $sPB = 1$, $l/\overline{e} = 1$, $sm = 1$, $chRI2 = 1$
  - Copie de a dans le registre PC
    PC $\leftarrow$ **RI2** : $chPC = 1$

Petit exemple introductif : unité de commande

- Initialisation de PC à 0, puis répétition à l'infini du traitement d'une instruction



- Construction du circuit correspondant en utilisant les mêmes principes que pour le circuit de calcul du PGCD de deux entiers
- Un circuit possible
  - Une bascule D pour chaque état de l'unité de commande
  - Génération des commandes en fonction de l'état actif
    - ★ Exemple : $chACC = 1$ pour les états 5 et 7

Exercice 2

- Dérouler l'exécution du programme suivant (Etats du graphe, contenu des registres)

	ld	#3	$ACC \leftarrow 3$
	st	8	$MEM[8] \leftarrow ACC$
	add	8	$ACC \leftarrow ACC + MEM[8]$
etiq:	jmp	etiq	Boucle infinie

Caractéristiques principales du CRAPS

- Version simplifiée du SPARC version 8
- Processeur 32 bits
- Mémoire de 2^{32} octets
- Architecture de type *big-endian*
- Chaque instruction est codée sur un mot de 32 bits
- Processeur RISC (Reduced Instruction Set Computer) : jeu d'instructions réduit
- Machine de type *load/store* : un petit nombre d'instructions spécialisées pour le transfert des données entre la mémoire centrale et les registres
- 16 registres généraux, un compteur ordinal, un registre instruction, un registre d'états, ...
- **En travaux pratiques, mise en œuvre d'un mini-CRAPS, puis programmation du CRAPS complet**

Langages de programmation des processeurs

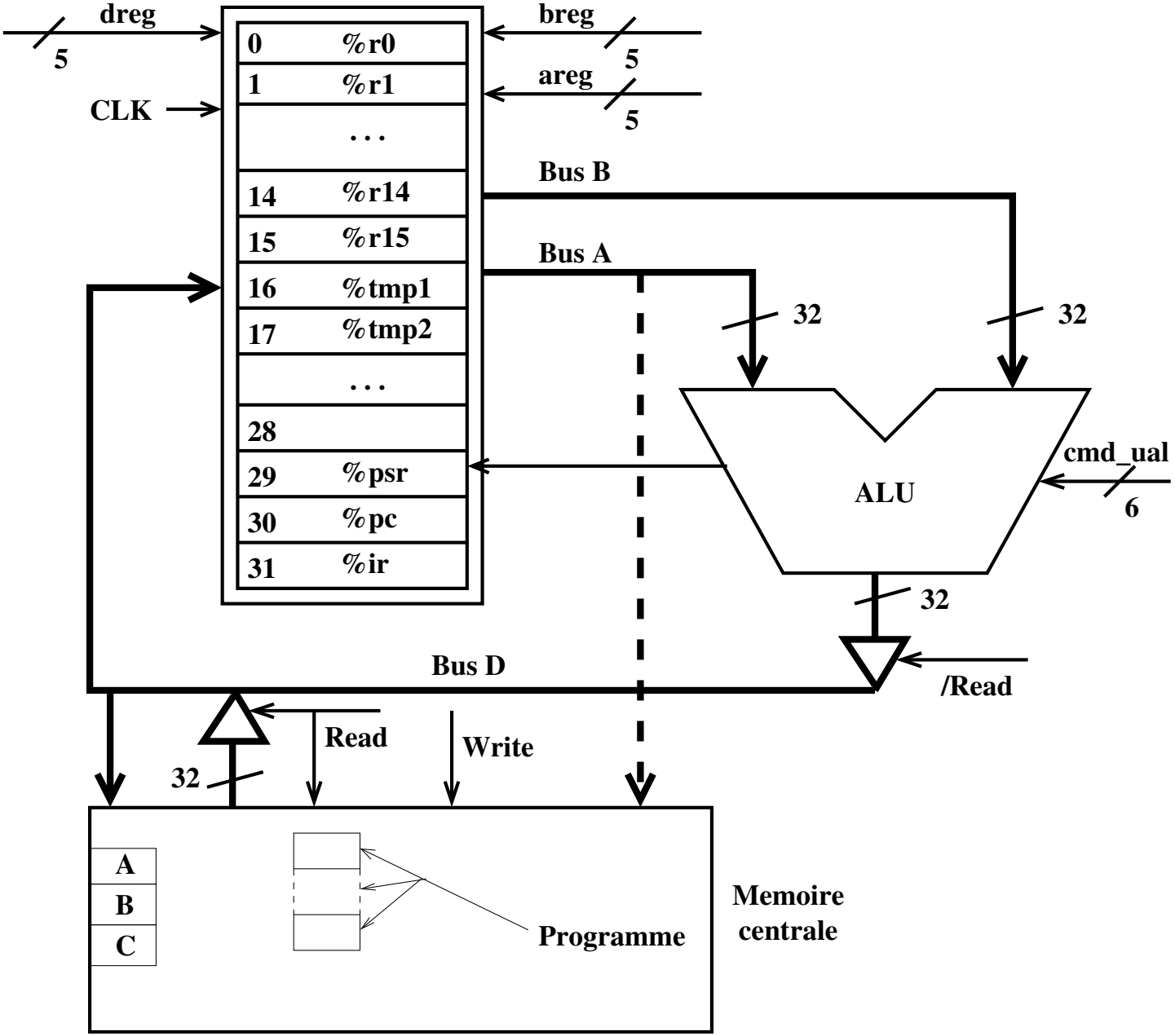
- Le processeur ne comprend que le binaire $\Rightarrow$ toute instruction s'exprime sous la forme d'une séquence de 0 et de 1
- L'écriture d'un programme en binaire est difficile (peu lisible $\Rightarrow$ risque élevé d'erreur)
- Association d'un nom à chaque instruction pour faciliter le travail
- Une instruction du processeur est une opération élémentaire de l'unité de calcul $\Rightarrow$ un programme nécessite de très nombreuses instructions et manque souvent de structure
- Utilisation de langages structurés de plus haut niveau (une instruction du langage correspond à une séquence d'instructions du processeur) : Java, Pascal, C, ...
- Compilateurs pour traduire les programmes en langage de haut niveau en programmes en langage du processeur
- Un compilateur différent pour chaque processeur

Exemple de programme

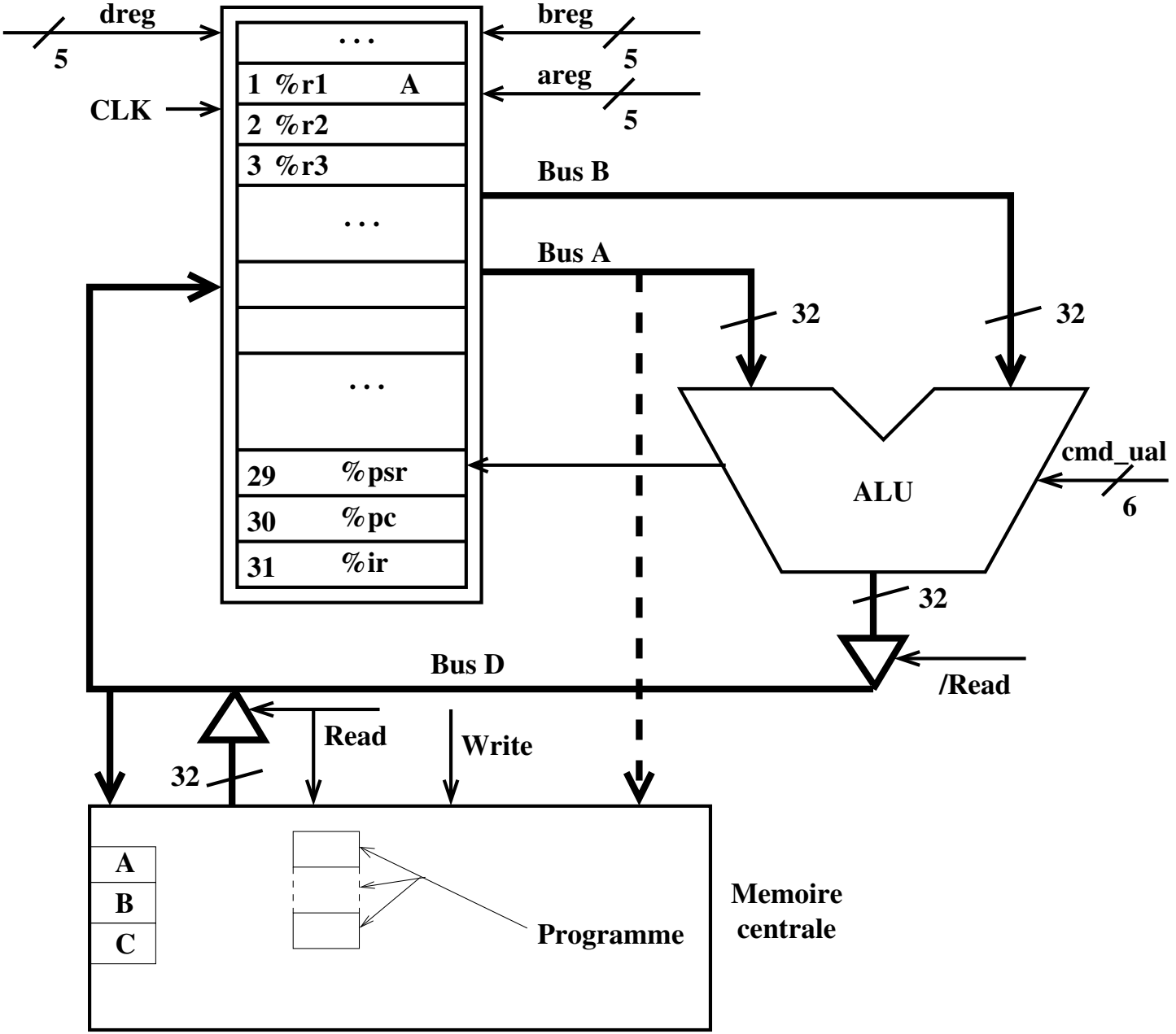
```
if (A <= B)
        C = A + B;
else
        C = A - B;
```

- Addition/soustraction de deux variables avec résultat dans une troisième variable
- Comparaison avec saut éventuel
- Saut inconditionnel

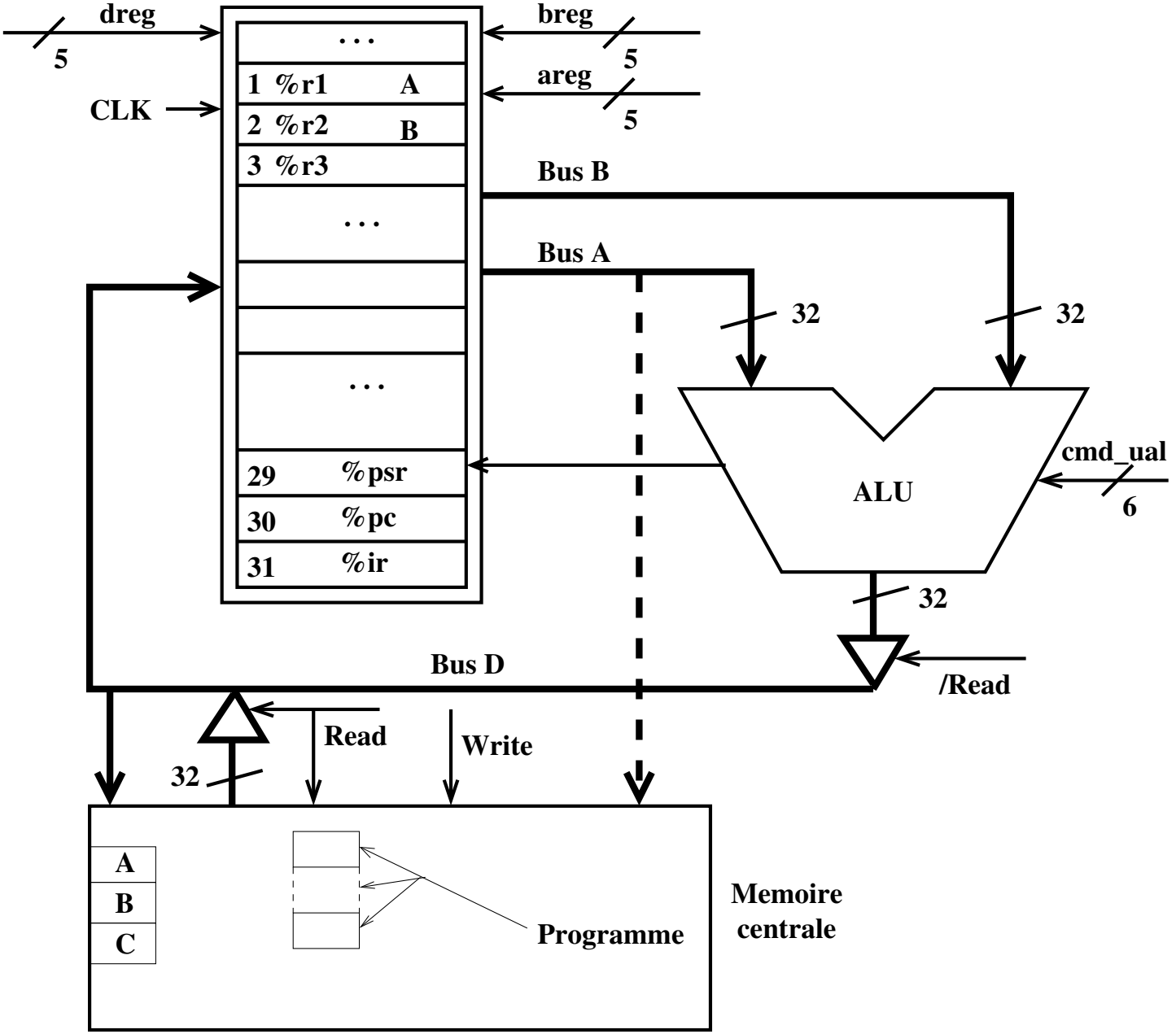
Point de départ : A et B en mémoire



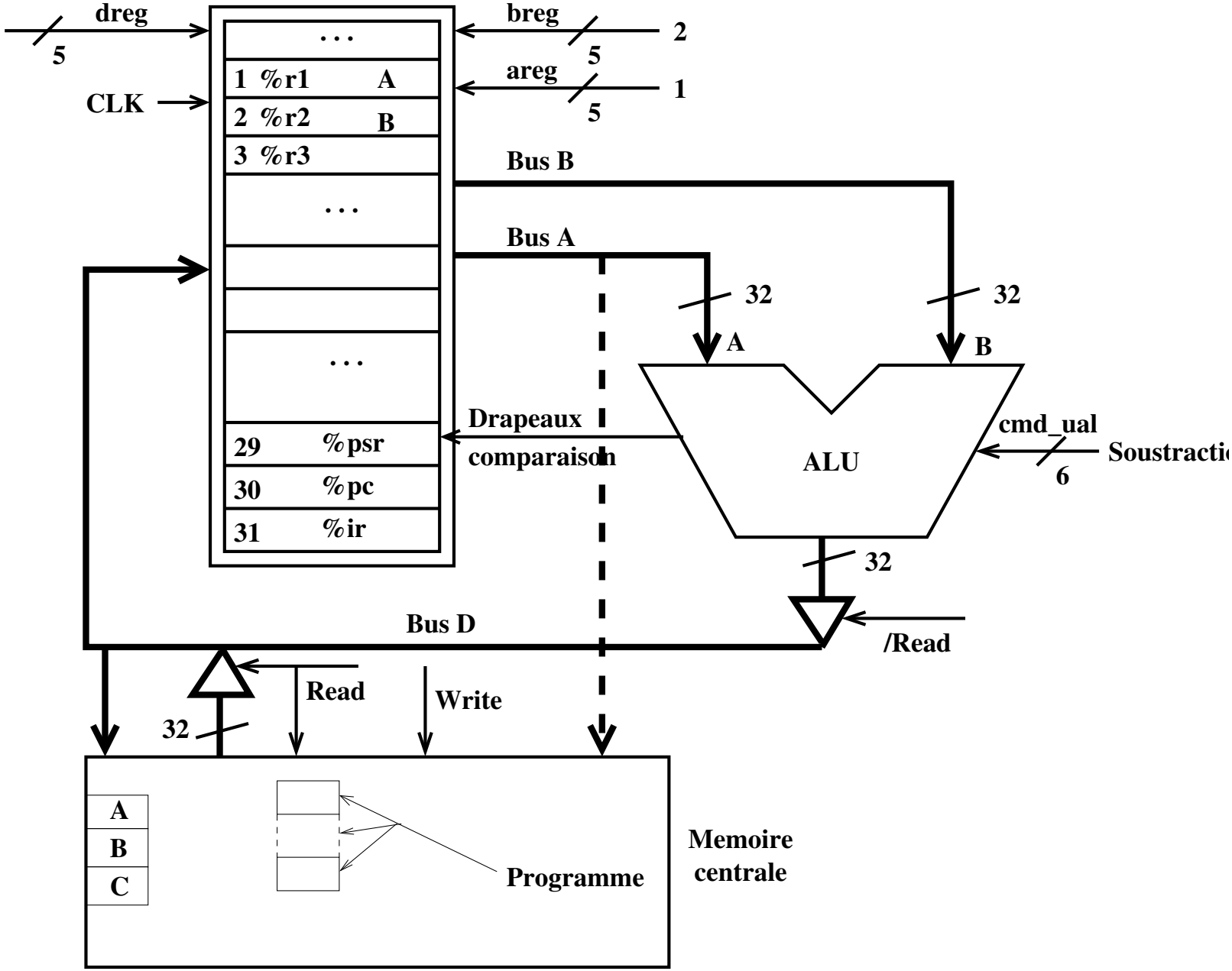
Chargement de A dans un registre



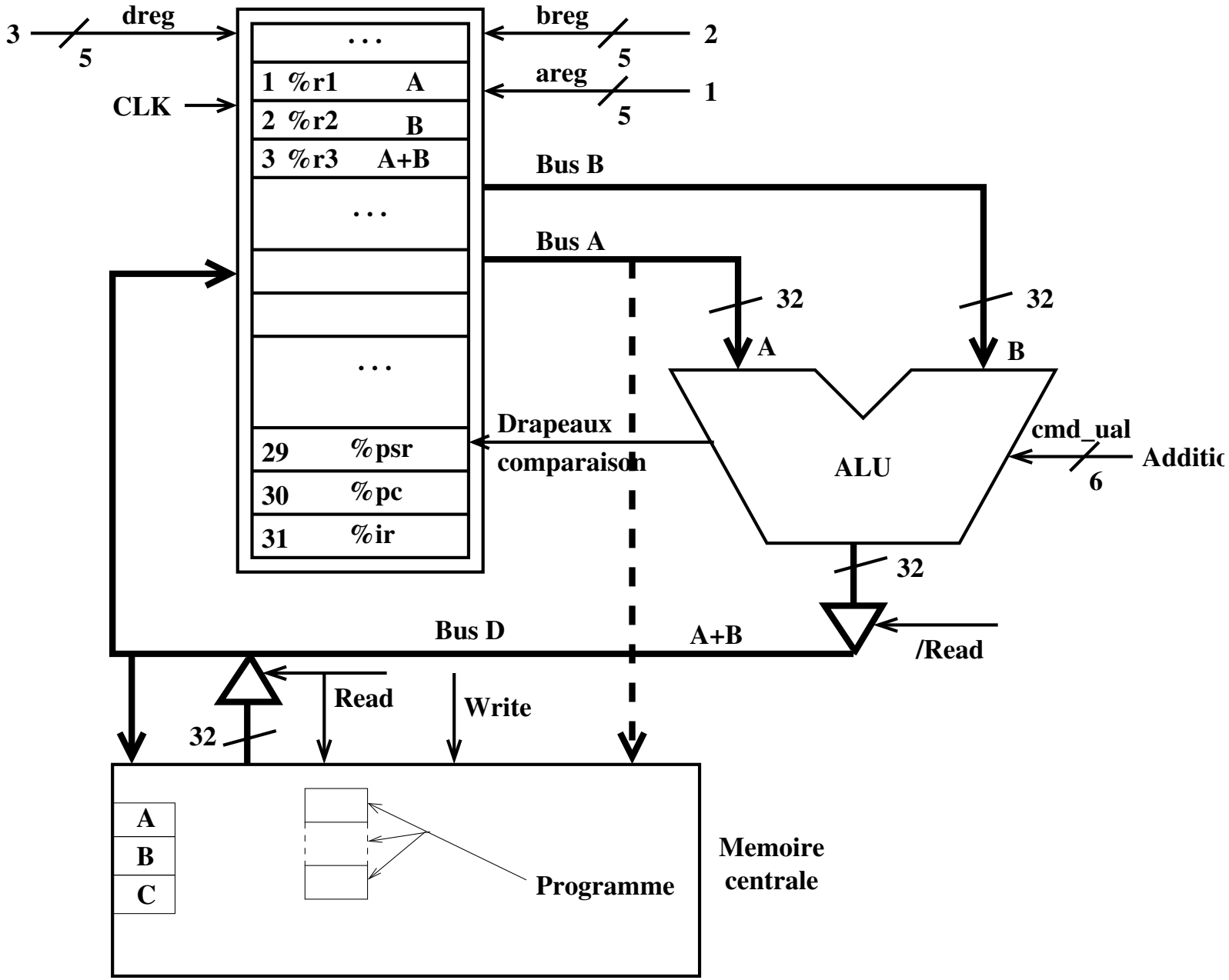
Chargement de B dans un registre



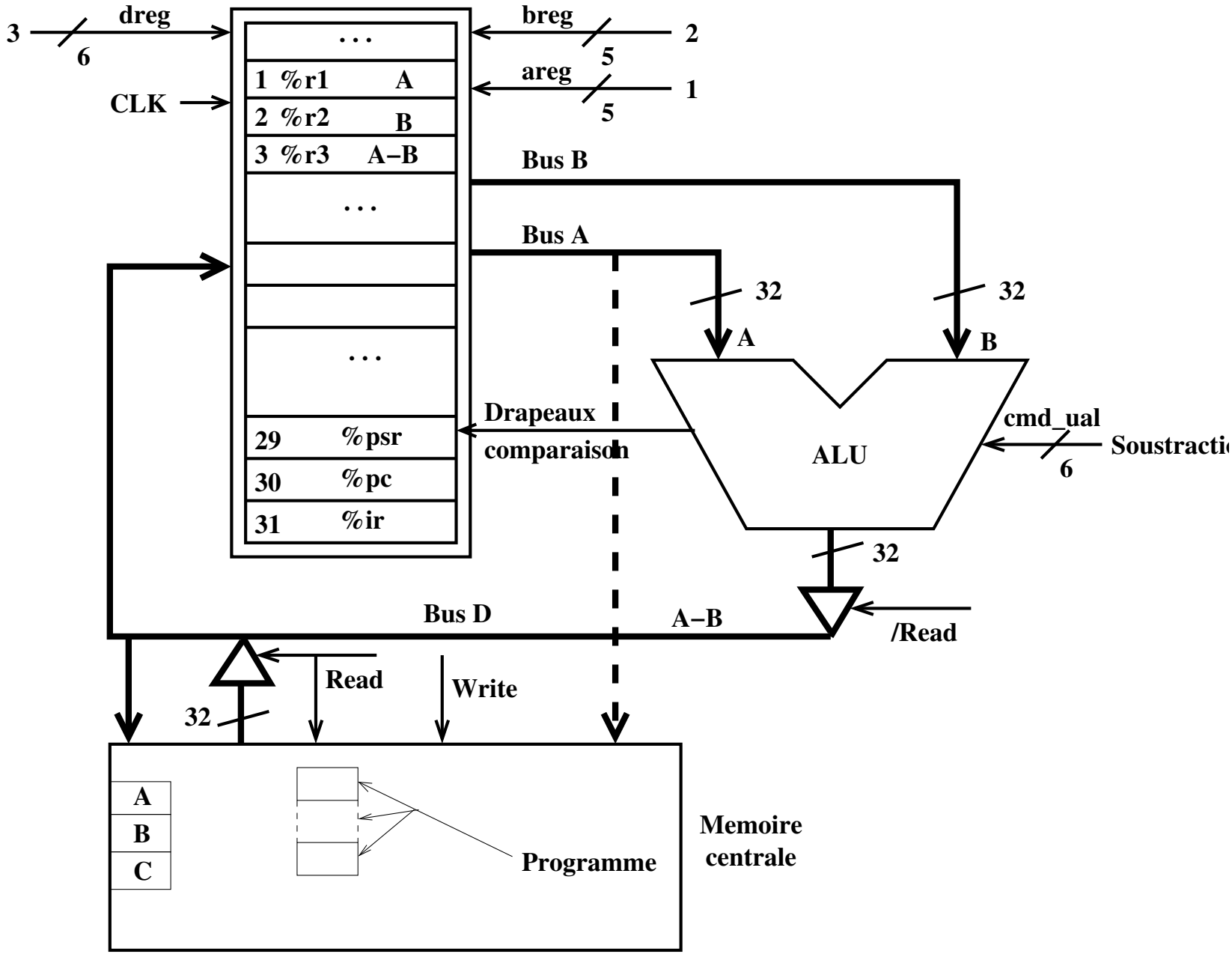
Comparaison de A et de B



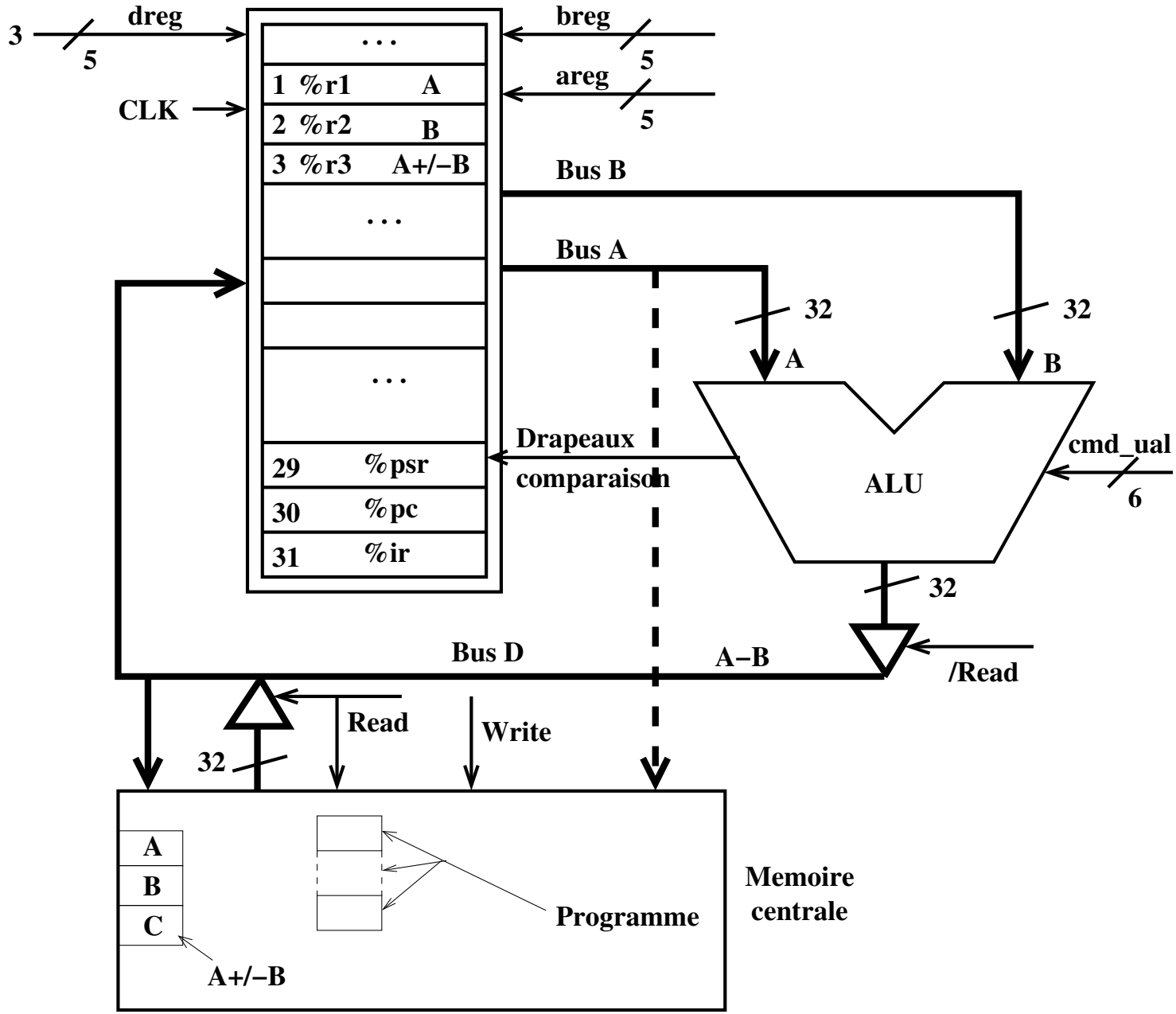
Calcul de $A + B$


dreg
3
5
CLK
breg
2
5
areg
1
5
1 %r1 A
2 %r2 B
3 %r3 A+B
29 %psr
30 %pc
31 %ir
Bus B
Bus A
32
A
32
B
Drapeaux
comparaison
ALU
cmd_ual
6
32
Bus D
A+B
/Read
Read
Write
32
A
B
C
Programme
Memoire
centrale

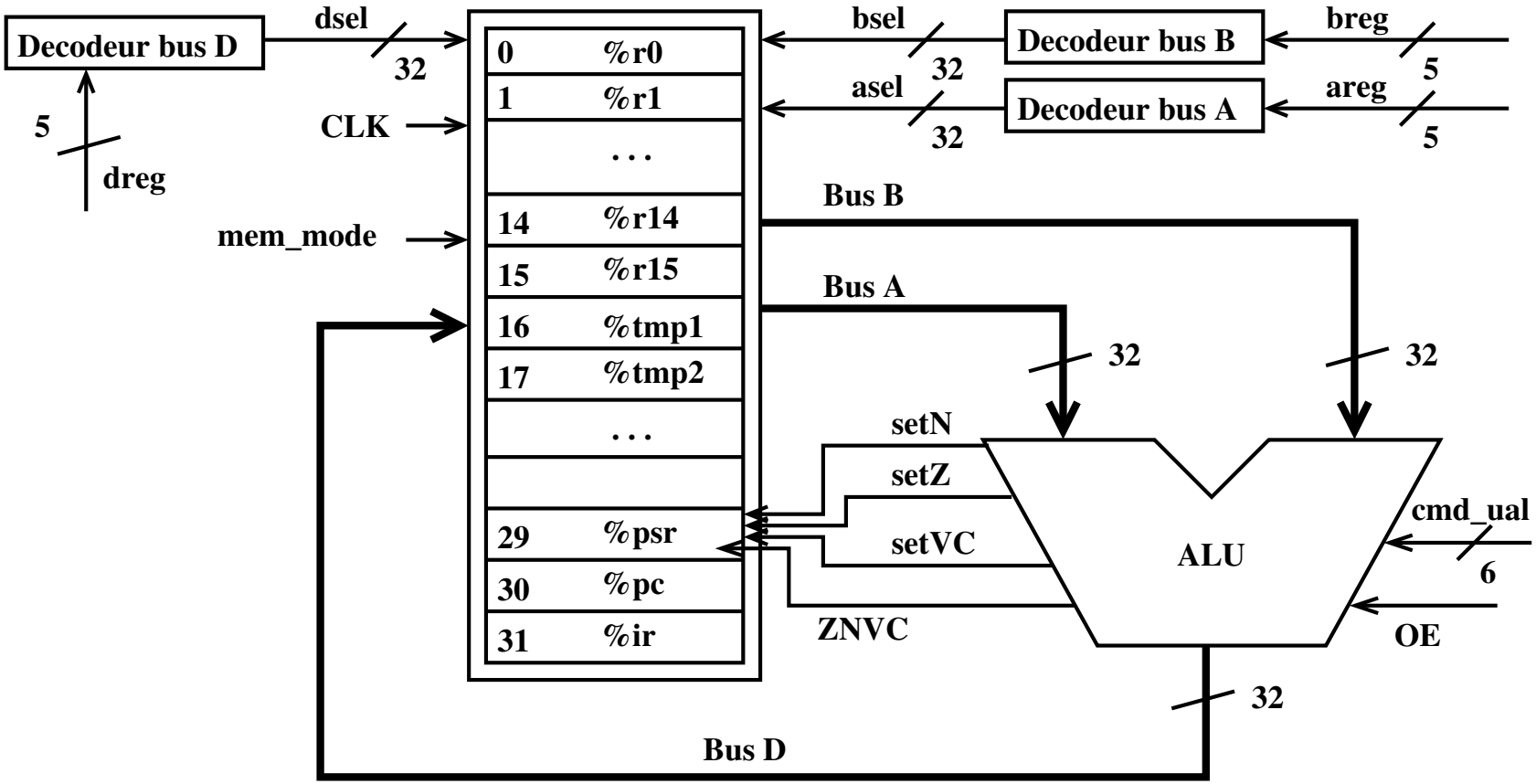
Calcul de $A - B$


3
dreg
6
CLK
breg
5
2
areg
5
1
1 %r1 A
2 %r2 B
3 %r3 A–B
29 %psr
30 %pc
31 %ir
Bus B
Bus A
32
A
32
B
Drapeaux
comparaison
ALU
cmd_ual
6
32
Bus D
A–B
/Read
Read
Write
32
A
B
C
Programme
Memoire
centrale

Sauvegarde du résultat en mémoire

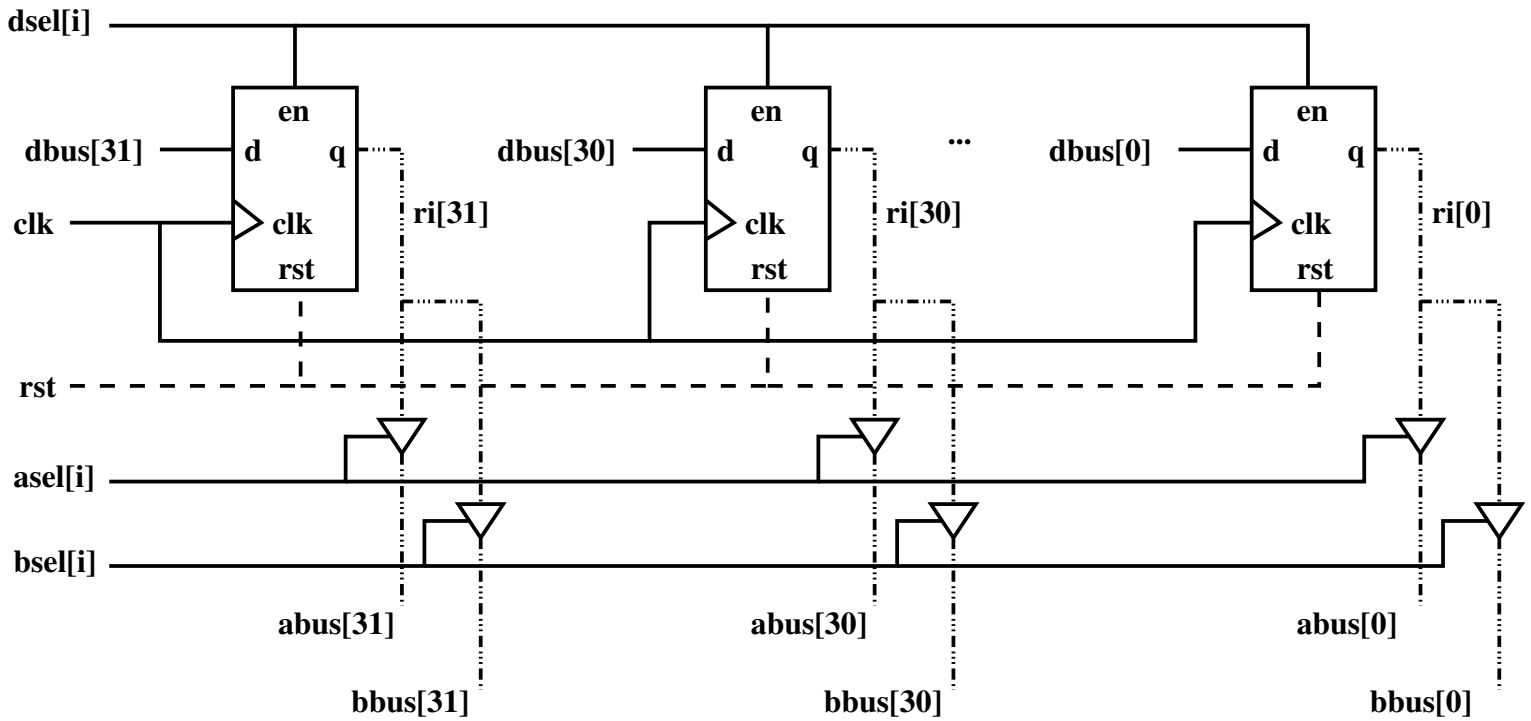


Architecture pour l'addition



- 16 registres utilisateurs (*%r0 à %r15*), 1 registre d'état (*%psr*)
- 3 bus pour les échanges de données entre l'UAL et les registres

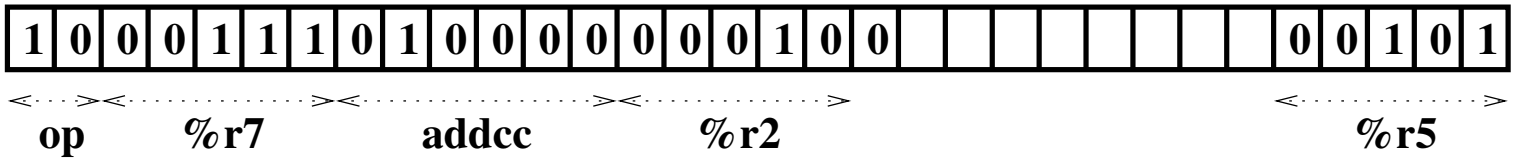
Le composant de mémorisation : le registre %ri



- Le registre contient la valeur $\%ri[31] \ldots \%ri[0]$
- A chaque front montant de l'horloge *clk*, la valeur sur le bus *D* est chargée dans le registre ssi $dsel[i] = 1$
- *rst* permet la remise à 0 du registre, indépendamment de l'horloge
- La valeur contenue dans le registre est placée sur le bus *A* et/ou le bus *B* en fonction des valeurs de $asel[i]$ et $bsel[i]$

Déroulement de l'intruction *addcc %r2 %r5 %r7*

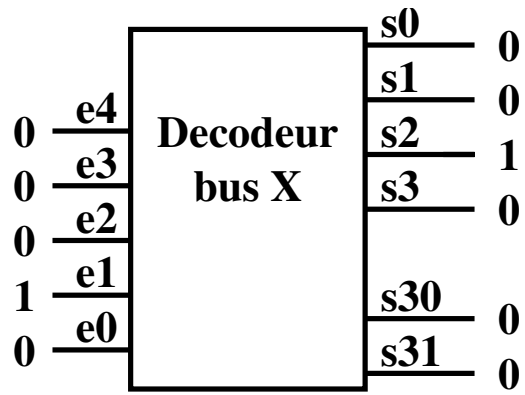
- Code de l'instruction



1. Mise en place des entrées pour l'opération d'addition
   - Le contenu du registre *%r2* est placé sur le bus *A* (**asel[2] = 1**)
   - Le contenu du registre *%r5* est placé sur le bus *B* (**bsel[5] = 1**)
   - La commande d'addition est envoyée à l'unité arithmétique et logique (**cmd_alu = 010000**)
2. Opération d'addition
   - L'unité arithmétique et logique effectue l'addition des deux valeurs et place le résultat sur le bus *D*
   - L'unité arithmétique et logique génère les indicateurs ZNVC
3. Sauvegarde du résultat
   - Le résultat de l'addition est mémorisé dans *%r7* (**dsel[7] = 1**)
   - Les valeurs de ZNVC sont mémorisées dans le registre *%psr* (**setN = 1, setZ = 1, setVC = 1**)

Principe des décodeurs associés aux bus

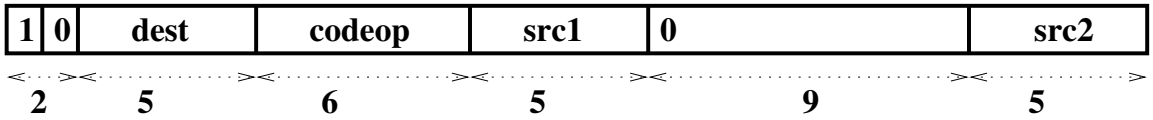
- Activation de la ligne de sortie s_i avec $i = e_5 e_4 e_3 e_2 e_1 e_0$



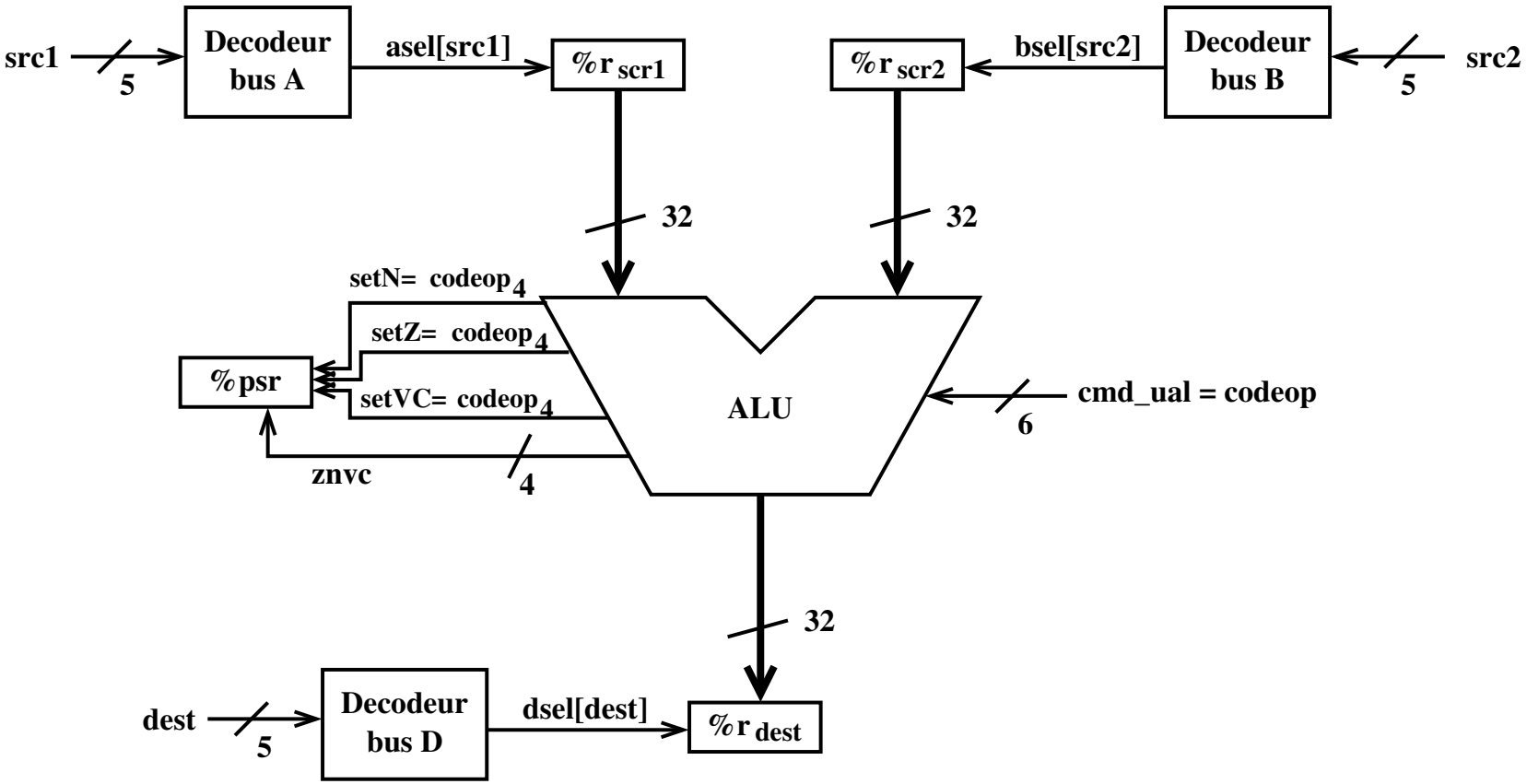
$$
\begin{aligned}
s_0 &= \overline{e_4} \cdot \overline{e_3} \cdot \overline{e_2} \cdot \overline{e_1} \cdot \overline{e_0} \\
s_1 &= \overline{e_4} \cdot \overline{e_3} \cdot \overline{e_2} \cdot \overline{e_1} \cdot e_0 \\
s_2 &= \overline{e_4} \cdot \overline{e_3} \cdot \overline{e_2} \cdot e_1 \cdot \overline{e_0} \\
&\ldots \\
s_{31} &= e_4 \cdot e_3 \cdot e_2 \cdot e_1 \cdot e_0
\end{aligned}
$$

Synthèse d'une instruction d'addition ou de soustraction

- Code de l'instruction



- Vue d'ensemble du déroulement de l'instruction



Exercice 3

- Donner les valeurs des champs de l'instruction

$$orcc \quad \%r2, \%r4, \%r6$$