Structure des ordinateurs

Pierre Wolper

Email: pw@montefiore.ulg.ac.be

URL: http: //www.montefiore.ulg.ac.be/~pw/
http: //www.montefiore.ulg.ac.be/
~pw/cours/struct.html

Références

Stephen A. Ward and Robert H. Halstead, *Computation Structures*, MIT Press, 1990.

http://6004.lcs.mit.edu.

M. Ben-Ari, Principles of Concurrent and Distributed Programming, Prentice Hall, 1990.

Objectifs du cours

1. **Langage machine et construction d'un ordinateur**

   (a) Etudier en détail une réalisation simple de machine programmable.

   (b) Aborder les techniques permettant d'améliorer les performances des processeurs.

   (c) Présenter les concepts utilisés pour simplifier la programmation des machines : gestion d'interruption et mode privilégié, mémoire virtuelle, ...

2. **Systèmes d'exploitation et programmation parallèle**

(a) Introduire la programmation par processus et le fonctionnement d'un noyau de système.

(b) Approfondir les notions relatives à la programmation par processus parallèles et de la communication entre processus.

(c) Etudier les problèmes classiques de programmation parallèle.

(d) Présenter quelques notions sur l'évolution des ordinateurs et notamment les machines parallèles et leur programmation.

Travaux pratiques

- Exercices sur la conception d'un processeur et la micro-programmation.

- Exercices et projet sur la programmation en assembleur et la conception d'un noyau de système.

- Exercices et projet sur la programmation par processus parallèles et les mécanismes de communication entre processus (langage C).

Electronique Digitale : rappels

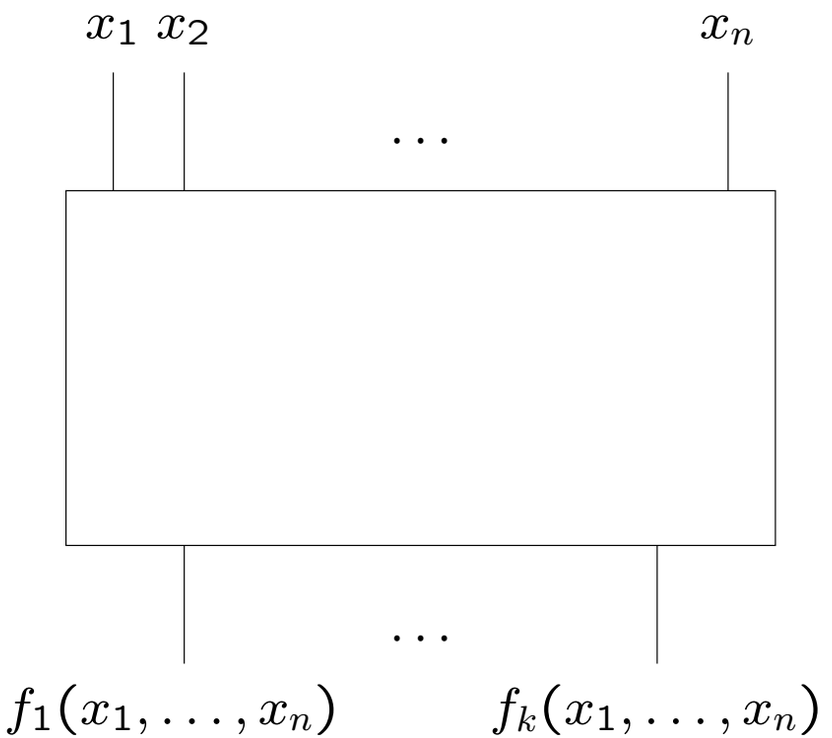
Nous utiliserons des notions élémentaires d'électronique digitale.

- En électronique digitale on utilise un signal électrique (tension) pour représenter une information booléenne (binaire) $(0, 1)$, par exemple $[0, 1]$ volts représente 0 et $[2, 3]$ volts représente 1.

- Les circuits de l'électronique digitale permettent d'effectuer des opérations sur les informations booléennes représentées.

- Nous utiliserons trois types de circuits : les circuits combinatoires, les éléments de mémorisation et les circuits séquentiels synchrones.

Electronique Digitale
rappels : les circuits combinatoires

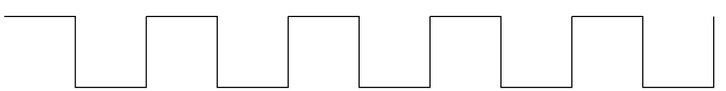


- Chaque f_i est une fonction booléenne $(\wedge, \vee, \neg)$ de $x_1, \ldots, x_n$.

- Si les entrées (x_i) changent, il faut un certain délai pour que les sorties (f_i) aient une valeur correcte.

Electronique Digitale
rappels : les éléments de mémorisation

- CLK est un signal d'horloge :

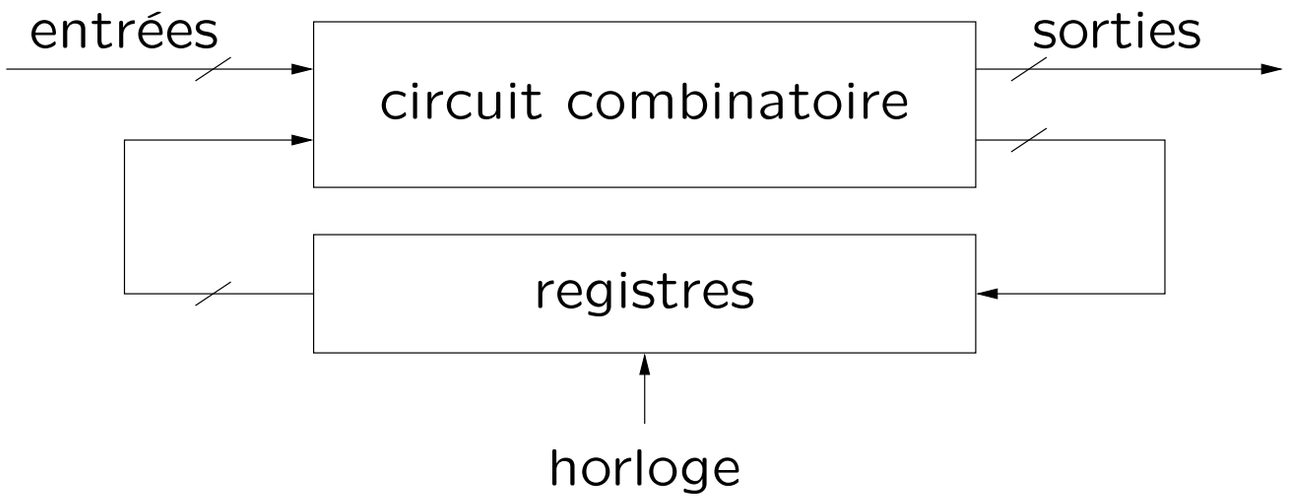


- Lorsque LD est actif (à 1), les valeurs de $x_1, \ldots, x_n$ sont mémorisées à la transition ($0 \rightarrow 1$) d'horloge suivante à l'adresse donnée par $a_1, \ldots, a_k$.

- Les valeurs des sorties $y_1, \ldots, y_n$ sont égales aux valeurs mémorisées à l'adresse $a_1, \ldots, a_k$.

- Un registre est une mémoire à une seule place (pas d'adresse).

Electronique Digitale
rappels : les circuits séquentiels synchrones



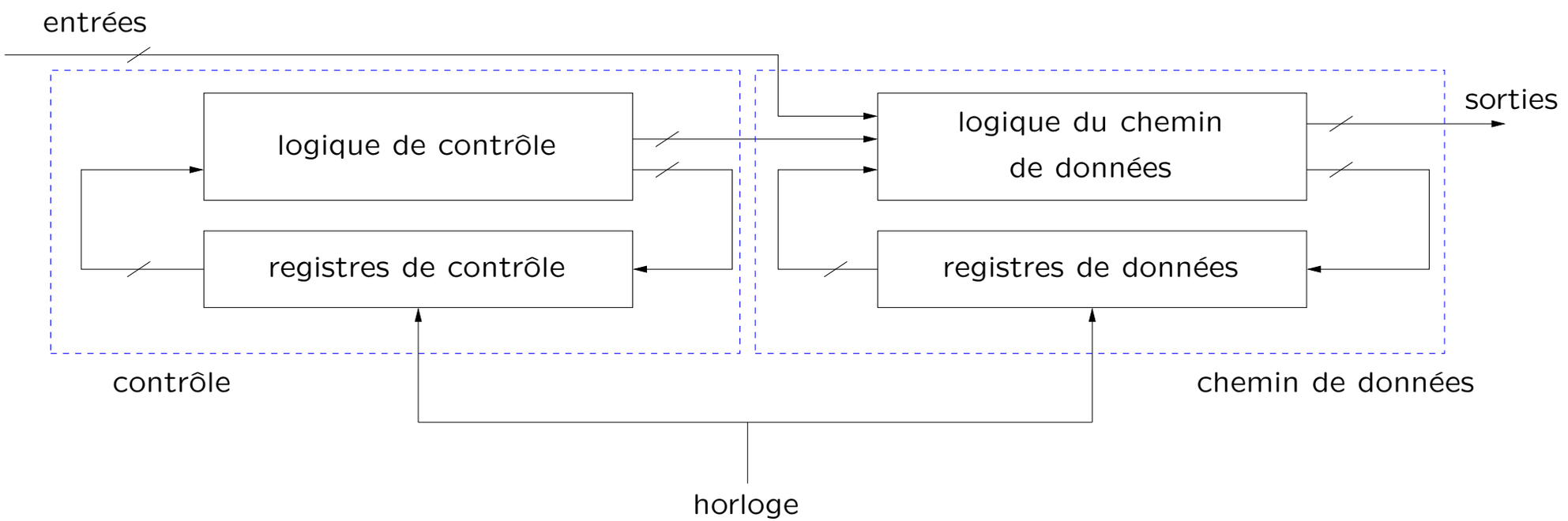
- A chaque transition d'horloge, on charge dans les registres une fonction booléenne des entrées et du contenu précédent des registres.

- Les sorties sont une fonction booléenne des entrées et du contenu des registres.

Des circuits synchrones aux processeurs

Un processeur est un circuit séquentiel synchrone avec quelques particularités.

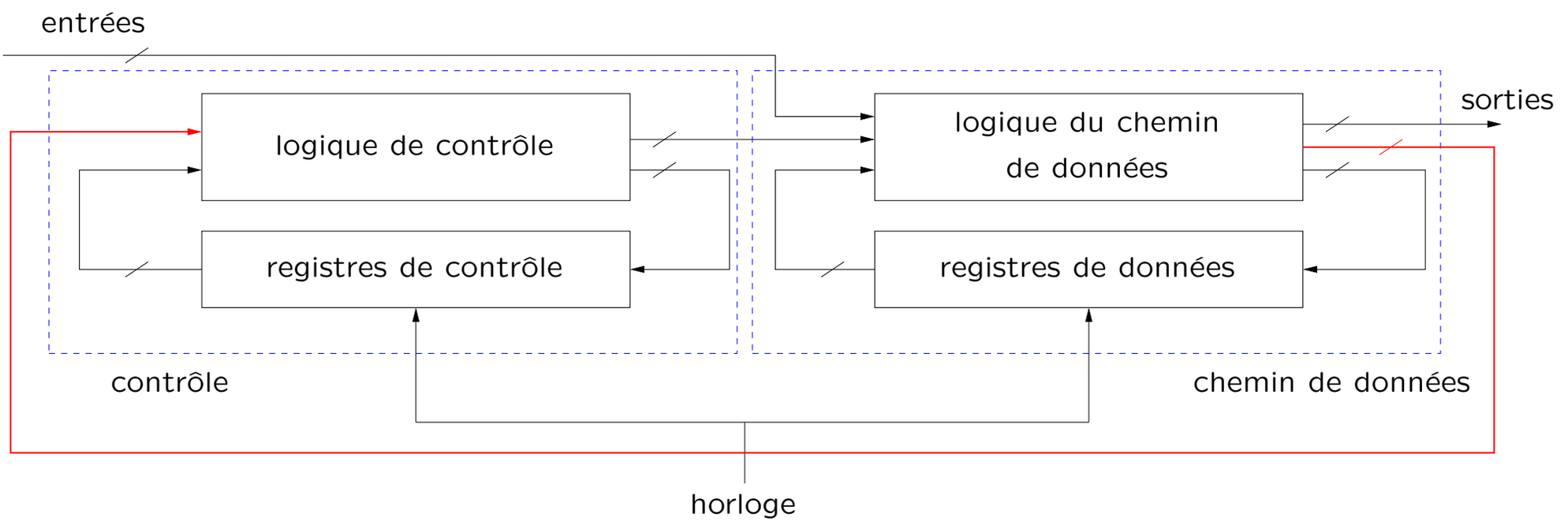
1. Le circuit est organisé en une partie “données” et une partie “contrôle”. La partie “données” effectue des opérations sur des données mémorisées ; le “contrôle” détermine les opérations qui sont effectuées.



2. Dans le cadre de la partie “données”, on incorpore une mémoire de grande capacité (la mémoire principale de la machine).

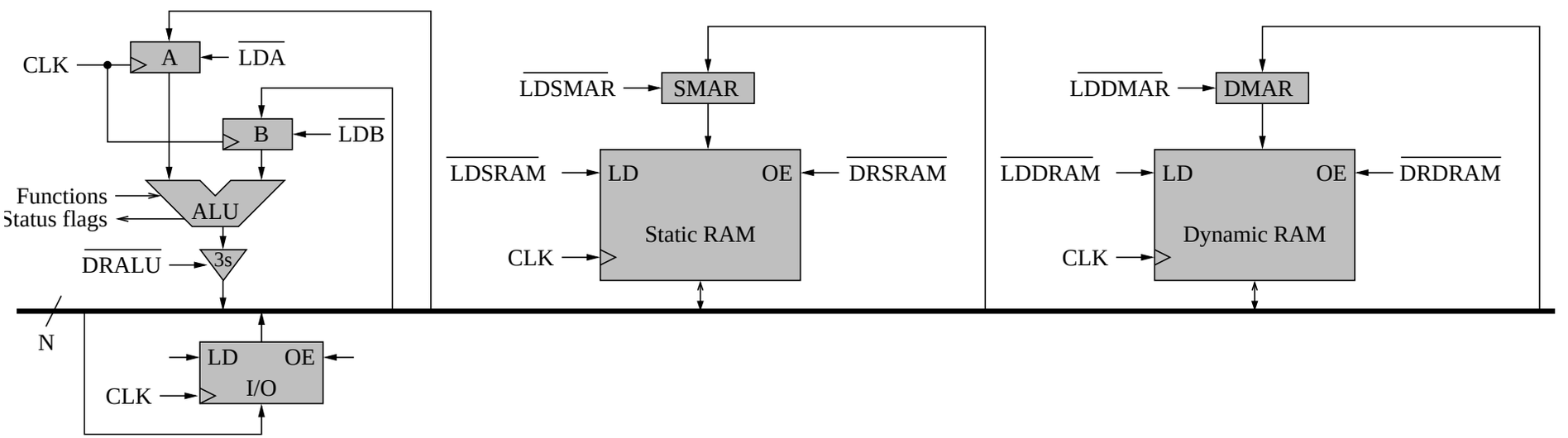
3. La partie contrôle a un comportement qui dépend de la partie données :

   - contrôle influencé par le résultat des opérations sur les données ;
   - contrôle déterminé par un programme conservé dans la mémoire du chemin de données.



Un chemin de données simple

Dans le chemin de données ci-dessous, il y a des éléments de mémorisation et un élément de calcul (circuit combinatoire) interconnecté par un *bus*.



Le bus est est un ensemble de lignes d'interconnexion partagées.

- Chaque élément connecté en sortie sur le bus peut être actif (0 ou 1) ou inactif (déconnecté) – on parle de logique 3 états (3 states).

- Il s'agit d'un bus simple à contrôle centralisé géré par l'unité de contrôle.

Les signaux nécessaires au fonctionnement du chemin de données seront générés par l'unité de contrôle.

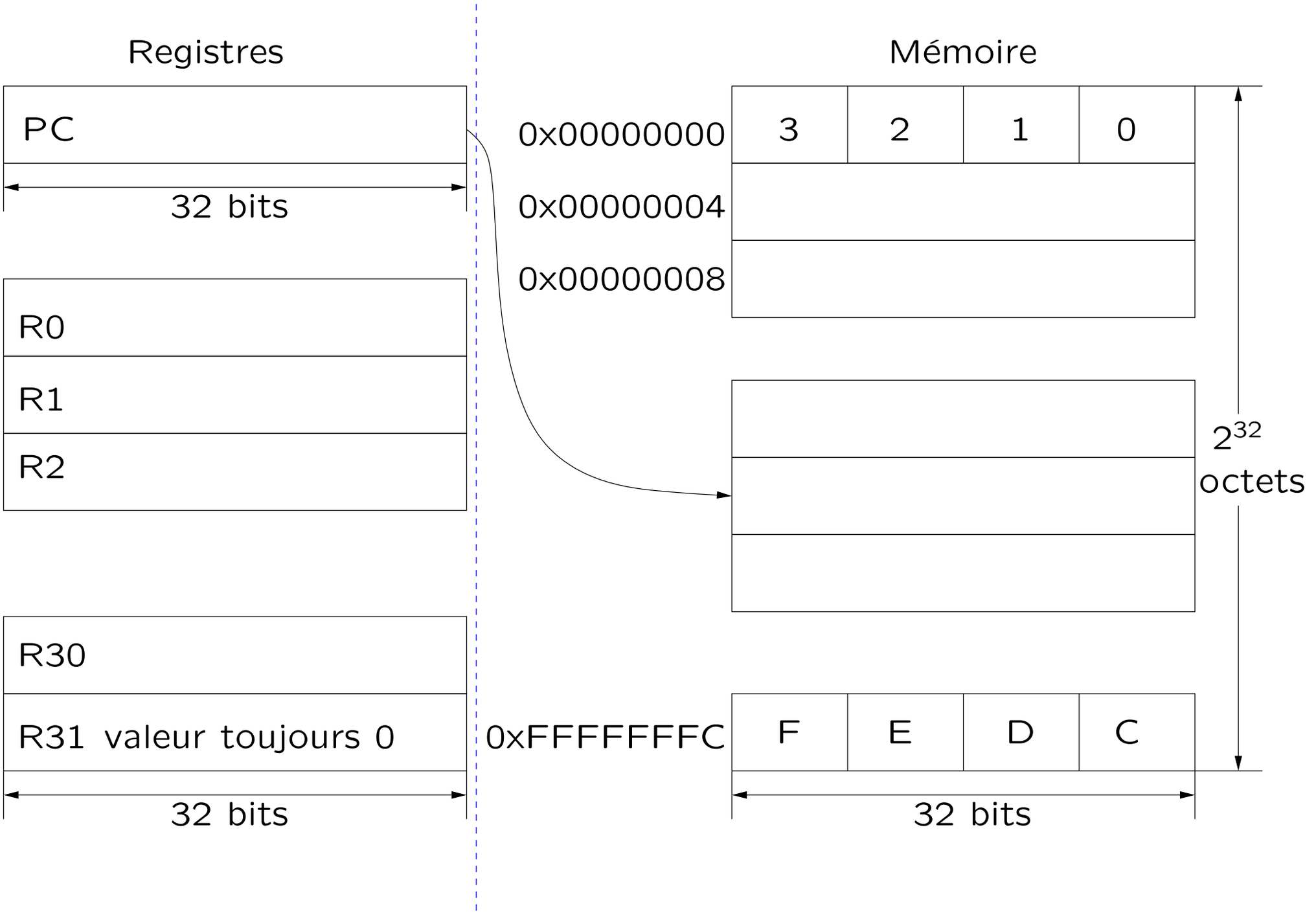
Pour concevoir l'unité de contrôle, il faut d'abord définir la machine qui sera réalisée.

Une architecture de machine : La machine β

- Architecture 32 bits : les registres contiennent tous 32 bits et les adresses mémoire comportent 32 bits, ce qui permet d'adresser 2^{32} octets de mémoire.

- La machine comporte 32 (0 à 31) registres de 32 bits et un compteur de programme de 32 bits dont la valeur est toujours un multiple de 4. Le registre 31 contient toujours la valeur 0.

- Les instructions sont toutes codées sur 32 bits.

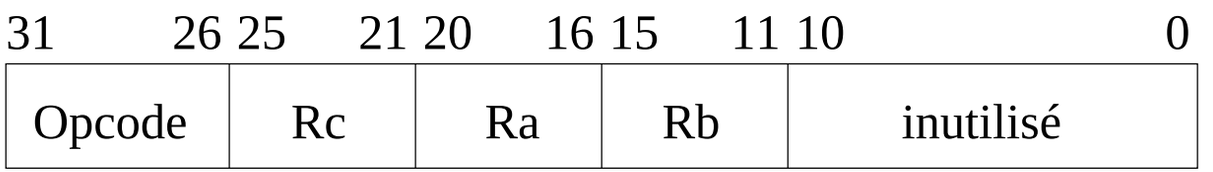

Registres
Mémoire
PC
32 bits
R0
R1
R2
R30
R31 valeur toujours 0
32 bits
0x00000000
0x00000004
0x00000008
0xFFFFFFFC
3
2
1
0
F
E
D
C
2^{32}
octets
32 bits

Machine β : format des instructions

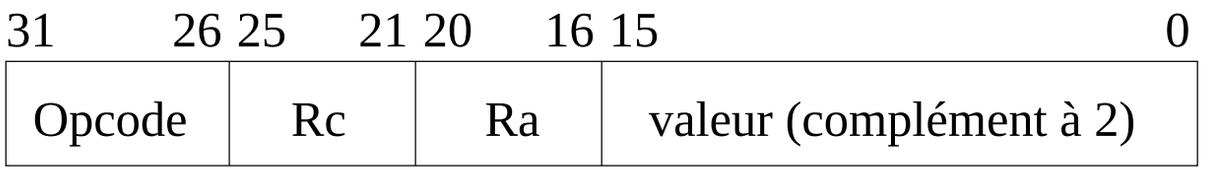
Toutes les opérations de la machine β (sauf les accès à la mémoire) sont effectuées entre registres.

Il y a deux formats possibles pour les instructions :

- Le format sans valeur constante (literal)



- Le format avec valeur constante (literal)



Machine β : instructions arithmétiques

Sans valeur		Avec valeur	
Opcode	nom	Opcode	nom
$0x20$	ADD	$0x30$	ADDC
$0x21$	SUB	$0x31$	SUBC
$0x22$	MUL	$0x32$	MULC
$0x23$	DIV	$0x33$	DIVC

ADD(Ra,Rb,Rc) : $PC \leftarrow PC + 4$
$Reg[Rc] \leftarrow Reg[Ra] + Reg[Rb]$

ADDC(Ra,literal,Rc) : $PC \leftarrow PC + 4$
$Reg[Rc] \leftarrow Reg[Ra] + SEXT(literal)$

SEXT(literal) représente la valeur contenue dans l'instruction convertie de 16 à 32 bits, le signe étant conservé.

La définition de SUB, MUL et DIV est similaire.

Machine β : instructions de comparaison

Sans valeur		Avec valeur	
Opcode	nom	Opcode	nom
$0x24$	COMPEQ	$0x34$	COMPEQC
$0x25$	COMPLT	$0x35$	COMPLTC
$0x26$	COMPLE	$0x36$	COMPLEC

COMPEQ(Ra,Rb,Rc) : PC $\leftarrow$ PC + 4
if Reg[Ra] = Reg[Rb] **then** Reg[Rc] $\leftarrow$ 1
else Reg[Rc] $\leftarrow$ 0

COMPEQC(Ra,literal,Rc) : PC $\leftarrow$ PC + 4
if Reg[Ra] = SEXT(literal) **then** Reg[Rc] $\leftarrow$ 1
else Reg[Rc] $\leftarrow$ 0

CMPLT et CMPLTC comparent selon la relation $<$, CMPLE et CMPLEC selon $\leq$.

Machine β : instructions logiques

Sans valeur		Avec valeur	
Opcode	nom	Opcode	nom
$0x28$	AND	$0x38$	ANDC
$0x29$	OR	$0x39$	ORC
$0x2A$	XOR	$0x3A$	XORC

AND(Ra,Rb,Rc) : PC $\leftarrow$ PC + 4
Reg[Rc] $\leftarrow$ Reg[Ra] & Reg[Rb]

ANDC(Ra,literal,Rc) : PC $\leftarrow$ PC + 4
Reg[Rc] $\leftarrow$ Reg[Ra] & SEXT(literal)

OR et ORC calculent le "ou", XOR et XORC, le "ou" exclusif.

Machine β : instructions de décalage

Sans valeur		Avec valeur	
Opcode	nom	Opcode	nom
$0x2C$	SHL	$0x3C$	SHLC
$0x2D$	SHR	$0x3D$	SHRC
$0x2E$	SRA	$0x3E$	SRAC

SHL(Ra,Rb,Rc) : $PC \leftarrow PC + 4$
$Reg[Rc] \leftarrow Reg[Ra] \ll Reg[Rb]_{4:0}$

SHLC(Ra,literal,Rc) : $PC \leftarrow PC + 4$
$Reg[Rc] \leftarrow Reg[Ra] \ll literal_{4:0}$

Le contenu de Ra est décalé d'un nombre de positions donné par les 5 bits inférieurs de Rb et le résultat est placé dans Rc. Des 0 sont introduits dans les positions libérées.

SHR et SHRC décalent vers la droite (introduction de 0 dans les positions libérées).

SRA et SRAC décalent aussi vers la droite, mais le bit de signe ($Reg[Ra]_{31}$) est introduit dans les positions libérées.

Machine β : instructions d'accès à la mémoire

Opcode	nom
$0x18$	LD
$0x19$	ST

LD(Ra,literal,Rc) : $PC \leftarrow PC + 4$
$EA \leftarrow Reg[Ra] + SEXT(literal)$
$Reg[Rc] \leftarrow Mem[EA]$

ST(Rc,literal,Ra) : $PC \leftarrow PC + 4$
$EA \leftarrow Reg[Ra] + SEXT(literal)$
$Mem[EA] \leftarrow Reg[Rc]$

EA est ce qu'on appelle l'adresse "effective".

Machine β : instructions d'accès à la mémoire II

Opcode	nom
$0x1F$	LDR

LDR(label,Rc) : $PC \leftarrow PC + 4$
$EA \leftarrow PC + 4 \times SEXT(literal)$
$Reg[Rc] \leftarrow Mem[EA]$

Dans LDR, l'instruction fournie à l'assembleur contient un étiquette (label). Dans l'instruction assemblée, la valeur introduite (literal) est calculée par

$$literal = ((OFFSET(label) - OFFSET(inst.\ courante)) \div 4) - 1$$

Machine β : instructions de branchement

Opcode	nom
$0x1B$	JMP

JMP(Ra,Rc) : PC $\leftarrow$ PC + 4
EA $\leftarrow$ Reg[Ra] & $0x$FFFFFFFC
Reg[Rc] $\leftarrow$ PC
PC $\leftarrow$ EA

Les 2 bits inférieurs de Ra sont mis à 0 pour être sûr d'avoir une adresse de mot ; l'adresse de l'instruction suivant le JMP est sauvée dans Rc.

Machine β : instructions de branchement II

Opcode	nom
$0x1D$	BEQ/BF
$0x1E$	BNE/BT

BEQ(Ra,label,Rc) : $PC \leftarrow PC + 4$
$EA \leftarrow PC + 4 \times SEXT(literal)$
$TMP \leftarrow Reg[Ra]$
$Reg[Rc] \leftarrow PC$
if $TMP = 0$ **then** $PC \leftarrow EA$

Dans BEQ, l'instruction fournie à l'assembleur contient un étiquette (label). Dans l'instruction assemblée, la valeur introduite (literal) est calculée par

$$literal = ((OFFSET(label) - OFFSET(inst.\ courante)) \div 4) - 1$$

BNE est similaire à BEQ si ce n'est que l'on teste la nonégalité à 0.

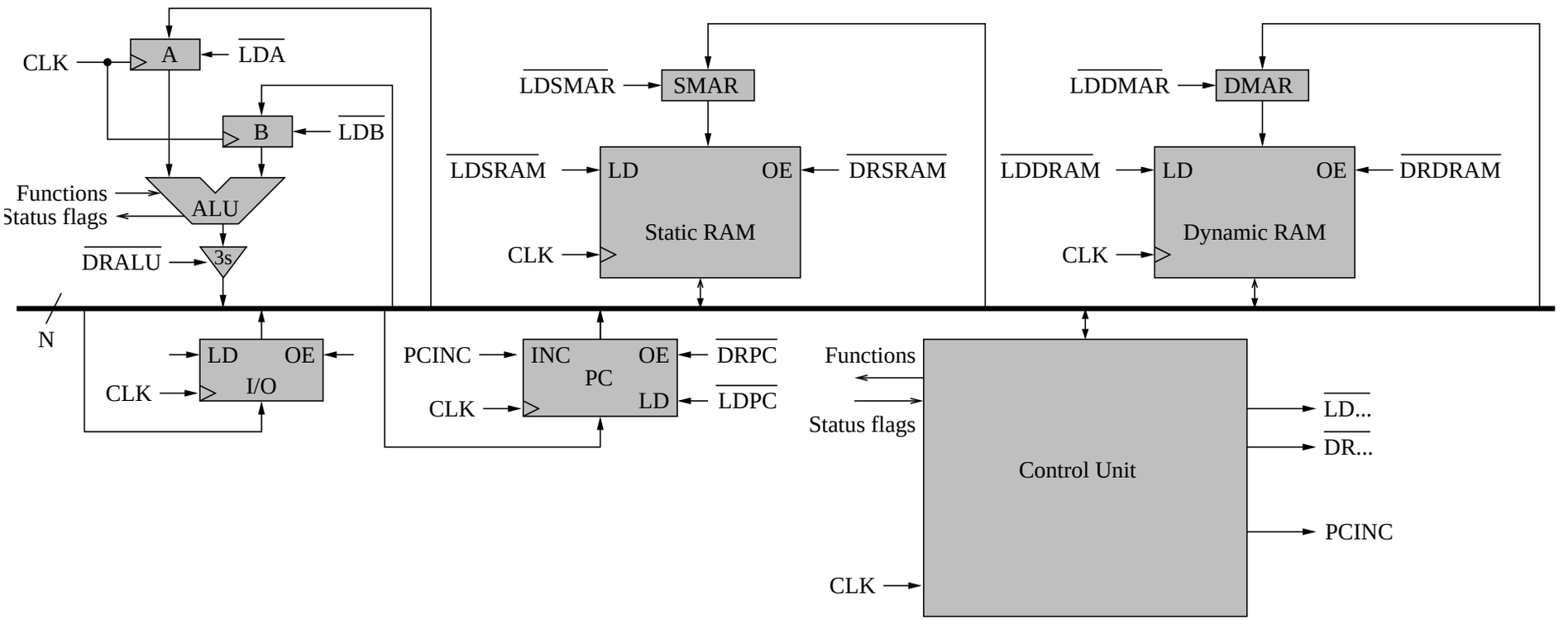
TMP est utilisé car Ra et Rc pourraient être identiques.

Un réalisation de la machine β : la machine ULg01

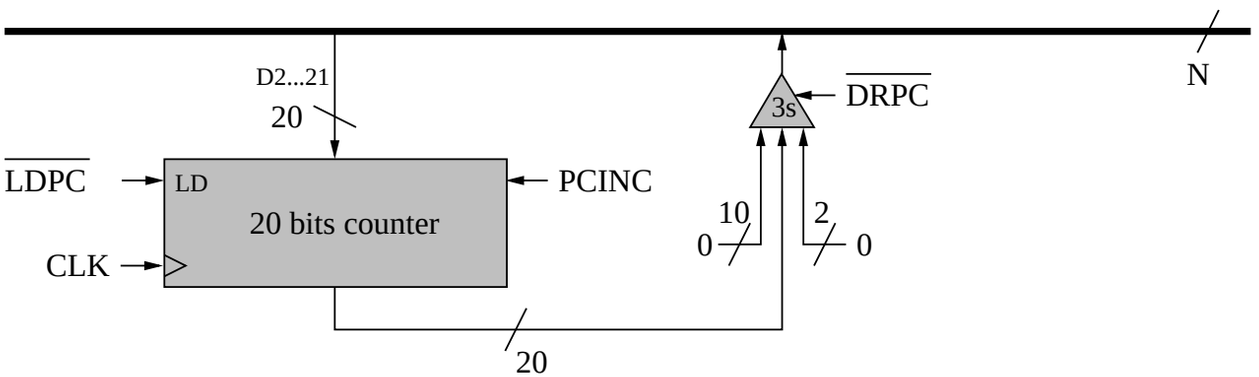
Nous partons du chemin de données décrit précédemment.

- La RAM statique servira à implémenter les registres.
- La RAM dynamique servira à implémenter la mémoire principale.
- Il faut introduire un compteur de programme (PC) et une unité de contrôle.

Un premier schéma de ULg01

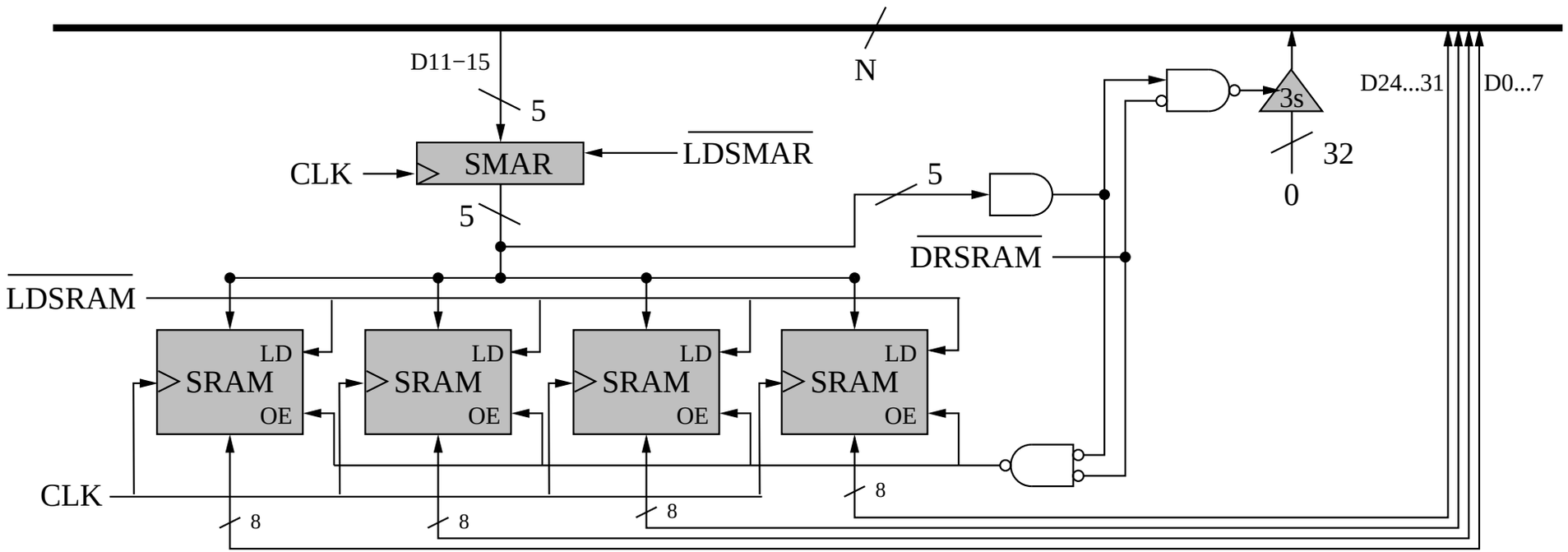


Le PC de ULg01 vu en détails



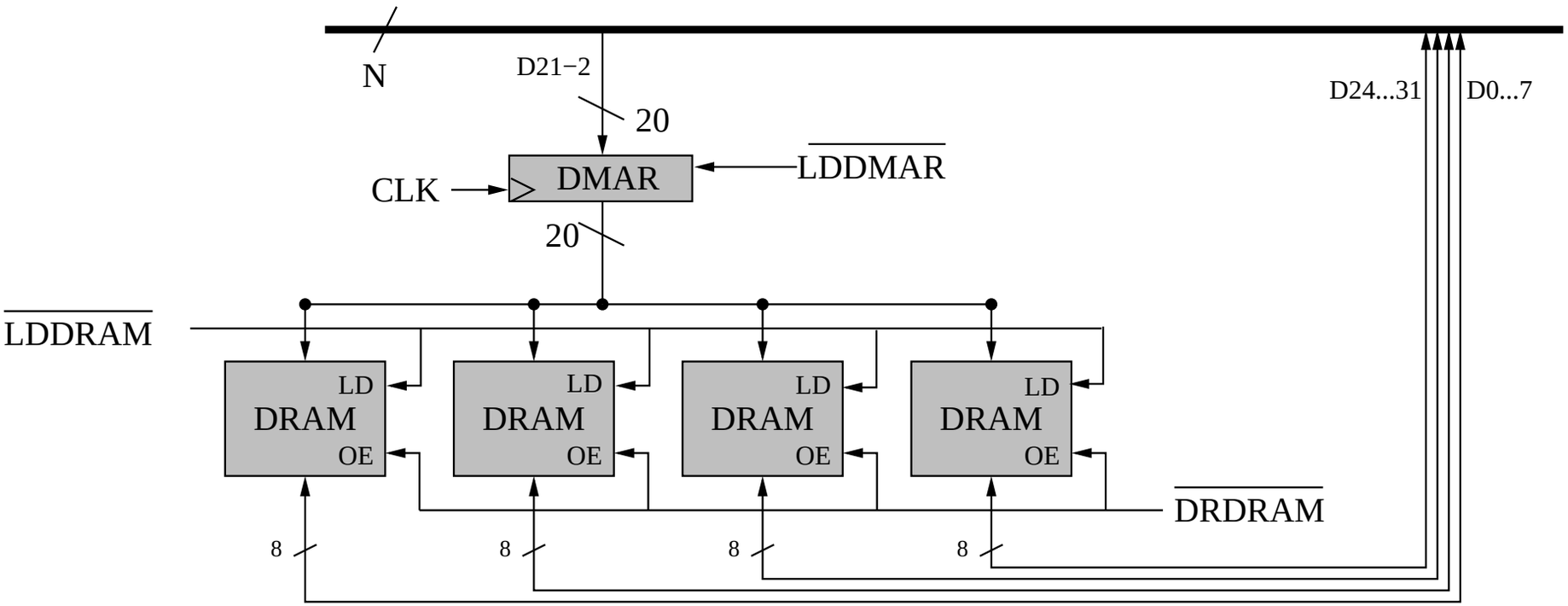
On se limite à 20 bits car la machine construite n'aura pas plus de 4 mega-octets (1 mega-mot) de mémoire.

La SRAM de ULg01 vue en détails



Ce circuit est câblé pour générer la valeur 0 lorsque le registre 31 est lu.

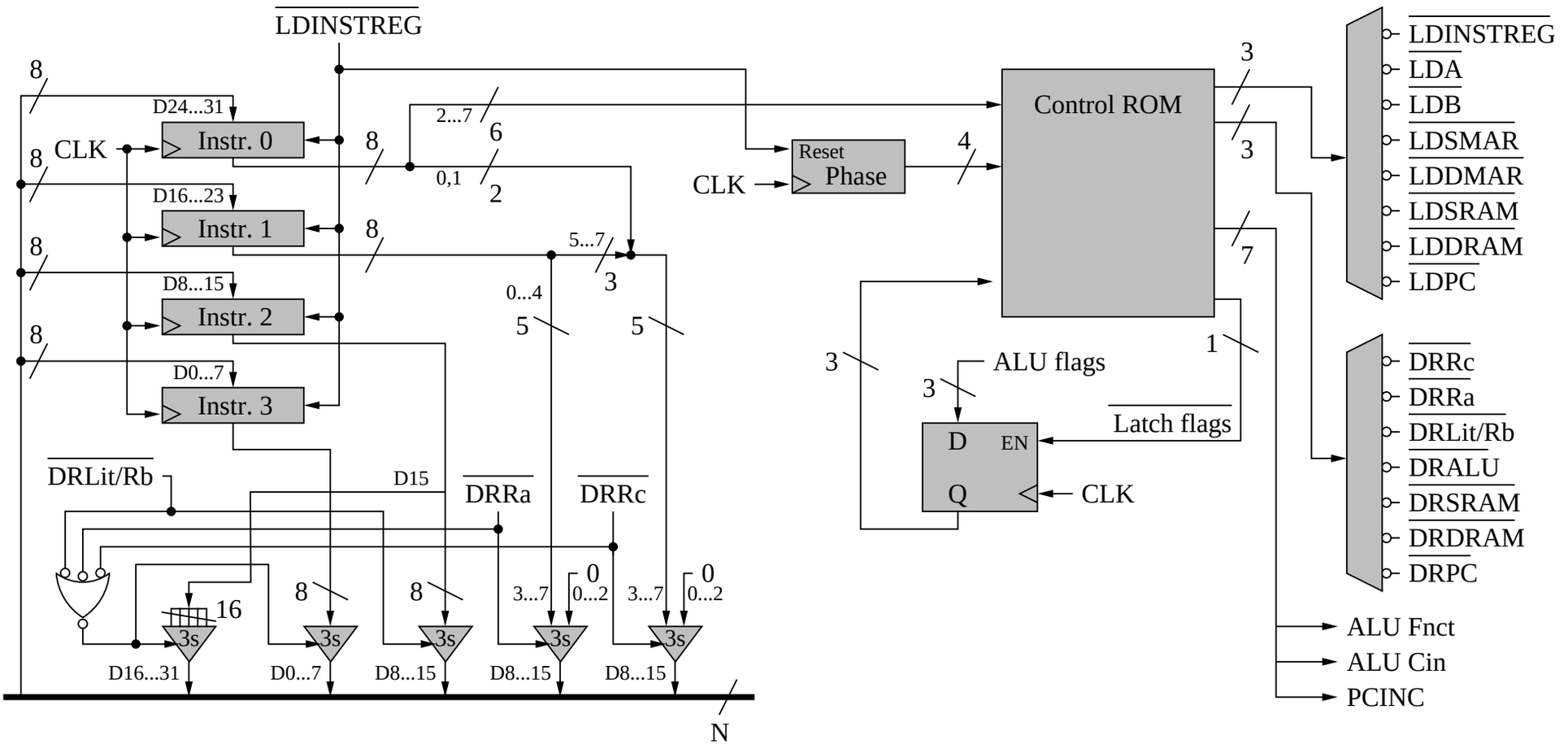
La DRAM de ULg01 vue en détails



Le module de DRAM doit contenir un circuit de rafraîchissement.

La fréquence de l'horloge est choisie pour que la DRAM travaille au même rythme que le reste. Dans une réalisation optimisée, la DRAM travaille plus lentement.

L'unité de Contrôle de ULg01 vue en détails



L'unité de contrôle est une unité microprogrammée qui permet jusqu'à 16 phases par instruction. Le microprogramme est contenu dans une ROM.

Les “Flags” de l’ALU sont utilisés en entrée de la ROM pour permettre l’implémentation d’instructions conditionnelles. Ils sont

- `E` qui vaut `1` si la sortie de l’ALU est égale à `0xFFFFFFFF`,
- $\overline{\texttt{C}}$ le bit de report complémenté,
- `N` le bit de signe (bit 31).

La partie gauche de l’unité de contrôle sert à séparer les différentes partie d’une instruction et à les amener à l’endroit voulu sur le bus.

Pour implémenter les instructions de la machine β, il ne faut plus que définir le microcode qui doit être placé dans la ROM.

Le microcode de ULg01

L'instruction **OR(Ra, Rb, Rc)**

Opcode	Phase	Flags	$\overline{\text{Latch}}$ $\overline{\text{flags}}$	ALU F,$\overline{C_{in}}$,Mode	LD SEL	DR SEL	PC+	
101001	0000	*	1	000000	011	001	0	SMAR ← Ra
101001	0001	*	1	000000	001	100	0	A ← SRAM
101001	0010	*	1	000000	011	010	0	SMAR ← Rb
101001	0011	*	1	000000	010	100	0	B ← SRAM
101001	0100	*	1	000000	011	000	0	SMAR ← Rc
101001	0101	*	1	111001	101	011	0	SRAM ← A | B
101001	0110	*	1	000000	100	110	1	DMAR ← PC; PC+
101001	0111	*	1	000000	000	101	0	INSTREG ← DRAM

L'instruction **JMP(Ra, Rc)**

Opcode	Phase	Flags	$\overline{\text{Latch}}$ $\overline{\text{flags}}$	ALU F,$\overline{C_{in}}$,Mode	LD SEL	DR SEL	PC+	
011011	0000	*	1	000000	011	001	0	SMAR ← Ra
011011	0001	*	1	000000	001	100	0	A ← SRAM
011011	0010	*	1	000000	011	000	0	SMAR ← Rc
011011	0011	*	1	000000	101	110	0	SRAM ← PC
011011	0100	*	1	111111	111	011	0	PC ← A
011011	0101	*	1	000000	100	110	1	DMAR ← PC; PC+
011011	0110	*	1	000000	000	101	0	INSTREG ← DRAM

L'instruction **BEQ(Ra, label, Rc)**

Opcode	Phase	Flags	$\overline{\text{Latch}}$ $\overline{\text{flags}}$	ALU F,$\overline{C_{in}}$,Mode	LD SEL	DR SEL	PC+	
011101	0000	*	1	000000	011	001	0	SMAR ← Ra
011101	0001	*	1	000000	001	100	0	A ← SRAM
011101	0010	*	0	111110	001	011	0	A ← A-1; Latch
011101	0011	*	1	000000	011	000	0	SMAR ← Rc
011101	0100	*	1	000000	101	110	0	SRAM ← PC
011101	0101	E=0	1	000000	100	110	1	DMAR ← PC; PC+
011101	0110	E=0	1	000000	000	101	0	INSTREG ← DRAM
011101	0101	E=1	1	000000	001	010	0	A ← Lit
011101	0110	E=1	1	110010	001	011	0	A ← A+A
011101	0111	E=1	1	110010	001	011	0	A ← A+A
011101	1000	E=1	1	000000	010	110	0	B ← PC
011101	1001	E=1	1	100110	111	011	0	PC ← A+B
011101	1010	E=1	1	000000	100	110	1	DMAR ← PC; PC+
011101	1011	E=1	1	000000	000	101	0	INSTREG ← DRAM

Rappel : le "literal" figurant dans l'instruction interprétée par le microcode est calculé (par l'assembleur) à partir du "label" figurant dans l'instruction présentée sous forme symbolique suivant l'expression

$$\text{literal} = ((\text{OFFSET}(\text{label}) - \text{OFFSET}(\text{inst. courante})) \div 4) - 1$$

L'instruction **LD(Ra, literal, Rc)**

Opcode	Phase	Flags	$\overline{\text{Latch}}$ $\overline{\text{flags}}$	ALU F,$\overline{C_{in}}$,Mode	LD SEL	DR SEL	PC+	
011000	0000	*	1	000000	011	001	0	SMAR ← Ra
011000	0001	*	1	000000	001	100	0	A ← SRAM
011000	0010	*	1	000000	010	010	0	B ← Lit
011000	0011	*	1	000000	011	000	0	SMAR ← Rc
011000	0100	*	1	100110	100	011	0	DMAR ← A+B
011000	0101	*	1	000000	101	101	0	SRAM ← DRAM
011000	0110	*	1	000000	100	110	1	DMAR ← PC; PC+
011000	0111	*	1	000000	000	101	0	INSTREG ← DRAM