Computation structures

Pierre Wolper

Email: Pierre.Wolper@ulg.ac.be

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

References

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, Pearson Education, 2006.

Course objectives

1. **Computer organization and machine language.**

   (a) To study in detail how a simple programmable machine can be built.

   (b) To introduce processor performance enhancement techniques.

   (c) To overview the concepts used in order to make the task of programming a computer more manageable: interrupts and supervisor (privileged) mode, virtual memory, . . .

2. **Operating systems and Concurrent programming**

(a) To introduce programming with processes and the organization of an operating system kernel.

(b) To develop the concepts pertaining to programming with processes and interprocess communication.

(c) To study classical parallel programming problems.

(d) To give a brief introduction to questions concerning parallel machines and how they are programmed.

Exercises and programming assignments

- Exercises on processor design and microcode.

- Exercises and a programming assignment in assembly language on low level programming and operating system kernels.

- Exercises and a programming assignment on using parallel processes and interprocess communication (using C as programming language).

Digital electronic circuits : review

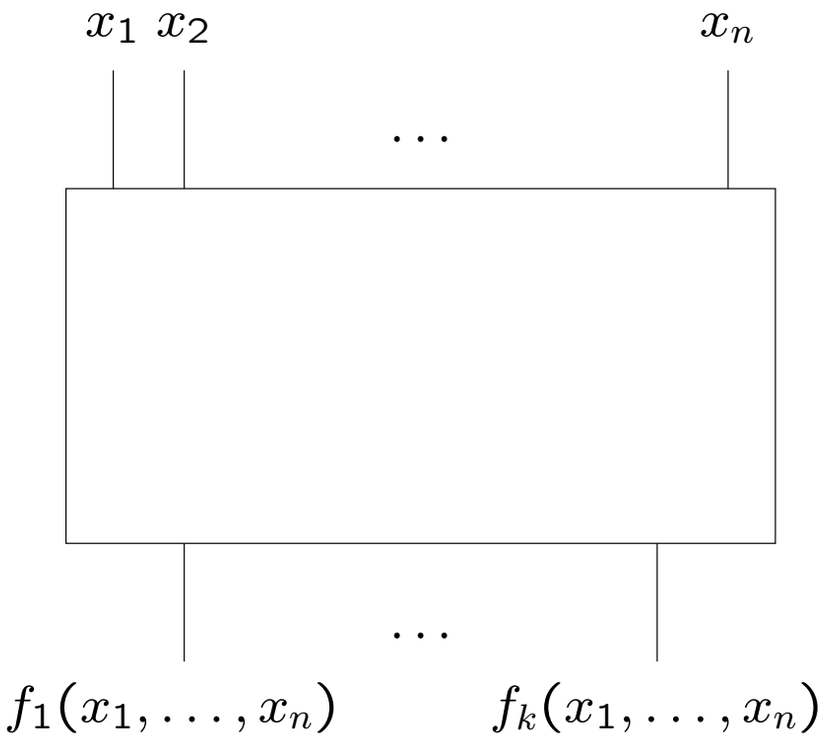
We will use basic facts about digital circuits.

- In digital circuits, an electrical signal is used to represent a Boolean (binary) piece of information $(0, 1)$, for example $[0, 1]$ volts represents 0 and $[2, 3]$ volts represents 1.

- Boolean circuits can perform operations on the represented Boolean values.

- We will use three types of circuits: combinatorial circuits, memory elements and synchronous sequential circuits.

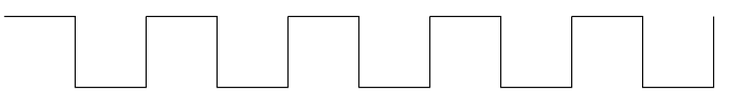
Digital circuits
combinatorial circuits



- Each f_i is a Boolean function $(\wedge, \vee, \neg)$ of $x_1, \ldots, x_n$. *If the inputs (x_i) change, there is a certain delay defore the outputs (f_i) stabilize at the correct value.*

Digital circuits
memory elements

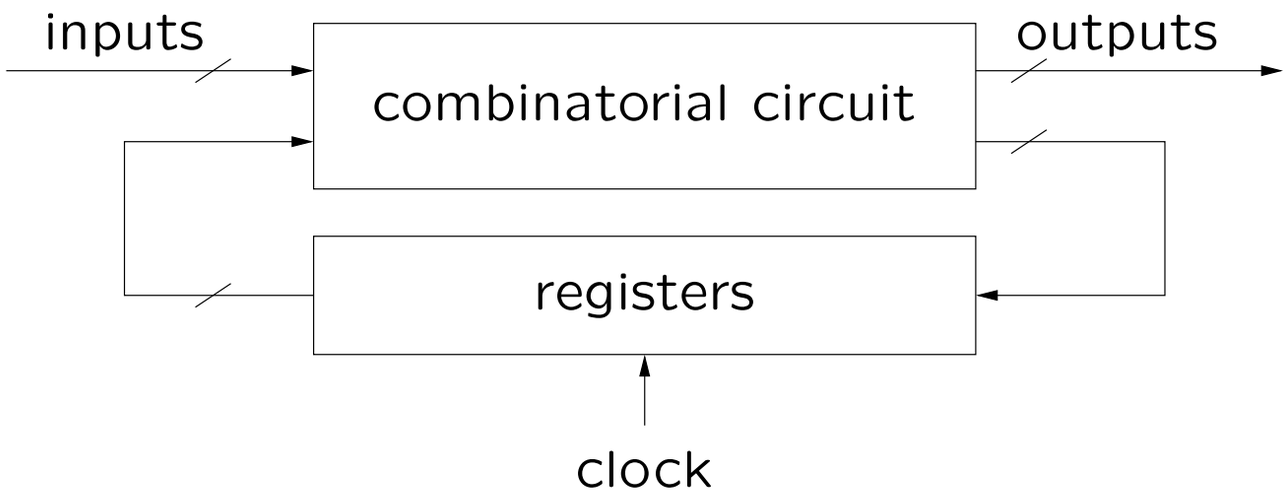
- CLK is a clock signal :



- When LD is active (value 1), the values of $x_1, \ldots, x_n$ are stored at the address given by $a_1, \ldots, a_k$ at the next ($0 \rightarrow 1$) clock transition. The values of the outputs $y_1, \ldots, y_n$ are equal to the values stored at the address $a_1, \ldots, a_k$.

- A register is a one location memory (no addess)

Digital circuits
Synchronous sequential circuits



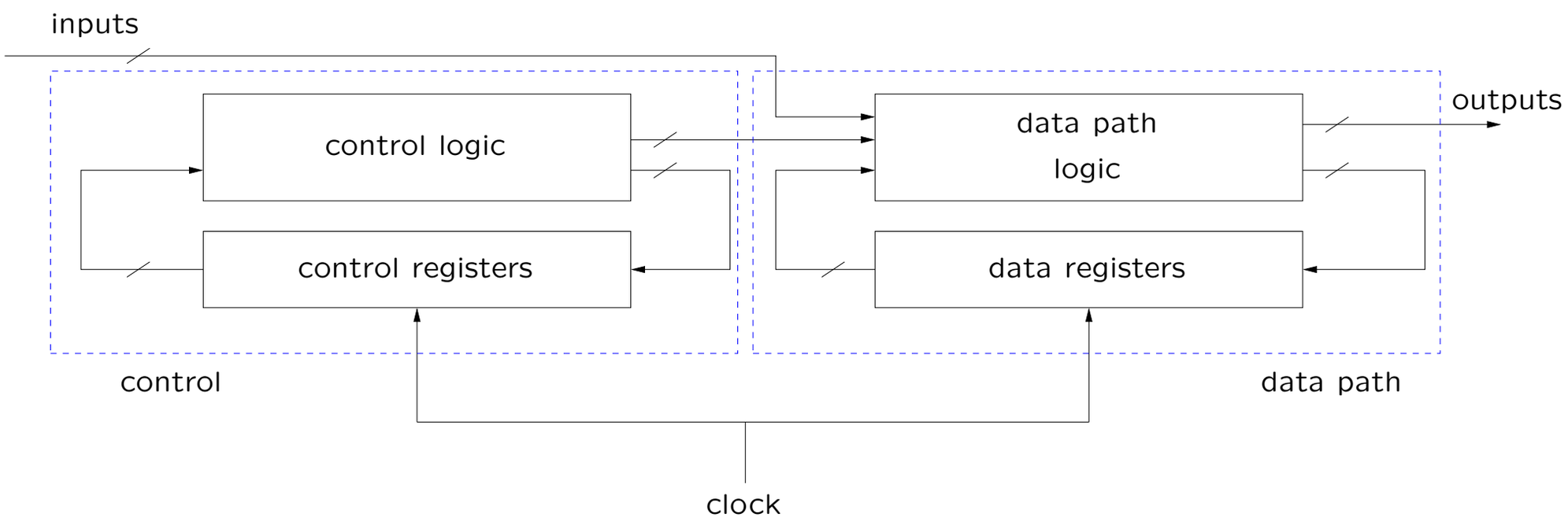
- At each clock transition, a Boolean function of the inputs and of the previous register contents is loaded in the registers.

- The outputs are a Boolean function of the inputs and of the register contents.

From synchronous circuits to processors

A processor is a synchronous sequential circuit with a few peculiarities.

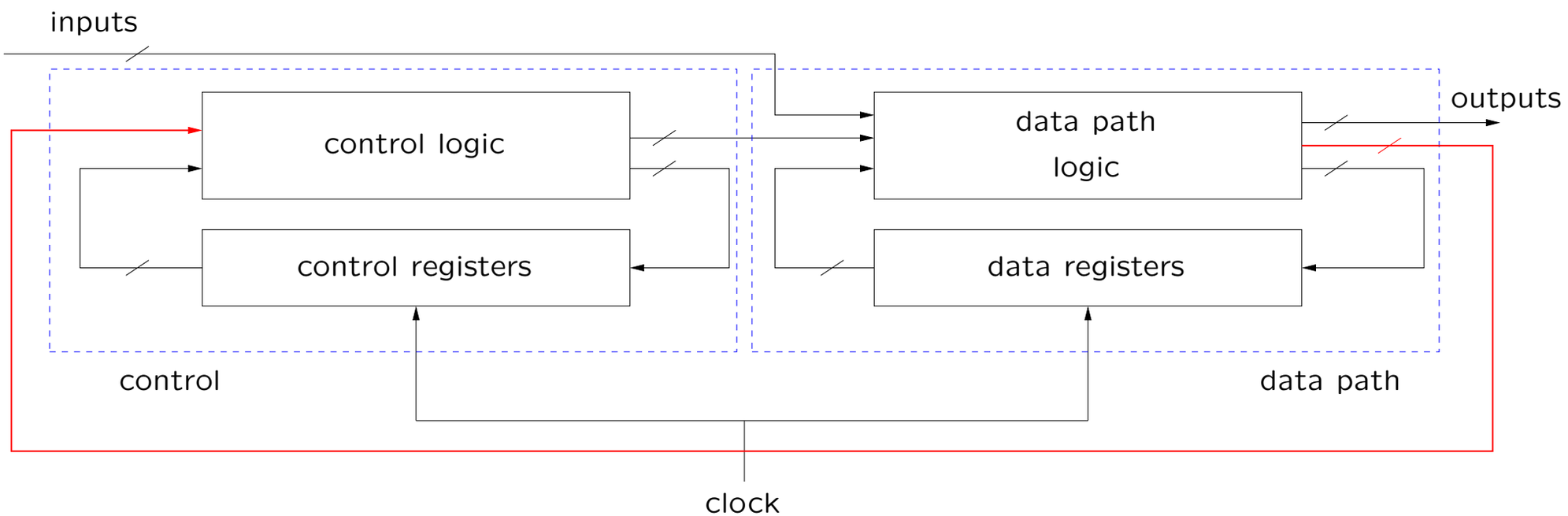
1. The circuit is divided into a "data" and a "control" part. The "data" part performs operations on memorized data; the "control" determines which operations are performed.



2. A large capacity memory is included in the "data" part (the machine's main memory).

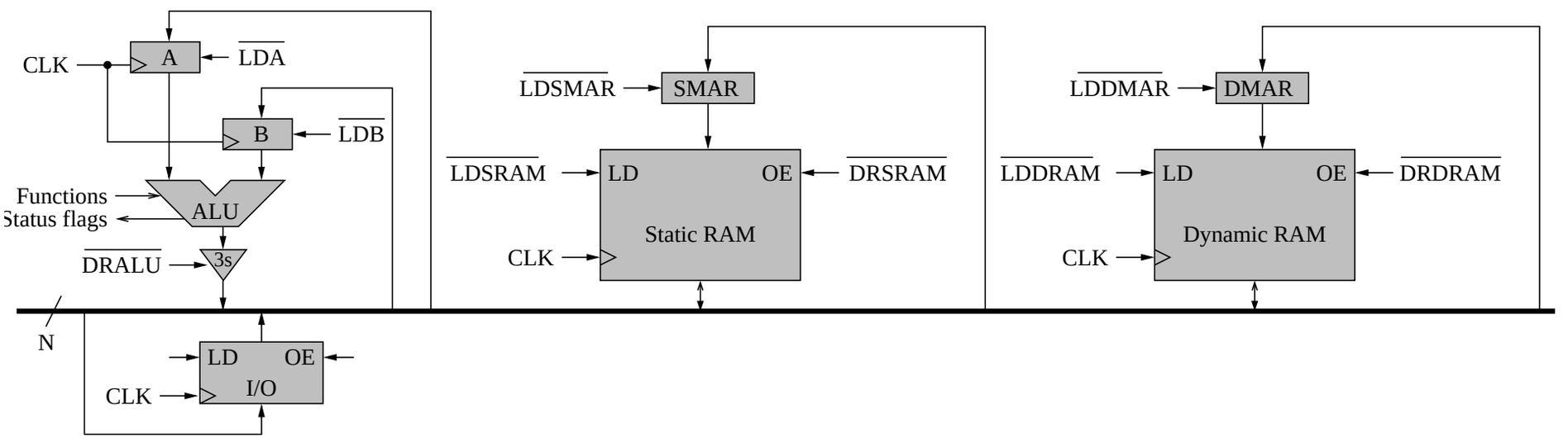
3. The behavior of the control part is influenced by the data part:

   - control influenced -by the results of operations on the data;
   - control determined by a program stored in the data path memory.



A simple data path

In the date path shown below, one finds memory elements and a computation element (a combinatorial circuit) interconnected by a *bus*.



A bus is a collection of shared interconnection lines.

- Each element with an output connected to the bus can be active (0 or 1) or inactive (disconnected – this is called 3 state logic).

- The bus presented is a simple bus with a centralized control managed by the control unit.

The signals needed for the data path to operate will be generated by the control unit.

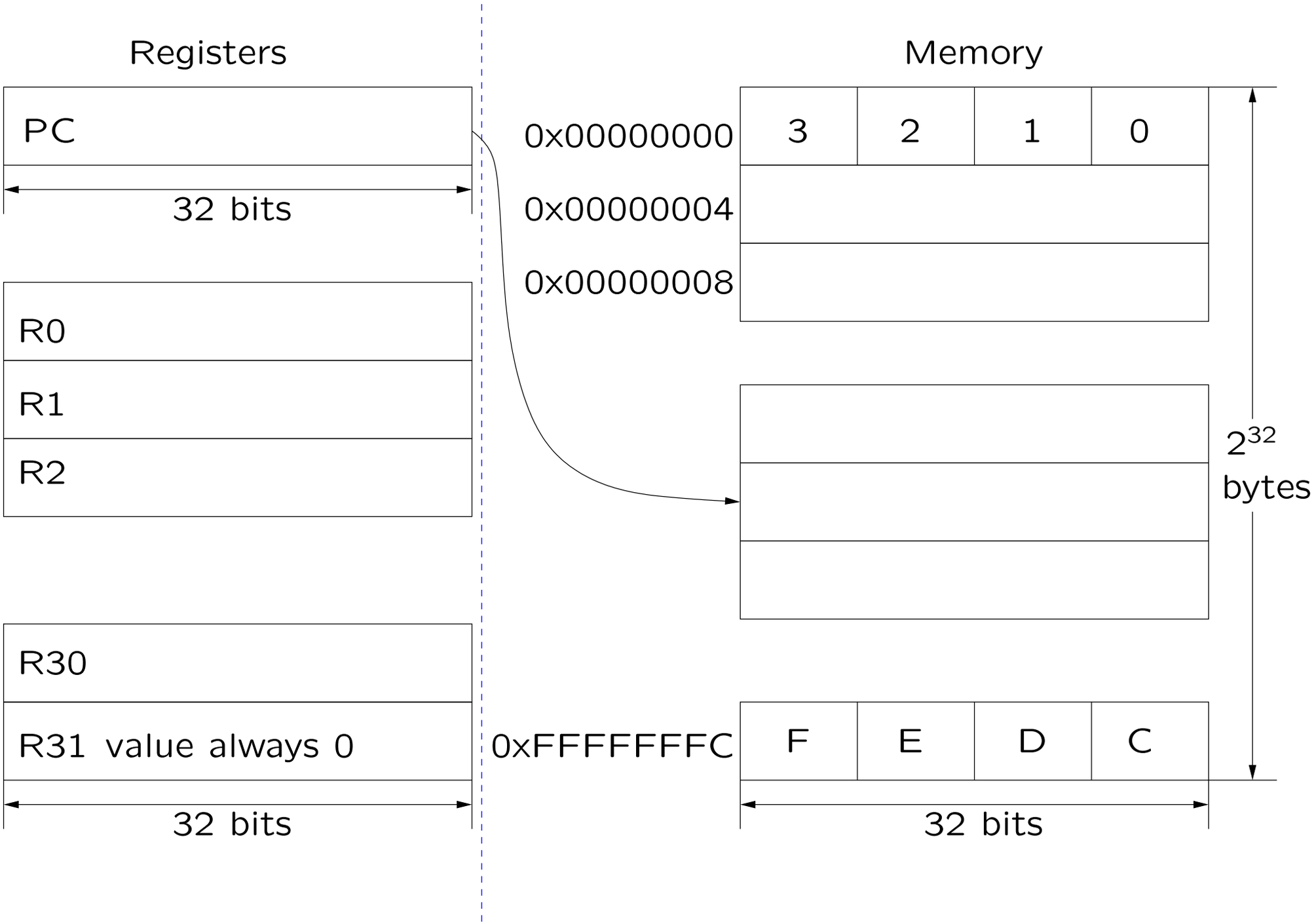
To design the control unit one must first define the machine that is to be built.

A machine architecture: The β machine

- A 32 bit architecture: the registers contain 32 bits and the memory addresses are 32 bits long, which makes it possible to address 2^{32} bytes of memory.

- The machine has 32 (0 à 31) 32-bit registers and a 32-bit program counter whose value is always a multiple of 4. Register 31 always contains the value 0.

- All instructions are 32-bit long.

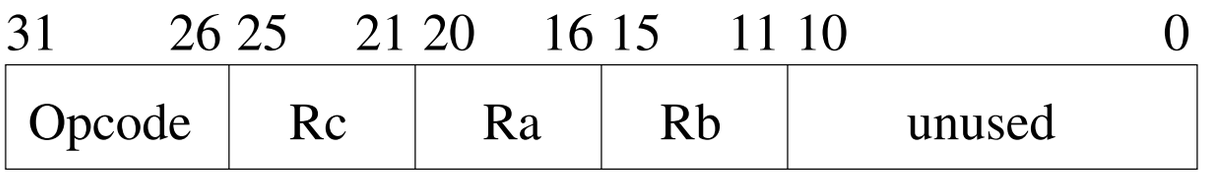

Registers
Memory
PC
32 bits
R0
R1
R2
R30
R31 value always 0
32 bits
0x00000000
0x00000004
0x00000008
0xFFFFFFFC
3
2
1
0
F
E
D
C
2^{32}
bytes
32 bits

β Machine: instruction formats

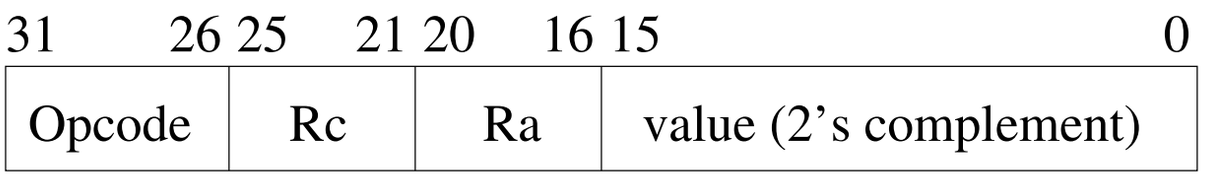
All operations of the β Machine (except those accessing the main memory) are performed on registers.

There are two possible instruction formats :

- The format without a literal (*valeur constante*)



- The format with a literal



β Machine : arithmetic instructions

Without a literal		With a literal	
Opcode	name	Opcode	name
$0x20$	ADD	$0x30$	ADDC
$0x21$	SUB	$0x31$	SUBC
$0x22$	MUL	$0x32$	MULC
$0x23$	DIV	$0x33$	DIVC

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

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

SEXT(literal) represents the value contained in the instruction extended from 16 to 32 bits, the sign being preserved.

The definitions of SUB, MUL and DIV are similar.

β Machine : comparison instructions

Without a literal		With a literal	
Opcode	name	Opcode	name
$0x24$	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 and CMPLTC compare according to the $<$ relation, CMPLE and CMPLEC according to $\leq$.

β Machine : logical instructions

Without a literal		With a literal	
Opcode	name	Opcode	name
$0x28$	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 and ORC compute "or", XOR and XORC exclusive "or".

β Machine : shift instructions

Without a literal		With a literal	
Opcode	name	Opcode	name
$0x2C$	SHL	$0x3C$	SHLC
$0x2D$	SHR	$0x3D$	SHRC
$0x2E$	SRA	$0x3E$	SRAC

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

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

The content of Ra is shifted by a number of positions given by the 5 least significant bits of Rb and the result is placed in Rc. 0 bits are introduced in the freed positions.

SHR and SHRC shift to the right (0 bits are introduced in the freed positions).

SRA and SRAC also shift to the right, but the sign bit ($Reg[Ra]_{31}$) is introduced in the freed positions.

β Machine : memory access instructions

Opcode	name
$0x18$	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 is called the "effective address".

β Machine : memory access instructions II

Opcode	name
$0x1F$	LDR

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

In LDR, the instruction given to the assembler contains a label. (label). In the assembled instruction, the literal is computed from the label as follows

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

β Machine : branch instructions

Opcode	name
$0x1B$	JMP

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

The 2 least significant bits of Ra are set to 0 to force the address to be a word address. The address of the instruction following the JMP is saved in Rc.

β Machine : branch instructions II

Opcode	name
$0x1D$	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

In BEQ, the instruction given to the assembler contains a label. In the assembled instruction, the literal is computed as follows

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

BNE is similat to BEQ except that the test is "different from 0"

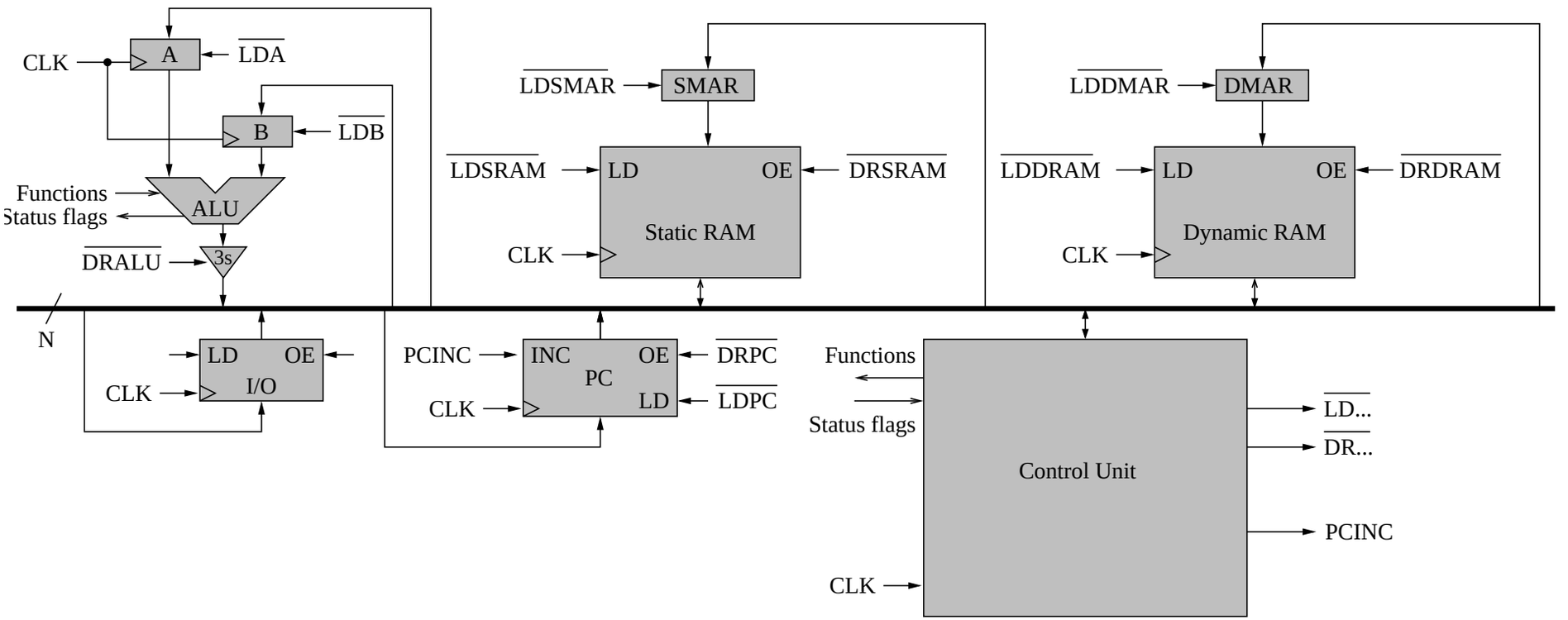
TMP is used because Ra and Rc could be the same register.

An implementation of the β Machine : The machine ULg01

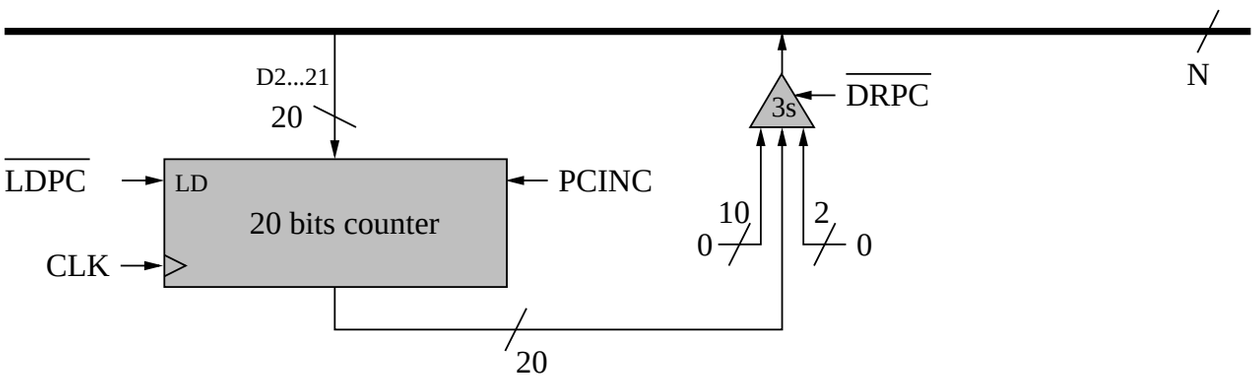
We start from the data path described previously.

- The static RAM will be used to implement the registers.
- The dynamic RAM will be used to implement the main memory.
- On still has to introduce a program counter (PC) and a control unit.

A first sketch of ULg01

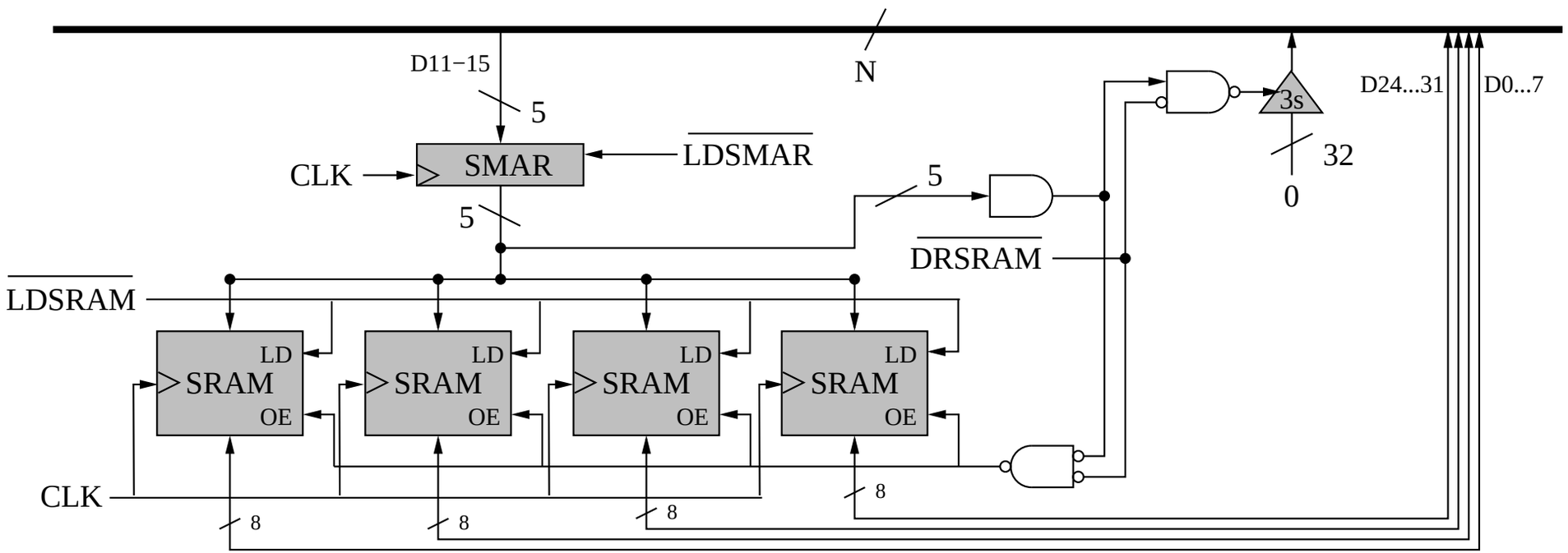


A detailed view of the PC of ULg01



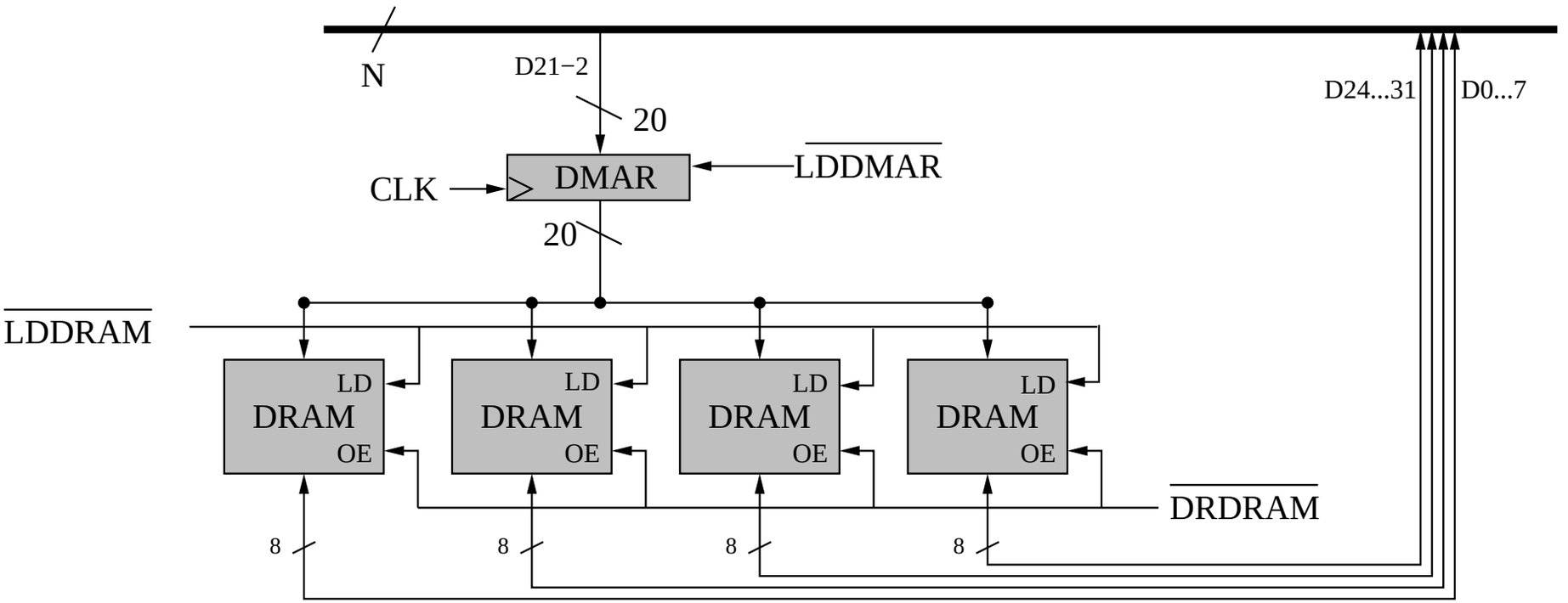
Only 20 bits are used because the machine being built will not have more than 4 megabytes (1 megaword) of memory.

A detailed view of the SRAM of ULg01



The circuit is designed to produce the value 0 when register 31 is read.

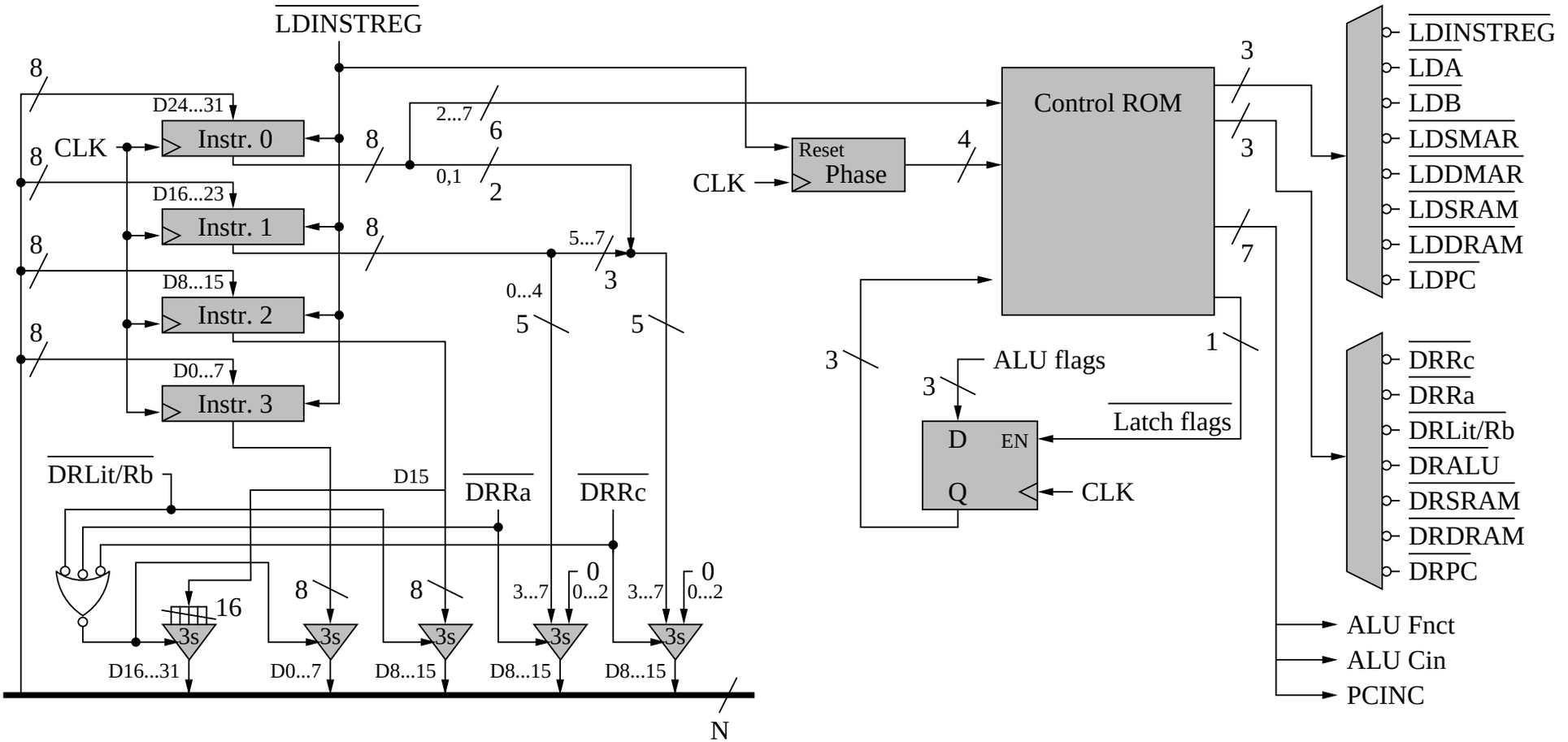
A detailed view of the DRAM of ULg01



The DRAM module must contain refresh circuitry.

The clock speed is chosen so that the DRAM can operate at the same speed as the rest if the machine. In an optimized implementation, the DRAM operates at a lower speed.

A detailed view of the control unit of ULg01



The control unit is microprogrammed and allows up to 16 phases by instruction. The microprogram is stored in a ROM.

The ALU flags are used as address bits into the ROM. This allows the implementation of conditional instructions. The flags are

- `E` whose value is `1` is the output of the ALU is equal to `0xFFFFFFFF`,
- $\overline{\texttt{C}}$ The complemented carry bit,
- `N` the sign bit (bit 31).

The left-hand part of the control unit is used to separate the different parts of an instruction and connect them to the appropriate lines of the bus.

All that needs to be done to finish the implementation of the β Machine, is to write the microcode that has to be placed in the ROM.

The microcode of ULg01

The instruction **OR(Ra, Rb, Rc)**

Opcode	Phase	Flags	$\overline{\text{Latch flags}}$	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

The 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

The 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

Reminder : The literal that is found in the instruction interpreted by the microcode is computed (by the assembler) from the label found in the symbolically written instruction as follows

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

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