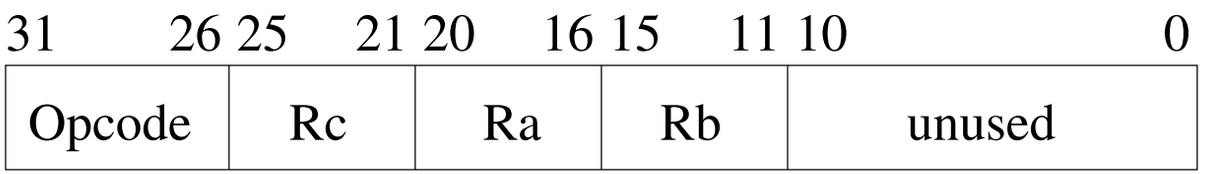
A performance-oriented implementation of the β machine

β Machine: instruction formats (review)

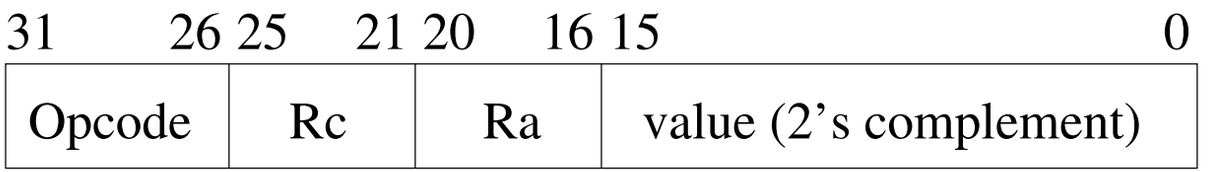
All operations of the β Machine (except those accessing the main memory) are performed on registers.

There are two possible instruction formats :

- The format without a literal



- The format with a literal



β Machine : arithmetic instructions (review)

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 : memory access instructions (review)

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 (review)

Opcode	name
$0x1F$	LDR

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

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 (review)

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 (review)

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

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

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 that does not use micro-code

The goal is to execute instructions within a single clock cycle. For doing this, one proceeds as follows.

- Several data transfer paths are used in order to make simultaneous transfers possible.
- Most elements then have their input connected to a multiplexer, making it possible to select a source among several.
- *Three port* registers are used. These allow two read and one write operation to be executed simultaneously (at the next clock transition).
- The goal of the control logic is then then to select the input of the various elements and to choose the function computed by the ALU.
- The data memory and the instruction memory are viewed as separate (caches).

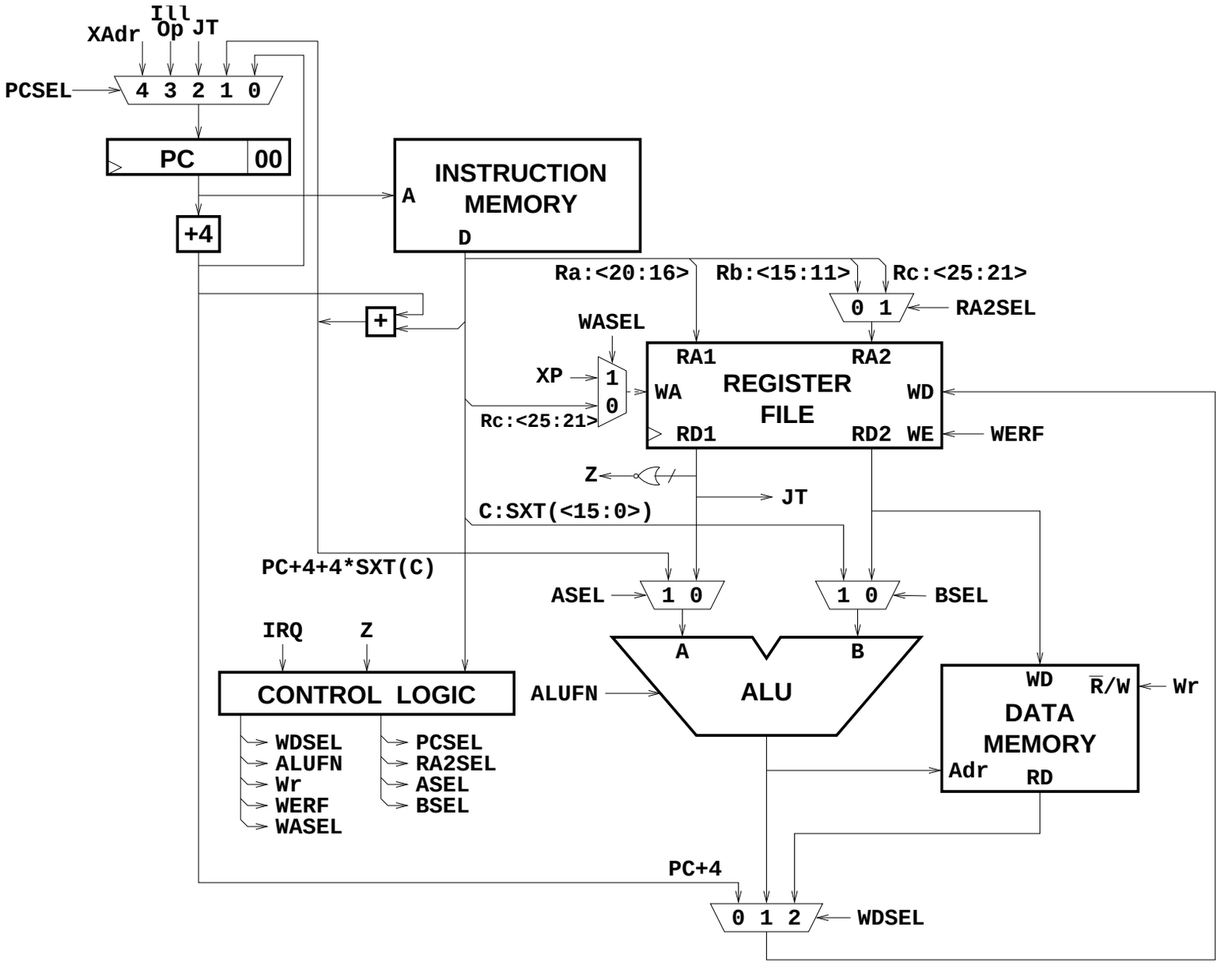
An implementation of the β machine that does not use micro-code (continued)

- The following slide presents the general schema of an implementation of the β machine that does not use micro-code.

- *Exceptions* upon which `PC + 4` is saved in `XP` (register 30) and control jumps to a predefined address are taken into account.

- Supervisor mode is not implemented.

- Virtual memory is not taken into account (one can assume that the caches work with virtual addresses).


Ill
XAdr
Op
JT
PCSEL
4 3 2 1 0
PC
00
+4
INSTRUCTION MEMORY
A
D
Ra:<20:16>
Rb:<15:11>
Rc:<25:21>
0 1
RA2SEL
+
WASEL
XP
1
0
Rc:<25:21>
RA1
RA2
WA
REGISTER FILE
WD
RD1
RD2
WE
WERF
Z
JT
C:SXT(<15:0>)
PC+4+4*SXT(C)
ASEL
1 0
1 0
BSEL
IRQ
Z
A
B
ALUFN
ALU
CONTROL LOGIC
WD
R/W
Wr
DATA MEMORY
Adr
RD
WDSEL
ALUFN
Wr
WERF
WASEL
PCSEL
RA2SEL
ASEL
BSEL
PC+4
0 1 2
WDSEL

The β machine without micro-code : the signals

`Xadr`	address of the interrupt handler.
`ILLOP`	address of the illegal operation handler
`JT`	target of a `JMP`instruction (jump target)
`PCSEL`	PC source selection
`RA2SEL`	selection of the second register read address
`WASEL`	selection of the register write address
`XP`	exception register (30)
`WERF`	Write Enable Register File
`Z`	0 test of the first register output
`A(B)SEL`	ALU A (B) input selection
`WDSEL`	register write data selection
`ALUFN`	ALU function selection
`Wr`	"Write" mode for the data memory

The β machine without micro-code : the control logic

This logic can be implemented with a ROM or a PLA.

	OP	OPC	LD	ST	JMP	BEQ	BNE	LDR	ILLOP	trap
ALUFN	F(op)	F(op)	"+"	"+"	-	-	-	"A"	-	-
WERF	1	1	1	0	1	1	1	1	1	1
BSEL	0	1	1	1	-	-	-	-	-	-
WDSEL	1	1	2	-	0	0	0	2	0	0
Wr	0	0	0	1	0	0	0	0	0	0
RA2SEL	0	-	-	1	-	-	-	-	-	-
PCSEL	0	0	0	0	2	Z==1	Z==0	0	3	4
ASEL	0	0	0	0	-	-	-	1	-	-
WASEL	0	0	0	-	0	0	0	0	1	1

The β machine without micro-code : performance?

- The implementation we have described executes an instruction per clock cycle, but the cycles have to be quite long.

- Indeed, a cycle has to be long enough for the successive stabilization of the instruction memory, the registers, the ALU, that data memory.

- The solution is to organize the implementation as a *pipeline*