Summary of β Instruction Formats

Operate Class:

Register	Symbol	Usage
R31	R31	Always zero
R30	XP	Exception pointer
R29	SP	Stack pointer
R28	LP	Linkage pointer
R27	BP	Base of frame pointer

31 … 26	25 … 21	20 … 16	15 … 11	10 … 0
10xxxx	Rc	Ra	Rb	*unused*

OP(Ra,Rb,Rc): Reg[Rc] ← Reg[Ra] op Reg[Rb]

Opcodes: **ADD** (plus), **SUB** (minus), **MUL** (multiply), **DIV** (divided by)
AND (bitwise and), **OR** (bitwise or), **XOR** (bitwise exclusive or)
CMPEQ (equal), **CMPLT** (less than), **CMPLE** (less than or equal) [result = 1 if true, 0 if false]
SHL (left shift), **SHR** (right shift w/o sign extension), **SRA** (right shift w/ sign extension)

31 … 26	25 … 21	20 … 16	15 … 0
11xxxx	Rc	Ra	literal (two's complement)

OPC(Ra,literal,Rc): Reg[Rc] ← Reg[Ra] op SEXT(literal)

Opcodes: **ADDC** (plus), **SUBC** (minus), **MULC** (multiply), **DIVC** (divided by)
ANDC (bitwise and), **ORC** (bitwise or), **XORC** (bitwise exclusive or)
CMPEQC (equal), **CMPLTC** (less than), **CMPLEC** (less than or equal) [result = 1 if true, 0 if false]
SHLC (left shift), **SHRC** (right shift w/o sign extension), **SRAC** (right shift w/ sign extension)

Other:

31 … 26	25 … 21	20 … 16	15 … 0
01xxxx	Rc	Ra	literal (two's complement)

LD(Ra,literal,Rc):	Reg[Rc] ← Mem[Reg[Ra] + SEXT(literal)]
ST(Rc,literal,Ra):	Mem[Reg[Ra] + SEXT(literal)] ← Reg[Rc]
JMP(Ra,Rc):	Reg[Rc] ← PC + 4; PC ← Reg[Ra]
BEQ/**BF**(Ra,label,Rc):	Reg[Rc] ← PC + 4; if Reg[Ra] = 0 then PC ← PC + 4 + 4*SEXT(literal)
BNE/**BT**(Ra,label,Rc):	Reg[Rc] ← PC + 4; if Reg[Ra] ≠ 0 then PC ← PC + 4 + 4*SEXT(literal)
LDR(label,Rc):	Reg[Rc] ← Mem[PC + 4 + 4*SEXT(literal)]

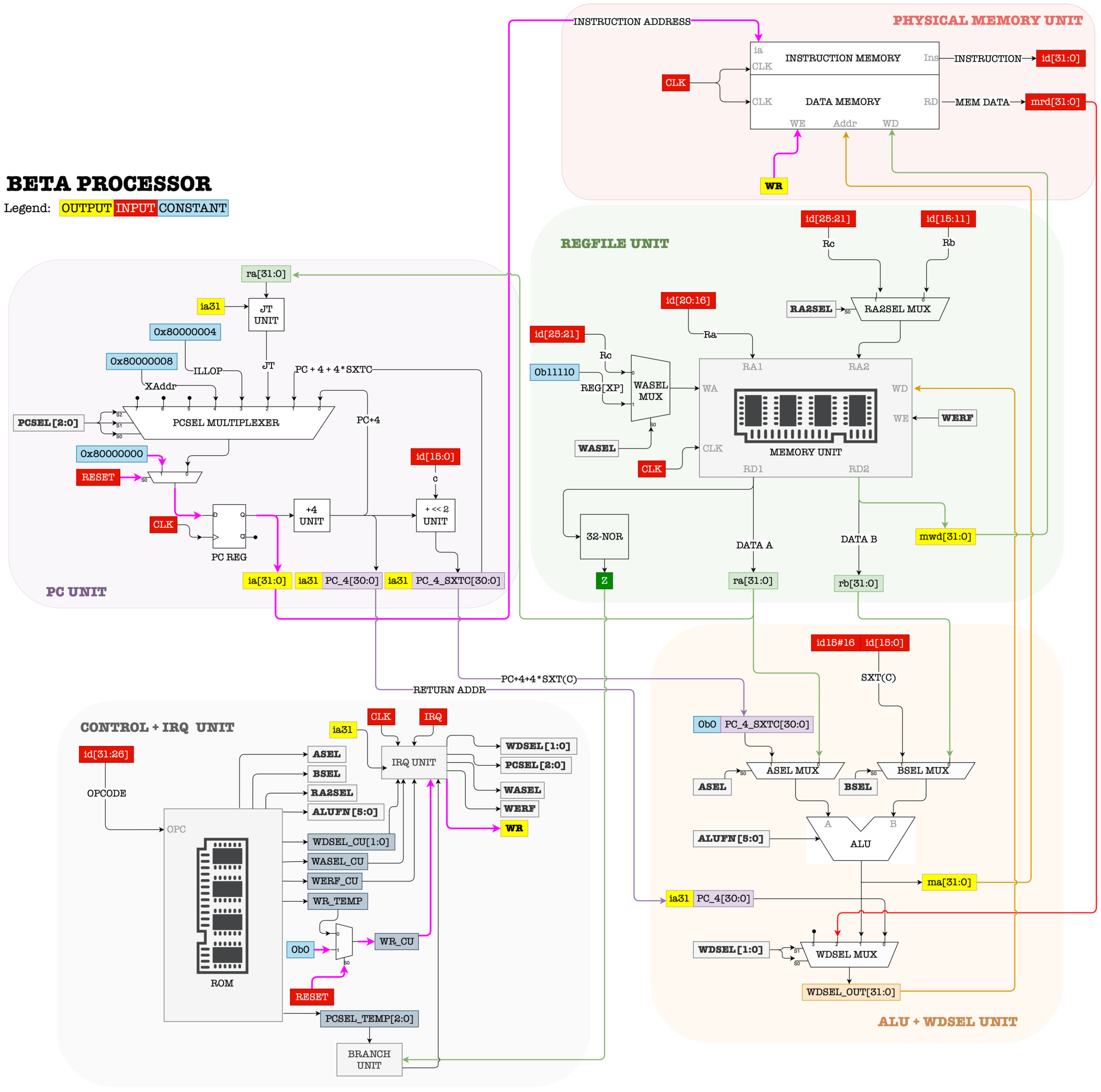
Opcode Table: (*optional opcodes)

2:0 / *5:3*	*000*	*001*	*010*	*011*	*100*	*101*	*110*	*111*
000								
001								
010								
011	**LD**	**ST**		**JMP**		**BEQ**	**BNE**	**LDR**
100	**ADD**	**SUB**	**MUL***	**DIV***	**CMPEQ**	**CMPLT**	**CMPLE**	
101	**AND**	**OR**	**XOR**		**SHL**	**SHR**	**SRA**	
110	**ADDC**	**SUBC**	**MULC***	**DIVC***	**CMPEQC**	**CMPLTC**	**CMPLEC**	
111	**ANDC**	**ORC**	**XORC**		**SHLC**	**SHRC**	**SRAC**	

	OP	OPC	LD	ST	JMP	BEQ	BNE	LDR	Illop	IRQ
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 : 0	Z ? 0 : 1	0	3	4
ASEL	0	0	0	0	--	--	--	1	--	--
WASEL	0	0	0	--	0	0	0	0	1	1

BETA PROCESSOR

Legend: OUTPUT INPUT CONSTANT



6.5 Privileged Instructions

Some instructions may be available while in supervisor mode which are not available in user mode (e.g., instructions which interface directly with I/O devices). These are called "privileged instructions". These instructions always have an opcode of 0x00; otherwise, their form and semantics are implementation-defined. Attempts to use privileged instructions while in user mode will result in an illegal instruction exception.

7. Software Conventions

This section describes our software programming conventions that supplement the basic architecture.

7.1 Reserved Registers

It is convenient to reserve a number of registers for pervasive standard uses. The hardware itself reserves R31 and R30; in addition, our software conventions reserve R29, R28, and R27.

These are summarized in the following table and are described more fully below.

Register	Symbol	Usage
R31	R31	Always zero
R30	XP	Exception pointer
R29	SP	Stack pointer
R28	LP	Linkage pointer
R27	BP	Base of frame pointer

7.2 Convenience Macros

We augment the basic β instruction set with the following macros, making it easier to express certain common operations:

Macro	Definition
BEQ(Ra, label)	BEQ(Ra, label, R31)
BF(Ra, label)	BF(Ra, label, R31)
BNE(Ra, label)	BNE(Ra, label, R31)
BT(Ra, label)	BT(Ra, label, R31)
BR(label, Rc)	BEQ(R31, label, Rc)
BR(label)	BR(label, R31)
JMP(Ra)	JMP(Ra, R31)
LD(label, Rc)	LD(R31, label, Rc)
ST(Rc, label)	ST(Rc, label, R31)
MOVE(Ra, Rc)	ADD(Ra, R31, Rc)
CMOVE(c, Rc)	ADDC(R31, c, Rc)

PUSH(Ra)	ADDC(SP, 4, SP)
	ST(Ra, -4, SP)

POP(Rc)	LD(SP, -4, Rc)
	SUBC(SP, 4, SP)

ALLOCATE(k)	ADDC(SP, 4*k, SP)
DEALLOCATE(k)	SUBC(SP, 4*k, SP)

7.3 Stack Implementation

SP is a reserved register that points to the top of the stack. The stack is an arbitrary contiguous region of memory. The contents of SP are always a multiple of 4 and each stack slot is 4 bytes. SP points to the location just beyond the topmost element on the stack. The stack grows upward in memory (i.e., towards higher addresses). Four macros are defined for manipulating the stack:

PUSH(Ra) - Push the contents of register Ra onto the stack

POP(Rc) - Pop the top element of the stack into Rc

ALLOCATE(k) - Push k words of uninitialized data onto the stack

DEALLOCATE(k) - Pop k words off of the stack and throw them away

7.4 Procedure Linkage

A procedure's arguments are passed on the stack. Specifically, when a procedure is entered, the topmost element on the stack is the first argument to the procedure; the next element on the stack is the second argument to the procedure, and so on. A procedure's return address is passed in LP, which is a register reserved for this purpose. A procedure returns its value (if any) in R0 and must leave all other registers, including the reserved registers, unaltered.

Thus, a typical call to a procedure named F looks like:

(push arg_{n-1})
...
(push arg_1)
(push arg_0)
BR (F, LP)
DEALLOCATE (n)
(use R0, which is now F(arg_0, arg_1, ... , arg_{n-1}))