Arithmetic integer operate instructions

addq	add	+
subq	subtract	-
mulq	multiply	*
umulh	top half of 128 bit multiply	*
divq/divqu	divide, signed/unsigned	/
modq/modqu	modulo, signed/unsigned	%
s8addq	scaled 8 add	8*operandA+operandB
S4addq	scaled 4 add	4*operandA+operandB

Shift integer operate instructions

sll	shift left logical	<<
srl	shift right logical	>>>
sra	shift right arithmetic	>>

Compare integer operate instructions

cmpeq	compare equal	==
cmplt/cmpult	compare less than signed/unsigned	<
cmple/cmpule	compare less than or equal signed/unsigned	<=

Logical integer operate instructions

and	and	&
bic	bit clear	& ~
bis/or	bit set/or	|
eqv/xornot	equivalent/exclusive or not	^ ~
ornot	or not	| ~
xor	exclusive or	^

cmoveq	conditional move equal
cmovne	conditional move not equal
cmovlt	conditional move less than
cmovle	conditional move less than or equal
cmovgt	conditional move greater than
cmovge	conditional move greater than or equal
cmovlbs	conditional move low bit set
cmovlbc	conditional move low bit clear

Memory instructions

```
Opcode $regA, displacement($regB)
Opcode $regA, ($regB)
Opcode $regA, constant
```

The displacement or constant is a 16 bit signed constant.

Load address instruction

intReg[regA] = displacement + intReg[regB]

lda	load address

Load memory instructions

intReg[regA] = Memory[displacement + intReg[regB]]

ldq	load quadword
ldl	load longword
ldwu	load word unsigned
ldbu	load byte unsigned

Store memory instructions

Memory[displacement + intReg[regB]] = intReg[regA]

stq	store quadword
stl	store longword
stw	store word
stb	store byte

Branch instructions

Conditional branch instructions

```
Opcode $regA, destination
```

if (relation holds for intReg[regA])
    programCounter = destination

beq	branch equal
bne	branch not equal
blt	branch less than
ble	branch less than or equal
bgt	branch greater than
bge	branch greater than or equal
blbs	branch low bit set
blbc	branch low bit clear

Unconditional branch instructions

```
Opcode destination;
```

programCounter = destination // br

intReg[ra] = programCounter // bsr
programCounter = destination

br	branch
bsr	branch to subroutime

Jump instruction

```
Opcode ($regA);
```

programCounter = intReg[regA] // jmp

intReg[ra] = programCounter // jsr
programCounter = intReg[regA]

jmp	jump
jsr	jump to subroutine

Return instruction

programCounter = intReg[ra]

ret	return

Callpal instruction

```
call_pal constant;
```

The constant is a 26 bit constant.

call_pal	call PALcode

Pseudoinstructions

Load immediate

```
ldiq $regA, constant
```

The constant is a 64 bit constant.
intReg[regA] = constant

ldiq	load immediate quadword

Clear

```
clr $regA
```

intReg[regA] = 0

clr	clear

Unary pseudoinstructions

```
Opcode $regB, $regC
```

intReg[regC] = op intReg[regB]

```
Opcode constantB, $regC
```

The constant is an 8 bit unsigned constant.
intReg[regC] = op constantB

mov	move
negq	negate