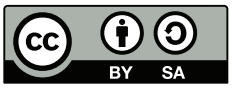
Instruction set of the ULM (Ulm Lecture Machine) 12 February 2019



Michael C. Lehn

Contents

Chapter 1

Instruction format and notation

1.1 Data types

The ULM architecture supports the following data types:

Byte	8 bits.
Word	16 bits.
Longword	32 bits.
Quadword	64 bits.

1.2 Signed and unsigned integer

If X is a bit pattern then $s(X)$ denotes its signed integer value and $u(X)$ its unsigned value. Signed integers are represented using two's complement format.

1.3 General-purpose register file and zero register

The zero register is named %0 and the 255 general-purpose registers are named %1 to %255. All these registers are 64-bit registers.

The zero register can be used in instructions like a general purpose register. In this case the zero register reads as zero when used as a source register and discards the result when used as a destination register.

The notation for referring to bit ranges assumes that the bits of a register are written down from highest to lowest significance:

$$
\begin{aligned}
\%X &= (x_{63}, \ldots, x_0) \\
u(\%X) \bmod 2^8 &= u(x_7, \ldots, x_0) \quad \text{(low byte)} \\
u(\%X) \bmod 2^{16} &= u(x_{15}, \ldots, x_0) \quad \text{(low word)} \\
u(\%X) \bmod 2^{32} &= u(x_{31}, \ldots, x_0) \quad \text{(low longword)}
\end{aligned}
$$

1.4 Endianness and notation for data in memory

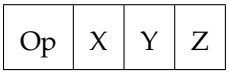
The ULM stores data in *big endian*. With $M_k(x)$ we denote the k consecutive bytes starting at address x. So in particular:

Notation	Meaning
$M_1(x)$	byte at address x
$M_2(x)$	word at address x
$M_4(x)$	long at address x
$M_8(x)$	quad at address x

Note that an address is just a 64 bit pattern that can be interpreted as an unsigned integer. However, this interpretation is not relevant for the architecture so we can simply write $M_k(\%X + \%Y)$ instead of $M_k(u_{64}(\%X + \%Y))$.

1.5 Instruction formats

Each instruction consists of 32 bits. The 4 bytes are denoted by Op, X, Y and Z:



(Op denotes the high byte)

1.5.1 Format RRR

The bytes X, Y and Z denote register names.

1.5.2 Format URR

The byte X denotes an unsigned immediate value. Bytes Y and Z denote register names.

1.5.3 Format SRR

The byte X denotes a signed immediate value. Bytes Y and Z denote register names.

1.5.4 Format U16R

The byte XY denotes a 16 bit unsigend immediate value. Byte Z denotes a register name.

1.5.5 Format S16R

The byte XY denotes a 16 bit sigend immediate value. Byte Z denotes a register name.

1.5.6 Format S24

The concatenated bit pattern of X, Y and Z is used for a relative jump. It encodes the number of instructions to jump ahead or back. Relative from the current instruction the jump destination has the offset $s\,(XYZ) \cdot 4$ in bytes, the instruction with address

$$u\,(\%\,\mathrm{ip}) + s\,(XYZ) \cdot 4$$

1.5.7 Empty format

An instruction (for example the *nop* instruction) does not use any operands. This is denoted by an empty format string.

1.5.8 Formats for describing memory locations

With **M(RR)** we denote that a memory address is the sum of two registers. And with **M(IR)** we denote that the memory address is the sum of a signed immediate value and a register.

1.5.8.1 Format M(RR)R

The memory address is $(u\,(\%X) + u\,(\%Y)) \bmod 2^{64}$.

1.5.8.2 Format M(IR)R

The memory address is $(s\,(X) + u\,(\%Y)) \bmod 2^{64}$.

1.5.8.3 Format RM(RR)

The memory address is $(u\,(\%Y) + u\,(\%Z)) \bmod 2^{64}$.

1.5.8.4 Format RM(IR)

The memory address is $(s\,(Y) + u\,(\%Z)) \bmod 2^{64}$.

Chapter 2

Instructions

2.1 addq

Integer addition.

If the first operand X is an immediate value it gets zero extended to 64 bits. The result of the integer addition gets stored in register Z.

Opcode	Format	Assembler notation	Effect
0x30	RRR	addq %x, %y, %z	$(u(\%X) + u(\%Y)) \bmod 2^{64} \rightarrow u(\%Z)$
0x38	URR	addq $x, %y, %z	$(u(X) + u(\%Y)) \bmod 2^{64} \rightarrow u(\%Z)$

Status flags

The status flags will be modified.

Format RRR

Flag	Condition
CF	$u(\%X) + u(\%Y) \geq 2^{64}$
OF	$s(\%X) + s(\%Y) \geq 2^{63} \quad \text{or} \quad s(\%X) + s(\%Y) < -2^{63}$
SF	$(u(\%X) + u(\%Y)) \bmod 2^{64} \geq 2^{63}$
ZF	$(u(\%X) + u(\%Y)) \bmod 2^{64} = 0$

Format URR

Flag	Condition
CF	$u(X) + u(\%Y) \geq 2^{64}$
OF	$s(X) + s(\%Y) \geq 2^{63} \quad \text{or} \quad s(X) + s(\%Y) < -2^{63}$
SF	$(u(X) + u(\%Y)) \bmod 2^{64} \geq 2^{63}$
ZF	$(u(X) + u(\%Y)) \bmod 2^{64} = 0$

2.2 andq

Bitwise AND operation.

If the first operand X is an immediate value it gets zero extended to 64 bits. The result of the bitwise AND operation gets stored in register Z.

Opcode	Format	Assembler notation	Effect
0x51	RRR	andq %X, %Y, %Z	$\mathrm{v}\,(\%X) \wedge \mathrm{v}\,(\%Y) \rightarrow \mathrm{v}\,(\%Z)$

Notation for vector of bits

With $\mathrm{v}\,(X)$ the content of a bit pattern X considered as vector of bits. Bit operations for bit vetors are defined component wise:

$$0 \wedge 0 = 0 \qquad 0 \vee 0 = 0$$

$$0 \wedge 1 = 0 \qquad 0 \vee 1 = 1$$

$$1 \wedge 0 = 0 \qquad 1 \vee 0 = 1$$

$$1 \wedge 1 = 1 \qquad 1 \vee 1 = 1$$

Furthermore, with $\overline{\mathrm{v}}(X)$ we denote the bitwise complement of $\mathrm{v}\,(X)$.

Status flags

The ZF gets updated:

Flag	Condition
ZF	$\%Z = 0$

2.3 divq

Unsigned integer division.

Divides a 128 bit unsigned integer stored in %Y and %Y+1 by a 64 bit unsigned integer (either $x = u\,(\%X)$ or $x = u\,(X)$). The quotient of the division is stored in registers %Z and %(Z+1). The remainder in %(Z+2).

Opcode	Format	Assembler notation	Effect
0x33	RRR	divq %X, %Y, %Z	$\left\lfloor \frac{u(\%(Y+1)\,\%Y)}{u(\%X)} \right\rfloor \rightarrow u\,(\%(Z+1)\,\%Z)$
			$u\,(\%(Y+1)\,\%Y) \bmod u\,(\%X) \rightarrow u\,(\%(Z+2))$
0x3B	URR	divq %X, %Y, %Z	$\left\lfloor \frac{u(\%(Y+1)\,\%Y)}{u(X)} \right\rfloor \rightarrow u\,(\%(Z+1)\,\%Z)$
			$u\,(\%(Y+1)\,\%Y) \bmod u\,(X) \rightarrow u\,(\%(Z+2))$

2.4 getc

Get a character from the input device.

Blocks the ULM until a (character) byte was read from the input device into a register.

Opcode	Format	Assembler notation	Effect
0x60	R	getc %X	in $\rightarrow u\,(\%X)$

2.5 halt

Halt program execution.

Halts the execution with an 8 bit exit code specified by the operand. The exit code is left in the last memory cell.

Opcode	Format	Assembler notation	Effect
0x00	R	halt %X	$u\,(\%X) \rightarrow u(M_1(2^{64}-1)),$ halt.
0x08	I	halt $X	$u\,(X) \rightarrow u(M_1(2^{64}-1)),$ halt.

2.6 idivq

Signed integer division.

Divides a 64 bit signed integer stored in %Y either by the 64 bit signed integer stored in %X or by $s_{64}(X)$. The division always rounds towards zero. The quotient of the division is stored in register %Z and the remainder in %(Z+1).

Opcode	**Format**	**Assembler notation**	**Effect**
0x35	RRR	idivq %X, %Y, %Z	$\left\lfloor \frac{s(\%Y)}{s(\%X)} \right\rfloor \rightarrow s\,(\%Z)$
			$s\,(\%Y) \bmod s\,(\%X) \rightarrow s\,(\%(Z{+}1))$
0x3D	SRR	idivq $X, %Y, %Z	$\left\lfloor \frac{s(\%Y)}{s(X)} \right\rfloor \rightarrow s\,(\%Z)$
			$s\,(\%Y) \bmod s\,(X) \rightarrow s\,(\%(Z{+}1))$

Internally it computes with $x = s\,(\%X)$ (or $x = s\,(X)$)

$$
\begin{aligned}
s &= \begin{cases} 1, & s_{64}(\%Y) \cdot x > 0, \\ -1, & \text{else.} \end{cases} \\
q &= s \cdot \left\lfloor \frac{|s_{64}(\%Y)|}{|x|} \right\rfloor \\
r &= s_{64}(\%Y) - x \cdot q
\end{aligned}
$$

and then stores q and r in registers %Z and %Z+1 respectively:

$$
\begin{aligned}
q &\rightarrow \%Z \\
r &\rightarrow \%(Z+1)
\end{aligned}
$$

2.7 imulq

Signed integer multiplication.

Computes the product of two signed 64 bit integers.

Opcode	Format	Assembler notation	Effect
0x34	RRR	imulq %x, %y, %z	$(s(\%X) \cdot s(\%Y)) \bmod 2^{64} \rightarrow s(\%Z)$
0x3C	SRR	imulq $x, %y, %z	$(s(X) \cdot s(\%Y)) \bmod 2^{64} \rightarrow s(\%Z)$

Status flags

The status flags OF and CF will be modified.

Format RRR

Flag	Condition
CF	$s(\%X) \cdot s(\%Y) \neq (s(\%X) \cdot s(\%Y)) \bmod 2^{64}$
OF	$s(\%X) \cdot s(\%Y) \neq (s(\%X) \cdot s(\%Y)) \bmod 2^{64}$

Format SRR

Flag	Condition
CF	$s(X) \cdot s(\%Y) \neq (s(\%X) \cdot s(\%Y)) \bmod 2^{64}$
OF	$s(X) \cdot s(\%Y) \neq (s(\%X) \cdot s(\%Y)) \bmod 2^{64}$

2.8 ja / jnbe

Unsigned conditional jump.

The instructions *ja* (jump if above) and *jnbe* (jump if not below equal) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy $(\mathrm{CF} = 0) \wedge (\mathrm{ZF} = 0)$.

Opcode	Format	Assembler notation	Effect
0x4B	S24	ja $XYZ	If $(\mathrm{CF} = 0) \wedge (\mathrm{ZF} = 0)$:
			$(u\,(\%\,\mathrm{ip}) + 4 \cdot s\,(XYZ)) \bmod 2^{64} \rightarrow u\,(\%\,\mathrm{ip})$
0x4B	S24	jnbe $XYZ	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b > 0$ where a and b are both unsigned integers.

2.9 jae / jnb

Unsigned conditional jump.

The instructions *jae* (jump if above equal) and *jnb* (jump if not below) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy CF $= 0$.

Opcode	Format	Assembler notation	Effect
0x49	S24	jae $XYZ * 4	If CF $= 0$:
			$(u\,(\%\,\mathrm{ip}) + 4 \cdot s\,(XYZ)) \bmod 2^{64} \to u\,(\%\,\mathrm{ip})$
0x49	S24	jnb $XYZ * 4	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b \geq 0$ where a and b are both unsigned integers.

2.10 jnae / jb

Unsigned conditional jump.

The instructions *jnae* (jump if not above equal) and *jb* (jump if below) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy CF $= 1$.

Opcode	Format	Assembler notation	Effect
0x48	S24	jnae $XYZ * 4	If CF $= 1$:
			$(u\,(\%\,\mathrm{ip}) + 4 \cdot s\,(XYZ)) \bmod 2^{64} \to u\,(\%\,\mathrm{ip})$
0x48	S24	jb $XYZ * 4	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b < 0$ where a and b are both unsigned integers.

2.11 jna / jbe

Unsigned conditional jump.

The instructions *jna* (jump if not above) and *jbe* (jump if below equal) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy $(\mathrm{CF} = 1) \wedge (\mathrm{ZF} = 1)$.

Opcode	Format	Assembler notation	Effect
0x4A	S24	jnae $XYZ * 4	If $(\mathrm{CF} = 1) \wedge (\mathrm{ZF} = 1)$:
			$(u\,(\%\,\mathrm{ip}) + 4 \cdot s\,(XYZ)) \bmod 2^{64} \rightarrow u\,(\%\,\mathrm{ip})$
0x4A	S24	jb $XYZ * 4	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b \leq 0$ where a and b are both unsigned integers.

2.12 je / jz

Unsigned/signed conditional jump.

Instructions *je* (jump if equal) and *jz* (jump if zero) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy ZF $= 1$.

Opcode	Format	Assembler notation	Effect
0x42	S24	je $XYZ * 4	If ZF $= 1$:
			$(u\,(\%\,\mathrm{ip}) + 4 \cdot s\,(XYZ)) \bmod 2^{64} \to u\,(\%\,\mathrm{ip})$
0x42	S24	jz $XYZ * 4	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b = 0$ where a and b are either both signed or both unsigned integers.

2.13 jg / jnle

Signed conditional jump.

Instructions *jg* (jump if greater) and *jnle* (jump if not less equal) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy $(\mathrm{ZF} = 1) \wedge (\mathrm{SF} = \mathrm{OF})$.

Opcode	Format	Assembler notation	Effect
0x47	S24	jg $XYZ * 4	If $(\mathrm{ZF} = 1) \wedge (\mathrm{SF} = \mathrm{OF})$: $(u\,(\%\,\mathrm{ip}) + 4 \cdot s\,(XYZ)) \bmod 2^{64} \rightarrow u\,(\%\,\mathrm{ip})$
0x47	S24	jnle $XYZ * 4	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b > 0$ where a and b are both signed integers.

2.14 jge / jnl

Signed conditional jump.

Instructions *jge* (jump if greater equal) and *jnl* (jump if not less) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy SF = OF.

Opcode	Format	Assembler notation	Effect
0x45	S24	jge $XYZ * 4	If SF = OF:
			$(u(\%\,\mathrm{ip}) + 4 \cdot s(XYZ)) \bmod 2^{64} \to u(\%\,\mathrm{ip})$
0x45	S24	jnl $XYZ * 4	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b \geq 0$ where a and b are both signed integers.

2.15 jl / jnge

Signed conditional jump.

Instructions *jl* (jump if less) and *jnge* (jump if not greater equal) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy SF $\neq$ OF.

Opcode	Format	Assembler notation	Effect
0x44	S24	jl $XYZ * 4	If SF $\neq$ OF:
			$(u\,(\%\,\mathrm{ip}) + 4 \cdot s\,(XYZ)) \bmod 2^{64} \rightarrow u\,(\%\,\mathrm{ip})$
0x44	S24	jnge $XYZ * 4	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b < 0$ where a and b are both signed integers.

2.16 jle / jng

Signed conditional jump.

Instructions *jle* (jump if less) and *jng* (jump if not greater equal) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy $(\mathrm{ZF} = 1) \wedge (\mathrm{SF} \neq \mathrm{OF})$.

Opcode	Format	Assembler notation	Effect
0x46	S24	jle $XYZ * 4	If $(\mathrm{ZF} = 1) \wedge (SF \neq OF)$: $(u(\%\,\mathrm{ip}) + 4 \cdot s(XYZ)) \bmod 2^{64} \rightarrow u(\%\,\mathrm{ip})$
0x46	S24	jng $XYZ * 4	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b \leq 0$ where a and b are both signed integers.

2.17 jmp

Unconditional jump.

Opcode	Format	Assembler notation	Effect
0x40	RR	jmp %X, %Y	$u\left(\%IP\right)+4 \rightarrow u\left(\%Y\right),\ u\left(\%X\right) \rightarrow u\left(\%\,\mathrm{ip}\right)$
0x41	S24	jmp $XYZ * 4	$\left(u\left(\%\,\mathrm{ip}\right)+4\cdot s\left(XYZ\right)\right) \bmod 2^{64} \rightarrow u\left(\%\,\mathrm{ip}\right)$

Notes for programming

Format: RR

The instruction jumps to the absolute address stored in %X. The instruction stores the (absolute) return address in %Y.

Format: S24

The 24 bit operand is used for a (signed) 26 bit relative jump address.

2.18 jne / jnz

Unsigned/signed conditional jump.

Instructions *jne* (jump if not equal) and *jnz* (jump if not zero) are identical.

The 24 bit operand is used for a (signed) 26 bit relative jump address. The condition for a jump is met, if and only if the status flags satisfy ZF $= 0$.

Opcode	Format	Assembler notation	Effect
0x43	S24	jne $XYZ * 4	If ZF $= 0$:
			$(u\,(\%\,\mathrm{ip}) + 4 \cdot s\,(XYZ)) \bmod 2^{64} \rightarrow u\,(\%\,\mathrm{ip})$
0x43	S24	jnz $XYZ * 4	

Notes for programming

Together with *subq* the instruction can be used to jump if $a - b \neq 0$ where a and b are either both signed or both unsigned integers.

2.19 ldswq

Load signed word in register.

The first operand XY is a signed 16 bit immediate value it gets signed extended to 64 bits and stored in register Z.

Opcode	Format	Assembler notation	Effect
0x57	S16R	ldswq $XY, %Z	$s\left(XY\right) \rightarrow s\left(\%Z\right)$

Status flags

The ZF gets updated:

Flag	Condition
ZF	$u\left(\%Z\right) = 0$

2.20 ldzwq

Load unsigned word in register.

The first operand XY is an unsigned 16 bit immediate value it gets zero extended to 64 bits and stored in register Z.

Opcode	Format	Assembler notation	Effect
0x56	U16R	ldswq $XY, %Z	$u\left(XY\right) \rightarrow u\left(\%Z\right)$

Status flags

The ZF gets updated:

Flag	Condition
ZF	$u\left(\%Z\right) = 0$

2.21 movb

Move byte.

Store a byte to memory.

Opcode	Format	Assembler notation	Effect
0x23	RM(RR)	movb %X, (%Y, %Z)	$u\left(\%X\right) \bmod 2^8 \rightarrow u\left(M_1(A)\right)$
0x2B	RM(UR)	movb %X, Y(%Z)	$u\left(\%X\right) \bmod 2^8 \rightarrow u\left(M_1(A)\right)$
0x93	RM(RR)	movb %X, (%Y, %Z, 2)	$u\left(\%X\right) \bmod 2^8 \rightarrow u\left(M_1(A)\right)$
0xB3	RM(RR)	movb %X, (%Y, %Z, 4)	$u\left(\%X\right) \bmod 2^8 \rightarrow u\left(M_1(A)\right)$
0xD3	RM(RR)	movb %X, (%Y, %Z, 8)	$u\left(\%X\right) \bmod 2^8 \rightarrow u\left(M_1(A)\right)$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := \left(u\left(\%Base\right) + scale \cdot u\left(\%Offset\right)\right) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := \left(u\left(\%Base\right) + u\left(\%Offset\right)\right) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := \left(u\left(\%Base\right) + s\left(displace\right)\right) \bmod 2^{64}$

2.22 movl

Move long.

Store a long to memory.

Opcode	Format	Assembler notation	Effect
0x21	RM(RR)	movl %X, (%Y, %Z)	$u\,(\%X) \bmod 2^{32} \rightarrow u\,(M_4(A))$
0x29	RM(UR)	movl %X, Y(%Z)	$u\,(\%X) \bmod 2^{32} \rightarrow u\,(M_4(A))$
0x91	RM(RR)	movl %X, (%Y, %Z, 2)	$u\,(\%X) \bmod 2^{32} \rightarrow u\,(M_4(A))$
0xB1	RM(RR)	movl %X, (%Y, %Z, 4)	$u\,(\%X) \bmod 2^{32} \rightarrow u\,(M_4(A))$
0xD1	RM(RR)	movl %X, (%Y, %Z, 8)	$u\,(\%X) \bmod 2^{32} \rightarrow u\,(M_4(A))$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := (u\,(\%Base) + scale \cdot u\,(\%Offset)) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := (u\,(\%Base) + u\,(\%Offset)) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := (u\,(\%Base) + s\,(displace)) \bmod 2^{64}$

Alignment

The computed address has to be aligned:

$$A \bmod 4 = 0$$

2.23 movq

Move quad.

Store a quad in memory or fetch a quad from memory.

Opcode	Format	Assembler notation	Effect
0x10	M(RR)R	movq (%X, %Y), %Z	$u(M_8(A) \rightarrow \%Z$
0x18	M(UR)R	movq X(%Y), %Z	$u(M_8(A) \rightarrow \%Z$
0x20	RM(RR)	movq %X, (%Y, %Z)	$u(\%X) \rightarrow u(M_8(A))$
0x28	RM(UR)	movq %X, Y(%Z)	$u(\%X) \rightarrow u(M_8(A))$
0x80	M(RR)R	movq (%X, %Y, 2), %Z	$u(M_8(A)) \rightarrow u(\%Z)$
0x90	RM(RR)	movq %X, (%Y, %Z, 2)	$u(\%X) \rightarrow u(M_8(A))$
0xA0	M(RR)R	movq (%X, %Y, 4), %Z	$u(M_8(A)) \rightarrow u(\%Z)$
0xB0	RM(RR)	movq %X, (%Y, %Z, 4)	$u(\%X) \rightarrow u(M_8(A))$
0xC0	M(RR)R	movq (%X, %Y, 8), %Z	$u(M_8(A)) \rightarrow u(\%Z)$
0xD0	RM(RR)	movq %X, (%Y, %Z, 8)	$u(\%X) \rightarrow u(M_8(A))$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := (u(\%Base) + scale \cdot u(\%Offset)) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := (u(\%Base) + u(\%Offset)) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := (u(\%Base) + s(displace)) \bmod 2^{64}$

Alignment

The computed address has to be aligned:

$$A \bmod 8 = 0$$

2.24 movsbq

Move signed byte.

Fetch a signed byte from memory. The fetched byte gets sign extended to 64 bits and stored in register %Z.

Opcode	Format	Assembler notation	Effect
0x17	M(RR)R	movsbq (%X, %Y), %Z)	$s\left(M_1(A)\right) \rightarrow s\left(\%Z\right)$
0x1F	M(UR)R	movsbq X(%Y), %Z	$s\left(M_1(A)\right) \rightarrow s\left(\%Z\right)$
0x87	M(RR)R	movsbq (%X, %Y, 2), %Z)	$s\left(M_1(A)\right) \rightarrow s\left(\%Z\right)$
0xA7	M(RR)R	movsbq (%X, %Y, 4), %Z)	$s\left(M_1(A)\right) \rightarrow s\left(\%Z\right)$
0xC7	M(RR)R	movsbq (%X, %Y, 8), %Z)	$s\left(M_1(A)\right) \rightarrow s\left(\%Z\right)$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := \left(u\left(\%Base\right) + scale \cdot u\left(\%Offset\right)\right) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := \left(u\left(\%Base\right) + u\left(\%Offset\right)\right) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := \left(u\left(\%Base\right) + s\left(displace\right)\right) \bmod 2^{64}$

2.25 movslq

Move signed long.

Fetch a signed long from memory. The fetched long gets sign extended to 64 bits and stored in register %Z.

Opcode	Format	Assembler notation	Effect
0x15	M(RR)R	movslq (%X, %Y), %Z	$s\left(M_4(A)\right) \rightarrow s\left(\%Z\right)$
0x1D	M(UR)R	movslq X(%Y), %Z	$s\left(M_4(A)\right) \rightarrow s\left(\%Z\right)$
0x85	M(RR)R	movslq (%X, %Y, 2), %Z	$s\left(M_4(A)\right) \rightarrow s\left(\%Z\right)$
0xA5	M(RR)R	movslq (%X, %Y, 4), %Z	$s\left(M_4(A)\right) \rightarrow s\left(\%Z\right)$
0xC5	M(RR)R	movslq (%X, %Y, 8), %Z	$s\left(M_4(A)\right) \rightarrow s\left(\%Z\right)$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := \left(u\left(\%Base\right) + scale \cdot u\left(\%Offset\right)\right) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := \left(u\left(\%Base\right) + u\left(\%Offset\right)\right) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := \left(u\left(\%Base\right) + s\left(displace\right)\right) \bmod 2^{64}$

Alignment

The computed address has to be aligned:

$$A \bmod 4 = 0$$

2.26 movswq

Move signed word.

Fetch a signed word from memory. The fetched word gets sign extended to 64 bits and stored in register %Z.

Opcode	Format	Assembler notation	Effect
0x16	M(RR)R	movswq (%X, %Y), %Z)	$s\left(M_2(A)\right) \rightarrow s\left(\%Z\right)$
0x1E	M(UR)R	movswq X(%Y), %Z	$s\left(M_2(A)\right) \rightarrow s\left(\%Z\right)$
0x86	M(RR)R	movswq (%X, %Y, 2), %Z)	$s\left(M_2(A)\right) \rightarrow s\left(\%Z\right)$
0xA6	M(RR)R	movswq (%X, %Y, 4), %Z)	$s\left(M_2(A)\right) \rightarrow s\left(\%Z\right)$
0xC6	M(RR)R	movswq (%X, %Y, 8), %Z)	$s\left(M_2(A)\right) \rightarrow s\left(\%Z\right)$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := \left(u\left(\%Base\right) + scale \cdot u\left(\%Offset\right)\right) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := \left(u\left(\%Base\right) + u\left(\%Offset\right)\right) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := \left(u\left(\%Base\right) + s\left(displace\right)\right) \bmod 2^{64}$

Alignment

The computed address has to be aligned:

$$A \bmod 2 = 0$$

2.27 movw

Move word.

Store a word to memory.

Opcode	Format	Assembler notation	Effect
0x22	RM(RR)	movw %X, (%Y, %Z)	$u\,(\%X) \bmod 2^{16} \rightarrow M_2(A)$
0x2A	RM(UR)	movw %X, Y(%Z)	$u\,(\%X) \bmod 2^{16} \rightarrow M_2(A)$
0x92	RM(RR)	movw %X, (%Y, %Z, 2)	$u\,(\%X) \bmod 2^{16} \rightarrow M_2(A)$
0xB2	RM(RR)	movw %X, (%Y, %Z, 4)	$u\,(\%X) \bmod 2^{16} \rightarrow M_2(A)$
0xD2	RM(RR)	movw %X, (%Y, %Z, 8)	$u\,(\%X) \bmod 2^{16} \rightarrow M_2(A)$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := (u\,(\%Base) + scale \cdot u\,(\%Offset)) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := (u\,(\%Base) + u\,(\%Offset)) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := (u\,(\%Base) + s\,(displace)) \bmod 2^{64}$

Alignment

The computed address has to be aligned:

$$A \bmod 2 = 0$$

2.28 movzbq

Move unsigned byte.

Fetch a unsigned word from memory. The fetched word gets zero extended to 64 bits and stored in register %Z.

Opcode	Format	Assembler notation	Effect
0x13	M(RR)R	movzbq (%X, %Y), %Z)	$u\left(M_1(A)\right) \rightarrow u\left(\%Z\right)$
0x1B	M(UR)R	movzbq X(%Y), %Z	$u\left(M_1(A)\right) \rightarrow u\left(\%Z\right)$
0x83	M(RR)R	movzbq (%X, %Y, 2), %Z)	$u\left(M_1(A)\right) \rightarrow u\left(\%Z\right)$
0xA3	M(RR)R	movzbq (%X, %Y, 4), %Z)	$u\left(M_1(A)\right) \rightarrow u\left(\%Z\right)$
0xC3	M(RR)R	movzbq (%X, %Y, 8), %Z)	$u\left(M_1(A)\right) \rightarrow u\left(\%Z\right)$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := \left(u\left(\%Base\right) + scale \cdot u\left(\%Offset\right)\right) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := \left(u\left(\%Base\right) + u\left(\%Offset\right)\right) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := \left(u\left(\%Base\right) + s\left(displace\right)\right) \bmod 2^{64}$

2.29 movzlq

Move unsigned long.

Fetch a unsigned long from memory. The fetched long gets zero extended to 64 bits and stored in register %Z.

Opcode	Format	Assembler notation	Effect
0x11	M(RR)R	movzlq (%X, %Y), %Z)	$u\left(M_4(A)\right) \rightarrow u\left(\%Z\right)$
0x19	M(UR)R	movzlq X(%Y), %Z	$u\left(M_4(A)\right) \rightarrow u\left(\%Z\right)$
0x81	M(RR)R	movzlq (%X, %Y, 2), %Z)	$u\left(M_4(A)\right) \rightarrow u\left(\%Z\right)$
0xA1	M(RR)R	movzlq (%X, %Y, 4), %Z)	$u\left(M_4(A)\right) \rightarrow u\left(\%Z\right)$
0xC1	M(RR)R	movzlq (%X, %Y, 8), %Z)	$u\left(M_4(A)\right) \rightarrow u\left(\%Z\right)$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := \left(u\left(\%Base\right) + scale \cdot u\left(\%Offset\right)\right) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := \left(u\left(\%Base\right) + u\left(\%Offset\right)\right) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := \left(u\left(\%Base\right) + s\left(displace\right)\right) \bmod 2^{64}$

Alignment

The computed address has to be aligned:

$$A \bmod 4 = 0$$

2.30 movzwq

Move unsigned word.

Fetch a unsigned word from memory. The fetched word gets zero extended to 64 bits and stored in register %Z.

Opcode	Format	Assembler notation	Effect
0x12	M(RR)R	movzwq (%X, %Y), %Z)	$u\left(M_2(A)\right) \rightarrow u\left(\%Z\right)$
0x1A	M(UR)R	movzwq X(%Y), %Z	$u\left(M_2(A)\right) \rightarrow u\left(\%Z\right)$
0x82	M(RR)R	movzwq (%X, %Y, 2), %Z)	$u\left(M_2(A)\right) \rightarrow u\left(\%Z\right)$
0xA2	M(RR)R	movzwq (%X, %Y, 4), %Z)	$u\left(M_2(A)\right) \rightarrow u\left(\%Z\right)$
0xC2	M(RR)R	movzwq (%X, %Y, 8), %Z)	$u\left(M_2(A)\right) \rightarrow u\left(\%Z\right)$

Addressing

Depending on the format for the memory location address A gets computed as follows:

M(RR)	$(\%Base, \%Offset, scale)$	$A := \left(u\left(\%Base\right) + scale \cdot u\left(\%Offset\right)\right) \bmod 2^{64}$
M(RR)	$(\%Base, \%Offset)$	$A := \left(u\left(\%Base\right) + u\left(\%Offset\right)\right) \bmod 2^{64}$
M(UR)	$displace(\%Base)$	$A := \left(u\left(\%Base\right) + s\left(displace\right)\right) \bmod 2^{64}$

Alignment

The computed address has to be aligned:

$$A \bmod 2 = 0$$

2.31 mulq

Unsigned integer multiplication.

Computes the 128 bit product of two unsigned 64 bit integers. The lower quad of the result is stored in register %Z, the upper quad in register %(Z+1).

Opcode	Format	Assembler notation	Effect
0x32	RRR	mulq %x, %y, %z	$u(\%X) \cdot u(\%Y) \rightarrow u(\%(Z+1)\,\%Z)$
0x3A	URR	mulq $x, %y, %z	$u(X) \cdot u(\%Y) \rightarrow u(\%(Z+1)\,\%Z)$

Status flags

The CF and OF gets updated:

Format RRR

Flag	Condition
CF	$(u(\%X) \cdot u(\%Y)) \bmod 2^{64} = 0$
OF	$(u(\%X) \cdot u(\%Y)) \bmod 2^{64} = 0$

Format URR

Flag	Condition
CF	$(u(X) \cdot u(\%Y)) \bmod 2^{64} = 0$
OF	$(u(X) \cdot u(\%Y)) \bmod 2^{64} = 0$

2.32 nop

No operation.

Opcode	Format	Assembler notation	Effect
0xFF		nop	no effect

2.33 notq

Bitwise NOT operation.

Opcode	Format	Assembler notation	Effect
0x5E		notq %X, %Y	$\overline{\mathrm{v}}(\%X) \rightarrow \mathrm{v}\,(\%Y)$

Notation for vector of bits

With $\mathrm{v}\,(X)$ the content of a bit pattern X considered as vector of bits. Bit operations for bit vetors are defined component wise:

$$0 \wedge 0 = 0 \qquad 0 \vee 0 = 0$$

$$0 \wedge 1 = 0 \qquad 0 \vee 1 = 1$$

$$1 \wedge 0 = 0 \qquad 1 \vee 0 = 1$$

$$1 \wedge 1 = 1 \qquad 1 \vee 1 = 1$$

Furthermore, with $\overline{\mathrm{v}}(X)$ we denote the bitwise complement of $\mathrm{v}\,(X)$.

Status flags

The ZF gets updated:

Flag	Condition
ZF	$\%Z = 0$

2.34 orq

Bitwise OR operation.

If the first operand X is an immediate value it gets zero extended to 64 bits. The result of the bitwise OR operation gets stored in register Z.

Opcode	Format	Assembler notation	Effect
0x50	RRR	orq %X, %Y, %Z	$\mathrm{v}(\%X) \vee \mathrm{v}(\%Y) \rightarrow \mathrm{v}(\%Z)$

Notation for vector of bits

With $\mathrm{v}(X)$ the content of a bit pattern X considered as vector of bits. Bit operations for bit vetors are defined component wise:

$$0 \wedge 0 = 0 \qquad 0 \vee 0 = 0$$

$$0 \wedge 1 = 0 \qquad 0 \vee 1 = 1$$

$$1 \wedge 0 = 0 \qquad 1 \vee 0 = 1$$

$$1 \wedge 1 = 1 \qquad 1 \vee 1 = 1$$

Furthermore, with $\overline{\mathrm{v}}(X)$ we denote the bitwise complement of $\mathrm{v}(X)$.

Status flags

The ZF gets updated:

Flag	Condition
ZF	$\%Z = 0$

2.35 putc

Put character to output device.

Put a character byte specified by an immediate value or the least significant byte of a register to the output device.

Opcode	Format	Assembler notation	Effect
0x61	RRR	putc %X	$u\,(\%X) \bmod 2^8 \rightarrow u\,(\text{output device})$
0x69	URR	putc $X	$u\,(X) \rightarrow u\,(\text{output device})$

2.36 salq / shlq

Bitwise LEFT SHIFT operation.

Left shifts (multiplies) the bit pattern of %Y for a given count (either $u_{64}(\%X)$ or $u_{64}(X)$) and stores the result in %Z. The high-order bit is shifted into the CF (carry flag). The low-order bit is set to 0.

Opcode	Format	Assembler notation	Effect
0x52	RRR	shlq %X, %Y, %Z	
0x52	RRR	salq %X, %Y, %Z	$\left(2^{u(\%X)} \cdot u\left(\%Y\right)\right) \bmod 2^{64} \rightarrow u\left(\%Z\right)$
			$\left(2^{u(\%X)-63} \cdot u\left(\%Y\right)\right) \bmod 2 \rightarrow u\left(\mathrm{cf}\right)$
0x5A	URR	shlq $X, %Y, %Z	
0x5A	URR	salq $X, %Y, %Z	$\left(2^{u(X)} \cdot u\left(\%Y\right)\right) \bmod 2^{64} \rightarrow u\left(\%Z\right)$
			$\left(2^{u(X)-63} \cdot u\left(\%Y\right)\right) \bmod 2 \rightarrow u\left(\mathrm{cf}\right)$

2.37 sarq

Bitwise RIGHT SHIFT operation.

Right shifts (signed divide) the bit pattern of %Y for a given count (either $u\left(\%X\right)$ or $u\left(X\right)$) and stores the result in %Z. The high-order bit remains unchanged (rounding toward negative infinity).

Opcode	Format	Assembler notation	Effect
0x54	RRR	sarq %X, %Y, %Z	$\lfloor 2^{-u(\%X)} \cdot s\left(\%Y\right) \rfloor \rightarrow u\left(\%Z\right)$
0x5C	URR	sarq $X, %Y, %Z	$\lfloor 2^{-u(X)} \cdot s\left(\%Y\right) \rfloor \rightarrow u\left(\%Z\right)$

2.38 shldwq

Shift register and load word.

The first operand XY is an unsigned 16 bit, the second operand Z the destination register. The instruction left shifts the bit pattern of %Z by 16 positions and overwrites the last 16 bits with XY.

Opcode	Format	Assembler notation	Effect
0x5D	S16R	shldwq $XY, %Z	$u\,(\%Z) \cdot 2^{16} + u\,(XY) \rightarrow u\,(\%Z)$

Status flags

The high-order bit of %Z is shifted into the CF (carry flag). The ZF (zero flag) is set if and only if %Z is zero.

2.39 shrq

Bitwise RIGHT SHIFT operation.

Right shifts (unsigned divide) the bit pattern of %Y for a given count (either $u\left(\%X\right)$ or $u\left(X\right)$) and stores the result in %Z. The high-order bit is set to zero.

Opcode	Format	Assembler notation	Effect
0x53	RRR	shrq %X, %Y, %Z	$\left\lfloor 2^{-u(\%X)} \cdot u\left(\%Y\right) \right\rfloor \rightarrow u\left(\%Z\right)$
0x5B	URR	shrq $X, %Y, %Z	$\left\lfloor 2^{-u(X)} \cdot u\left(\%Y\right) \right\rfloor \rightarrow u\left(\%Z\right)$

2.40 subq

Integer subtraction of the first operand from the second storing the result in the destination register. The status flags will be modified.

Opcode	Format	Assembler notation	Effect
0x31	RRR	subq %x, %y, %z	$(-u(\%X) + u(\%Y)) \bmod 2^{64} \rightarrow u(\%Z)$
0x39	URR	subq $x, %y, %z	$(-u(X) + u(\%Y)) \bmod 2^{64} \rightarrow u(\%Z)$

Format RRR

Flag	Condition
ZF	$(-u(\%X) + u(\%Y)) \bmod 2^{64} = 0$
CF	$-u(\%X) + u(\%Y) < 0$
OF	$-s(\%X) + s(\%Y) < -2^{63}$ or $-s(\%X) + s(\%Y) \geq 2^{63}$
SF	$(-u(\%X) + u(\%Y)) \bmod 2^{64} \geq 2^{63}$

Format URR

Flag	Condition
ZF	$(-u(X) + u(\%Y)) \bmod 2^{64} = 0$
CF	$-u(X) + u(\%Y) < 0$
OF	$-u(X) + s(\%Y) < -2^{63}$ or $-u(\%X) + s(\%Y) \geq 2^{63}$
SF	$(-u(X) + u(\%Y)) \bmod 2^{64} \geq 2^{63}$

2.41 trap

Trap to execution environment.

The trap instruction interrupts the current execution, and invokes an implementation-dependent trap handler. The first operand X specifies the trap number, the operand Y gives the address of a parameter block whose size and structure depends on the trap number. The return value of the trap handler gets stored in register Z.

Opcode	Format	Assembler notation	Effect
0x02	RRR	trap %X, %Y, %Z	return value from trap handler → %Z

The default implementation provides two trap numbers:

Trap number	Name	Parameter block	Description
0	read	24 bytes	read system call in conformance to IEEE Std 1003.1-2017
1	write	24 bytes	write system call in conformance to IEEE Std 1003.1-2017

The trap handlers for *read* and *write* expect a parameter block of 24 bytes:

Offset	Size	Parameter	Description
0	4	*fd*	file descriptor
8	8	*buf*	address of input/output buffer
16	8	*nbyte*	size of the buffer

The return value which is stored in %Z is that of the corresponding system call except that $-errno$ is returned instead of -1.