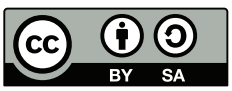
Instruction set of the ULM (Ulm Lecture Machine)



January 24, 2023

Contents

Chapter 1

Description of the ULM

1.1 Data Types

Binary digits are called *bits* and have the value 0 or 1. A *bit pattern* is a sequence of bits. For example

$$X := x_{n-1} \dots x_0 \text{ with } x_k \in \{0, 1\} \text{ for } 0 \leq k < n$$

denotes a bit pattern X with n bits. The number of bits in bit pattern is also called its size or width. The ULM architecture defines a *byte* as a bit pattern with 8 bits. Table 1.1 lists ULM's definitions for *word*, *long word*, *quad word* that refer to specific sizes of bit patterns.

1.2 Expressing the Interpretation of a Bit Pattern

For a bit pattern $X = x_{n-1} \dots x_0$ its *unsigned integer* value is expressed and defined through

$$u(X) = u(x_{n-1} \dots x_0) := \sum_{k=0}^{n-1} x_k \cdot 2^k$$

Signed integer values are represented using the *two's complement* and in this respect the notation

$$s(X) = s(x_{n-1}x_{n-2} \dots x_0) := \begin{cases} u(x_{n-2} \dots x_0), & \text{if } x_{n-1} = 0, \\ u(x_{n-2} \dots x_0) - 2^{n-1}, & \text{else} \end{cases}$$

is used.

1.3 Registers and Virtual Memory

The ULM has 256 registers denoted as %0x00, ..., %0xFF. Each of these registers has a width of 64 bits. The %0x00 is a special purpose register and also denoted as *zero register*. Reading form the zero register always gives a bit pattern where all bits have value 0 (zero bit pattern). Writing to the zero register has no effect.

The (virtual) memory of the ULM is an array of 2^{64} memory cells. Each memory cell can store exactly one byte. Each memory cell has an index which is called its *address*. The address is in the range from 0 to 2^{64-1} and the first memory cell of the array has address 0. In notations $M_1(a)$ denotes the memory cell with address a.

Data Size	Size in Bytes	Size in Number of Bits
Bytes	-	8
Word	2	16
Long Word	4	32
Quad Word	8	64

Table 1.1: Names for specific sizes of bit patterns.

1.3.1 Endianness

For referring to data in memory in quantities of words, long words and quad words the definitions

$$\begin{aligned} M_2(a) &:= M_1(a)M_1(a+1) \\ M_4(a) &:= M_2(a)M_2(a+2) \\ M_8(a) &:= M_4(a)M_4(a+4) \end{aligned}$$

are used. The ULM architecture is a *big endian* machine. Therefore we have the equalities

$$\begin{aligned} u(M_2(a)) &= u(M_1(a)M_1(a+1)) \\ u(M_4(a)) &= u(M_2(a)M_2(a+2)) \\ u(M_8(a)) &= u(M_4(a)M_4(a+4)) \end{aligned}$$

1.3.2 Alignment of Data

A quantity of k bytes are aligned in memory if they are stored at an address which is a multiple of k, i. e.

$$M_k(a) \text{ is aligned } \Leftrightarrow a \bmod k = 0$$

Chapter 2

Directives

2.1 .align <expr>

Pad the location counter (in the current segment) to a multiple of <expr>.

2.2 .bss

Set current segment to the BSS segment.

2.3 .byte <expr>

Expression is assembled into next byte.

2.4 .data

Set current segment to the data segment.

2.5 .equ <ident>, <expr>

Updates the symbol table. Sets the value of <ident> to <expr>.

2.6 .global <ident>

Updates the symbol table. Makes the symbol <ident> visible to the linker.

2.7 .globl <ident>

Equivalent to *.globl <ident>*:
Updates the symbol table. Makes the symbol <ident> visible to the linker.

2.8 .long <expr>

Expression <expr> is assembled into next long word (4 bytes).

2.9 .space <expr>

Emits <expr> bytes. Each byte with value 0x00.

2.10 .string <string-literal>

Emits bytes for the zero-terminated <string-literal>.

2.11 .text

Set current segment to the text segment.

2.12 .word <expr>

Expression <expr> is assembled into next word (2 bytes).

2.13 .quad <expr>

Expression <expr> is assembled into next quad word (8 bytes).

Chapter 3

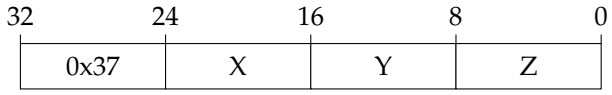
Instructions

3.1 adcq

3.1.1 Assembly Notation

adcq %X, %Y, %Z

Format



Effect

$$(u(\%Y) + u(\%X) + ZF) \bmod 2^{64} \rightarrow u(\%Z)$$

Updates the status flags:

Flag	Condition
ZF	$u(\%Y) + u(\%X) + ZF = 0$
CF	$u(\%Y) + u(\%X) + ZF \geq 2^{64}$
OF	$s(\%Y) + s(u(\%X) + ZF) \notin \{-2^{63}, \ldots, 2^{63} - 1\}$
SF	$s(\%Y) + s(u(\%X) + ZF) < 0$

3.2 addq

Integer addition. Adds either the content of register X or the unsigend immediate value X to register Y. The result is stored in register Z.

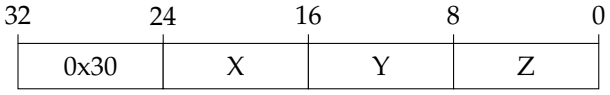
3.2.1 Assembly Notation

addq %X, %Y, %Z

Purpose

Adds register X with register Y. Stores the result in register Z.

Format



Effect

$$(u(\%Y) + u(\%X)) \bmod 2^{64} \rightarrow u\,(\%Z)$$

Updates the status flags:

Flag	Condition
ZF	$u(\%Y) + u(\%X) = 0$
CF	$u(\%Y) + u(\%X) \geq 2^{64}$
OF	$s(\%Y) + s(\%X) \notin \{-2^{63}, \ldots, 2^{63} - 1\}$
SF	$s(\%Y) + s(\%X) < 0$

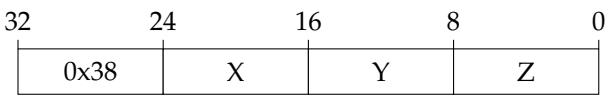
3.2.2 Assembly Notation

addq X, %Y, %Z

Purpose

Adds the unsigned immediate value X with the unsigned integer %Y. Stores the result in %Z. Updates the status flags.

Format



Effect

$$(u(\%Y) + u(X)) \bmod 2^{64} \to u\,(\%Z)$$

Updates the status flags:

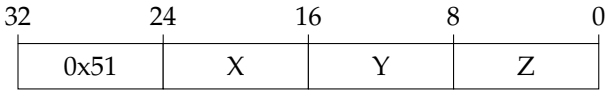
Flag	Condition
ZF	$u(\%Y) + u(X) = 0$
CF	$u(\%Y) + u(X) \geq 2^{64}$
OF	$s(\%Y) + s(X) \notin \{-2^{63}, \ldots, 2^{63} - 1\}$
SF	$s(\%Y) + s(X) < 0$

3.3 andq

3.3.1 Assembly Notation

andq %X, %Y, %Z

Format



Effect

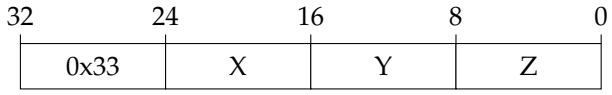
$u(\%X) \wedge_b u(\%Y) \rightarrow u\ (\%Z)$

3.4 divq

3.4.1 Assembly Notation

divq %X, %Y, %Z

Format



Effect

For the unsigned 128-bit numerator

$$b := u\,(\%\{u(Y)+1\}\%Y)$$

and unsigned 64-bia denominator

$$a := u(\%X)$$

computes the divisor

$$\left\lfloor \frac{b}{a} \right\rfloor \bmod 2^{128} \rightarrow u\,(\%\{u(Z)+1\}\%Z)$$

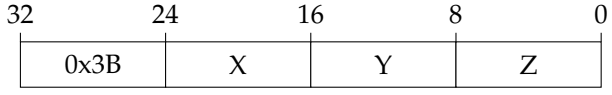
and the remainder

$$b \bmod a \rightarrow u\,(\%\{u(Z)+2\})$$

3.4.2 Assembly Notation

divq X, %Y, %Z

Format



Effect

For the unsigned 128-bit numerator

$$b := u\,(\%\{u(Y)+1\}\%Y)$$

and unsigned 64-bia denominator

$$a := u(X)$$

computes the divisor

$$\left\lfloor \frac{b}{a} \right\rfloor \bmod 2^{128} \rightarrow u\,(\%\{u(Z)+1\}\%Z)$$

and the remainder

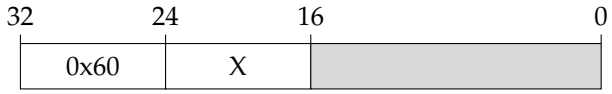
$$b \bmod a \rightarrow u\,(\%\{u(Z)+2\})$$

3.5 getc

3.5.1 Assembly Notation

getc %X

Format



Effect

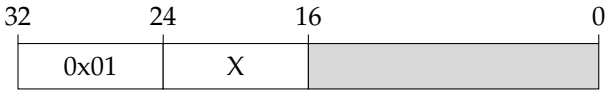
$s(ulm\_readChar()) \wedge_b 255 \bmod 2^{64} \rightarrow u\,(\%X)$

3.6 halt

3.6.1 Assembly Notation

halt %X

Format



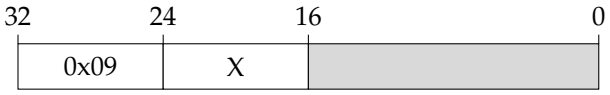
Effect

halt program execution with exit code $u(\%X) \bmod 2^8$

3.6.2 Assembly Notation

halt X

Format



Effect

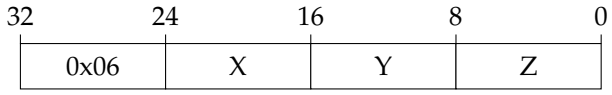
halt program execution with exit code $u(X) \bmod 2^8$

3.7 hm3Add

3.7.1 Assembly Notation

hm3Add X, %Y, %Z

Format



Effect

$$(u(\%Y) + u(X) \cdot 3) \bmod 2^{64} \rightarrow u(\%Z)$$

Updates the status flags:

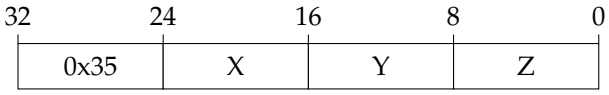
Flag	Condition
ZF	$u(\%Y) + u(X) \cdot 3 = 0$
CF	$u(\%Y) + u(X) \cdot 3 \geq 2^{64}$
OF	$s(\%Y) + s(u(X) \cdot 3) \notin \{-2^{63}, \ldots, 2^{63} - 1\}$
SF	$s(\%Y) + s(u(X) \cdot 3) < 0$

3.8 idivq

3.8.1 Assembly Notation

idivq %X, %Y, %Z

Format



Effect

Computes the divisor

$$\left\lfloor \frac{u(\%Y)}{u(\%X)} \right\rfloor_0 \rightarrow u\,(\%Z)$$

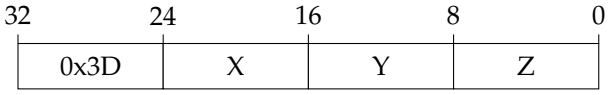
and the remainder

$$u(\%Y) \bmod u(\%X) \rightarrow u\,(\%Z)$$

3.8.2 Assembly Notation

idivq X, %Y, %Z

Format



Effect

Computes the divisor

$$\left\lfloor \frac{u(\%Y)}{u(X)} \right\rfloor_0 \rightarrow u\,(\%Z)$$

and the remainder

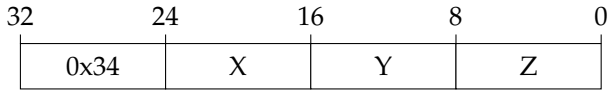
$$u(\%Y) \bmod u(X) \rightarrow u\,(\%Z)$$

3.9 imulq

3.9.1 Assembly Notation

imulq %X, %Y, %Z

Format



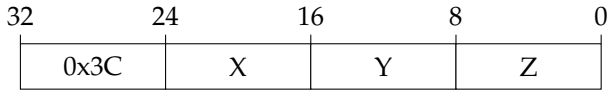
Effect

$u(\%X) \cdot u(\%Y) \bmod 2^{64} \rightarrow u\,(\%Z)$

3.9.2 Assembly Notation

imulq X, %Y, %Z

Format



Effect

$s(X) \cdot u(\%Y) \bmod 2^{64} \rightarrow u\,(\%Z)$

3.10 ja

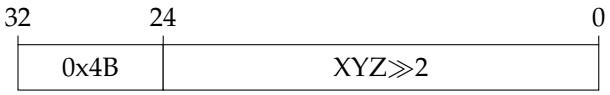
3.10.1 Assembly Notation

ja XYZ

Alternative Assembly Notation

jnbe XYZ

Format



Effect

If the condition

$$CF = 0 \wedge ZF = 0$$

evaluates to true then

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.11 jae

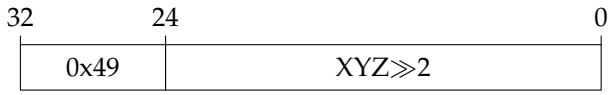
3.11.1 Assembly Notation

jae XYZ

Alternative Assembly Notation

jnb XYZ

Format



Effect

If the condition

$$CF = 0$$

evaluates to true then

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.12 jb

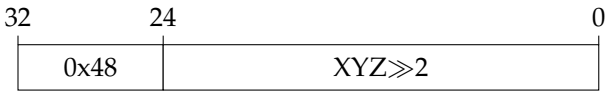
3.12.1 Assembly Notation

jb XYZ

Alternative Assembly Notation

jnae XYZ

Format



Effect

If the condition

$$CF = 1$$

evaluates to true then

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.13 jg

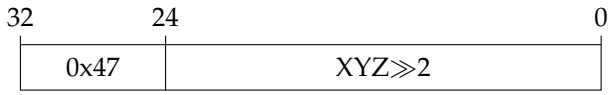
3.13.1 Assembly Notation

jg XYZ

Alternative Assembly Notation

jnle XYZ

Format



Effect

If the condition

$$ZF = 0 \wedge SF = OF$$

evaluates to true then

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.14 jge

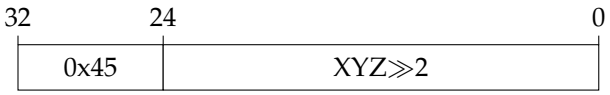
3.14.1 Assembly Notation

jge XYZ

Alternative Assembly Notation

jnl XYZ

Format



Effect

If the condition

$$SF = OF$$

evaluates to true then

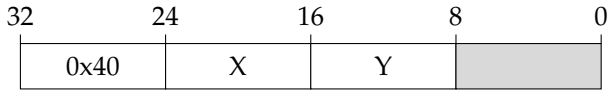
$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.15 jmp

3.15.1 Assembly Notation

jmp %X, %Y

Format



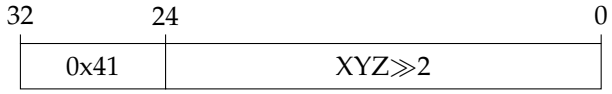
Effect

$$(u(\%IP) + 4) \bmod 2^{64} \rightarrow u(\%Y)$$
$$u(\%X) \rightarrow u(\%IP)$$

3.15.2 Assembly Notation

jmp XYZ

Format



Effect

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.16 jna

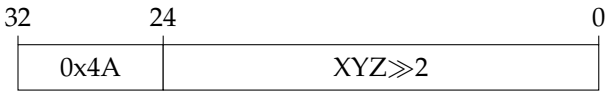
3.16.1 Assembly Notation

jna XYZ

Alternative Assembly Notation

jbe XYZ

Format



Effect

If the condition

$$CF = 1 \vee ZF = 1$$

evaluates to true then

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.17 jne

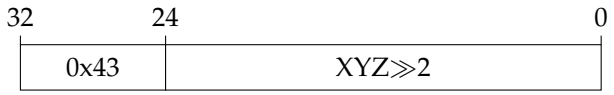
3.17.1 Assembly Notation

jne XYZ

Alternative Assembly Notation

jnz XYZ

Format



Effect

If the condition

$$ZF = 0$$

evaluates to true then

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.18 jng

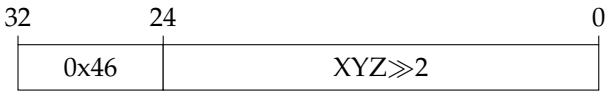
3.18.1 Assembly Notation

jng XYZ

Alternative Assembly Notation

jle XYZ

Format



Effect

If the condition

$$ZF = 1 \vee SF \neq OF$$

evaluates to true then

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.19 jnge

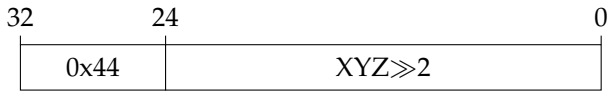
3.19.1 Assembly Notation

jnge XYZ

Alternative Assembly Notation

jl XYZ

Format



Effect

If the condition

$$SF \neq OF$$

evaluates to true then

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

3.20 jz

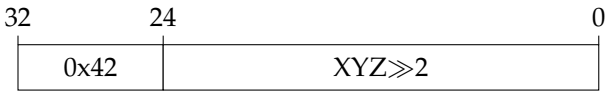
3.20.1 Assembly Notation

jz XYZ

Alternative Assembly Notation

je XYZ

Format



Effect

If the condition

$$ZF = 1$$

evaluates to true then

$$(u(\%IP) + s(XYZ)) \bmod 2^{64} \rightarrow u(\%IP)$$

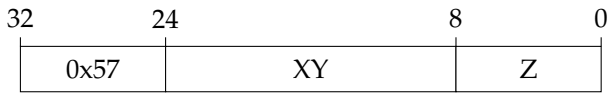
3.21 ldswq

Load a signed word into a register

3.21.1 Assembly Notation

ldswq XY, %Z

Format



Effect

$s(XY) \bmod 2^{64} \rightarrow u\,(\%Z)$

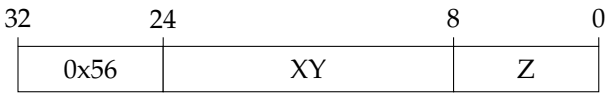
3.22 ldzwq

Load a unsigned word into a register

3.22.1 Assembly Notation

ldzwq XY, %Z

Format



Effect

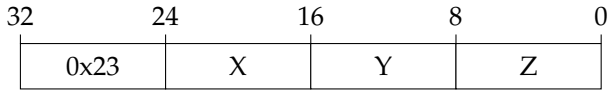
$u(XY) \bmod 2^{64} \rightarrow u\,(\%Z)$

3.23 movb

3.23.1 Assembly Notation

movb %X, (%Y, %Z)

Format



Effect

$$u(\%X) \bmod 2^8 \rightarrow u\left(\mathrm{M}_1\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z)\right) \bmod 2^{64}$$

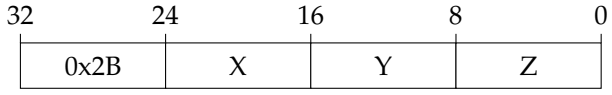
3.23.2 Assembly Notation

movb %X, Y(%Z)

Alternative Assembly Notation

movb %X, (%Z)

Format



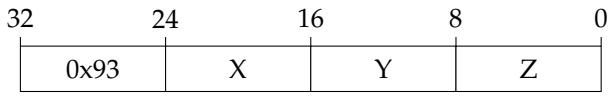
Effect

$$u(\%X) \bmod 2^8 \rightarrow u\left(\mathrm{M}_1\left(addr\right)\right) \text{ with } addr = \left(s(Y) + u(\%Z)\right) \bmod 2^{64}$$

3.23.3 Assembly Notation

movb %X, (%Y, %Z, 2)

Format



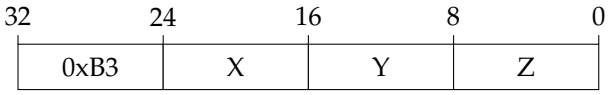
Effect

$$u(\%X) \bmod 2^8 \rightarrow u\left(\mathrm{M}_1\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 2\right) \bmod 2^{64}$$

3.23.4 Assembly Notation

movb %X, (%Y, %Z, 4)

Format



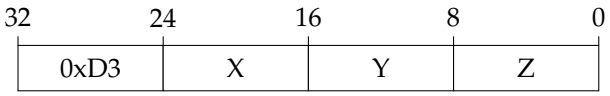
Effect

$$u(\%X) \bmod 2^8 \rightarrow u\left(\mathrm{M}_1\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 4\right) \bmod 2^{64}$$

3.23.5 Assembly Notation

movb %X, (%Y, %Z, 8)

Format



Effect

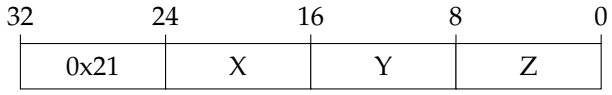
$$u(\%X) \bmod 2^8 \rightarrow u\left(\mathrm{M}_1\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 8\right) \bmod 2^{64}$$

3.24 movl

3.24.1 Assembly Notation

movl %X, (%Y, %Z)

Format



Effect

$$u(\%X) \bmod 2^{32} \rightarrow u\left(\mathrm{M}_4\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z)\right) \bmod 2^{64}$$

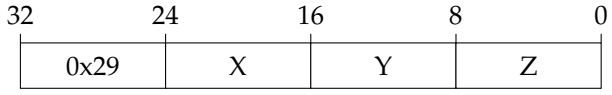
3.24.2 Assembly Notation

movl %X, Y(%Z)

Alternative Assembly Notation

movl %X, (%Z)

Format



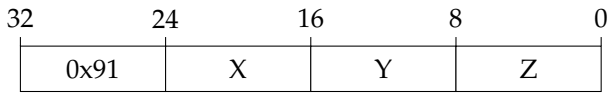
Effect

$$u(\%X) \bmod 2^{32} \rightarrow u\left(\mathrm{M}_4\left(addr\right)\right) \text{ with } addr = \left(s(Y) + u(\%Z)\right) \bmod 2^{64}$$

3.24.3 Assembly Notation

movl %X, (%Y, %Z, 2)

Format



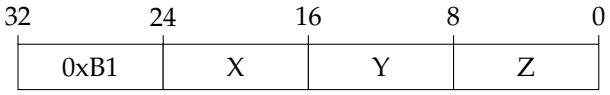
Effect

$$u(\%X) \bmod 2^{32} \rightarrow u\left(\mathrm{M}_4\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 2\right) \bmod 2^{64}$$

3.24.4 Assembly Notation

movl %X, (%Y, %Z, 4)

Format



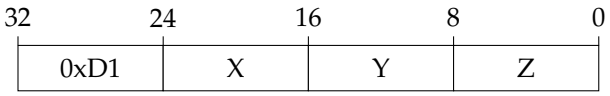
Effect

$$u(\%X) \bmod 2^{32} \rightarrow u\left(\mathrm{M}_4\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 4\right) \bmod 2^{64}$$

3.24.5 Assembly Notation

movl %X, (%Y, %Z, 8)

Format



Effect

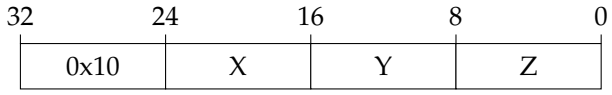
$$u(\%X) \bmod 2^{32} \rightarrow u\left(\mathrm{M}_4\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 8\right) \bmod 2^{64}$$

3.25 movq

3.25.1 Assembly Notation

movq (%X, %Y), %Z

Format



Effect

$$u\left(\mathrm{M}_8\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y)\right) \bmod 2^{64}$$

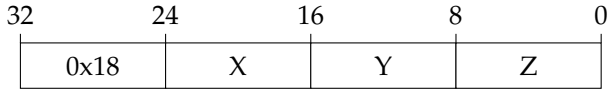
3.25.2 Assembly Notation

movq X(%Y), %Z

Alternative Assembly Notation

movq (%Y), %Z

Format



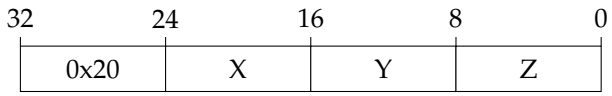
Effect

$$u\left(\mathrm{M}_8\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(s(X) + u(\%Y)\right) \bmod 2^{64}$$

3.25.3 Assembly Notation

movq %X, (%Y, %Z)

Format



Effect

$$u(\%X) \bmod 2^{64} \rightarrow u\left(\mathrm{M}_8\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z)\right) \bmod 2^{64}$$

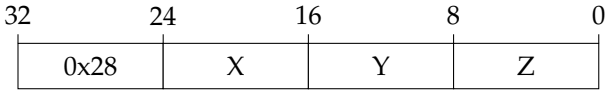
3.25.4 Assembly Notation

movq %X, Y(%Z)

Alternative Assembly Notation

movq %X, (%Z)

Format



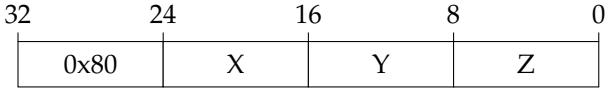
Effect

$$u(\%X) \bmod 2^{64} \rightarrow u\left(\mathrm{M}_8\left(addr\right)\right) \text{ with } addr = \left(s(Y) + u(\%Z)\right) \bmod 2^{64}$$

3.25.5 Assembly Notation

movq (%X, %Y, 2), %Z

Format



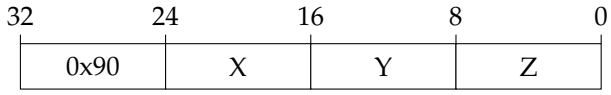
Effect

$$u\left(\mathrm{M}_8\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 2\right) \bmod 2^{64}$$

3.25.6 Assembly Notation

movq %X, (%Y, %Z, 2)

Format



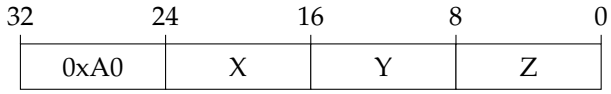
Effect

$$u(\%X) \bmod 2^{64} \rightarrow u\left(\mathrm{M}_8\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 2\right) \bmod 2^{64}$$

3.25.7 Assembly Notation

movq (%X, %Y, 4), %Z

Format



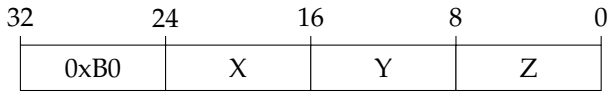
Effect

$$u\left(\mathrm{M}_8\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 4\right) \bmod 2^{64}$$

3.25.8 Assembly Notation

movq %X, (%Y, %Z, 4)

Format



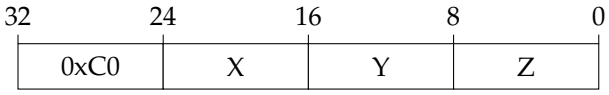
Effect

$$u(\%X) \bmod 2^{64} \rightarrow u\left(\mathrm{M}_8\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 4\right) \bmod 2^{64}$$

3.25.9 Assembly Notation

movq (%X, %Y, 8), %Z

Format



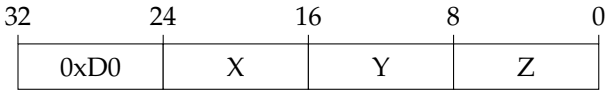
Effect

$u\left(\mathrm{M}_8\left(addr\right)\right) \rightarrow u\left(\%Z\right)$ with $addr = \left(u(\%X) + u(\%Y) \cdot 8\right) \bmod 2^{64}$

3.25.10 Assembly Notation

movq %X, (%Y, %Z, 8)

Format



Effect

$u(\%X) \bmod 2^{64} \rightarrow u\left(\mathrm{M}_8\left(addr\right)\right)$ with $addr = \left(u(\%Y) + u(\%Z) \cdot 8\right) \bmod 2^{64}$

3.26 movsbq

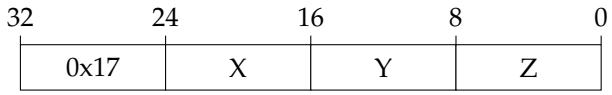
3.26.1 Assembly Notation

movsbq (%X, %Y), %Z

Alternative Assembly Notation

movsbq (%X, %Y, 1), %Z

Format



Effect

$$s\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y)\right) \bmod 2^{64}$$

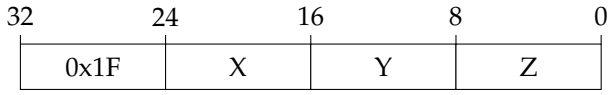
3.26.2 Assembly Notation

movsbq (%Y), %Z

Alternative Assembly Notation

movsbq X(%Y), %Z

Format



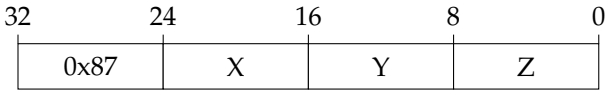
Effect

$$s\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(s(X) + u(\%Y)\right) \bmod 2^{64}$$

3.26.3 Assembly Notation

movsbq (%X, %Y, 2), %Z

Format



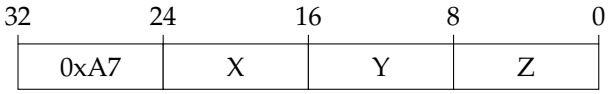
Effect

$$s\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%\mathrm{Z}\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 2\right) \bmod 2^{64}$$

3.26.4 Assembly Notation

movsbq (%X, %Y, 4), %Z

Format



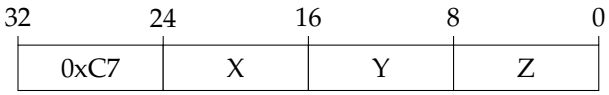
Effect

$$s\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%\mathrm{Z}\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 4\right) \bmod 2^{64}$$

3.26.5 Assembly Notation

movsbq (%X, %Y, 8), %Z

Format



Effect

$$s\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%\mathrm{Z}\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 8\right) \bmod 2^{64}$$

3.27 movslq

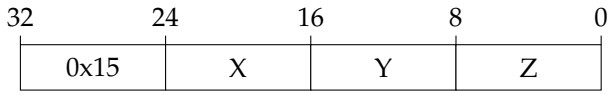
3.27.1 Assembly Notation

movslq (%X, %Y, 1), %Z

Alternative Assembly Notation

movslq (%X, %Y), %Z

Format



Effect

$$s\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y)\right) \bmod 2^{64}$$

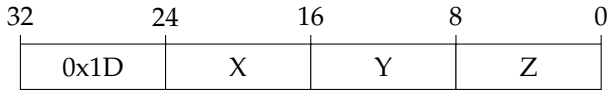
3.27.2 Assembly Notation

movslq X(%Y), %Z

Alternative Assembly Notation

movslq (%Y), %Z

Format



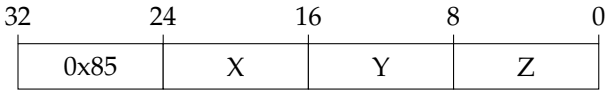
Effect

$$s\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(s(X) + u(\%Y)\right) \bmod 2^{64}$$

3.27.3 Assembly Notation

movslq (%X, %Y, 2), %Z

Format



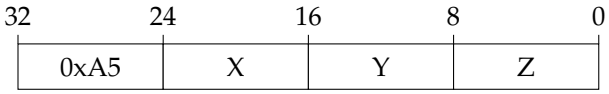
Effect

$$s\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%\mathrm{Z}\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 2\right) \bmod 2^{64}$$

3.27.4 Assembly Notation

movslq (%X, %Y, 4), %Z

Format



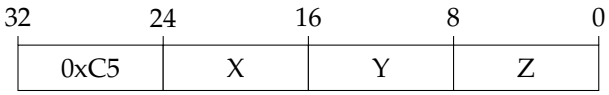
Effect

$$s\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%\mathrm{Z}\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 4\right) \bmod 2^{64}$$

3.27.5 Assembly Notation

movslq (%X, %Y, 8), %Z

Format



Effect

$$s\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%\mathrm{Z}\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 8\right) \bmod 2^{64}$$

3.28 movswq

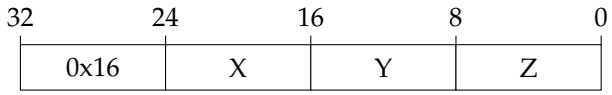
3.28.1 Assembly Notation

movswq (%X, %Y, 1), %Z

Alternative Assembly Notation

movswq (%X, %Y), %Z

Format



Effect

$$s\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y)\right) \bmod 2^{64}$$

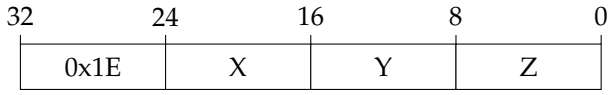
3.28.2 Assembly Notation

movswq (%Y), %Z

Alternative Assembly Notation

movswq X(%Y), %Z

Format



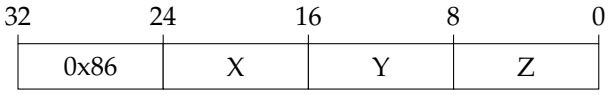
Effect

$$s\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(s(X) + u(\%Y)\right) \bmod 2^{64}$$

3.28.3 Assembly Notation

movswq (%X, %Y, 2), %Z

Format



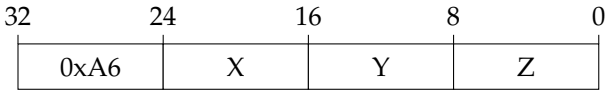
Effect

$$s\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%\mathrm{Z}\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 2\right) \bmod 2^{64}$$

3.28.4 Assembly Notation

movswq (%X, %Y, 4), %Z

Format



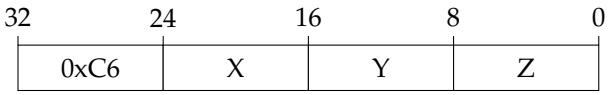
Effect

$$s\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%\mathrm{Z}\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 4\right) \bmod 2^{64}$$

3.28.5 Assembly Notation

movswq (%X, %Y, 8), %Z

Format



Effect

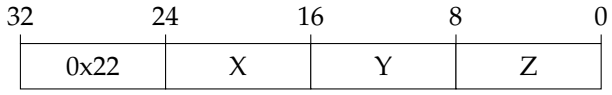
$$s\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%\mathrm{Z}\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 8\right) \bmod 2^{64}$$

3.29 movw

3.29.1 Assembly Notation

movw %X, (%Y, %Z)

Format



Effect

$$u(\%X) \bmod 2^{16} \rightarrow u\left(\mathrm{M}_2\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z)\right) \bmod 2^{64}$$

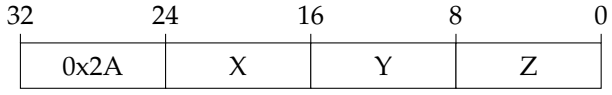
3.29.2 Assembly Notation

movw %X, Y(%Z)

Alternative Assembly Notation

movw %X, (%Z)

Format



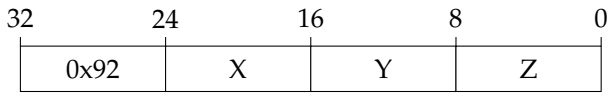
Effect

$$u(\%X) \bmod 2^{16} \rightarrow u\left(\mathrm{M}_2\left(addr\right)\right) \text{ with } addr = \left(s(Y) + u(\%Z)\right) \bmod 2^{64}$$

3.29.3 Assembly Notation

movw %X, (%Y, %Z, 2)

Format



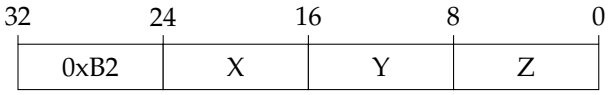
Effect

$$u(\%X) \bmod 2^{16} \rightarrow u\left(\mathrm{M}_2\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 2\right) \bmod 2^{64}$$

3.29.4 Assembly Notation

movw %X, (%Y, %Z, 4)

Format



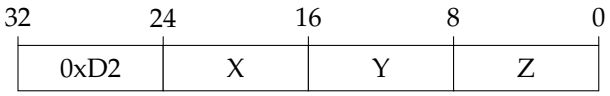
Effect

$$u(\%X) \bmod 2^{16} \rightarrow u\left(\mathrm{M}_2\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 4\right) \bmod 2^{64}$$

3.29.5 Assembly Notation

movw %X, (%Y, %Z, 8)

Format



Effect

$$u(\%X) \bmod 2^{16} \rightarrow u\left(\mathrm{M}_2\left(addr\right)\right) \text{ with } addr = \left(u(\%Y) + u(\%Z) \cdot 8\right) \bmod 2^{64}$$

3.30 movzbq

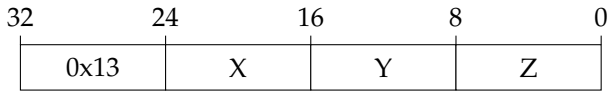
3.30.1 Assembly Notation

movzbq (%X, %Y, 1), %Z

Alternative Assembly Notation

movzbq (%X, %Y), %Z

Format



Effect

$u\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%Z\right)$ with $addr = \left(u(\%X) + u(\%Y)\right) \bmod 2^{64}$

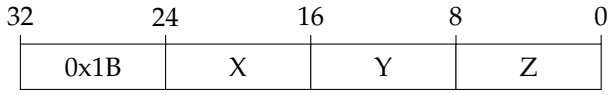
3.30.2 Assembly Notation

movzbq (%Y), %Z

Alternative Assembly Notation

movzbq X(%Y), %Z

Format



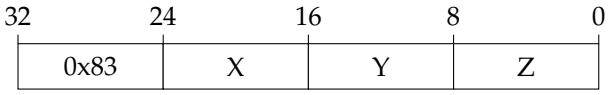
Effect

$u\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%Z\right)$ with $addr = \left(s(X) + u(\%Y)\right) \bmod 2^{64}$

3.30.3 Assembly Notation

movzbq (%X, %Y, 2), %Z

Format



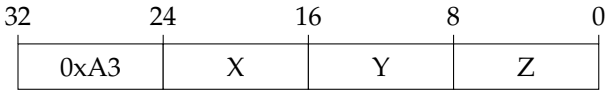
Effect

$$u\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 2\right) \bmod 2^{64}$$

3.30.4 Assembly Notation

movzbq (%X, %Y, 4), %Z

Format



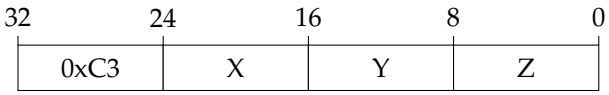
Effect

$$u\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 4\right) \bmod 2^{64}$$

3.30.5 Assembly Notation

movzbq (%X, %Y, 8), %Z

Format



Effect

$$u\left(\mathrm{M}_1\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 8\right) \bmod 2^{64}$$

3.31 movzlq

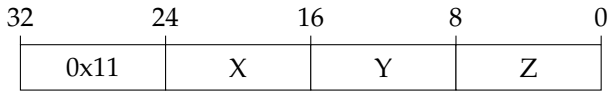
3.31.1 Assembly Notation

movzlq (%X, %Y, 1), %Z

Alternative Assembly Notation

movzlq (%X, %Y), %Z

Format



Effect

$$u\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y)\right) \bmod 2^{64}$$

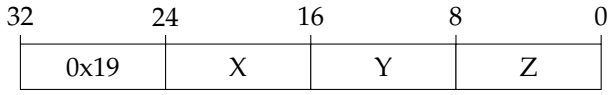
3.31.2 Assembly Notation

movzlq X(%Y), %Z

Alternative Assembly Notation

movzlq (%Y), %Z

Format



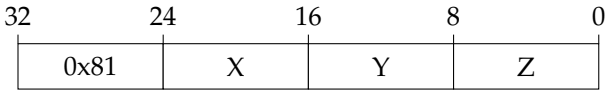
Effect

$$u\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(s(X) + u(\%Y)\right) \bmod 2^{64}$$

3.31.3 Assembly Notation

movzlq (%X, %Y, 2), %Z

Format



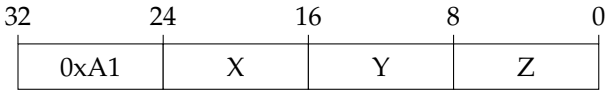
Effect

$$u\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 2\right) \bmod 2^{64}$$

3.31.4 Assembly Notation

movzlq (%X, %Y, 4), %Z

Format



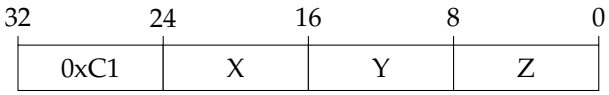
Effect

$$u\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 4\right) \bmod 2^{64}$$

3.31.5 Assembly Notation

movzlq (%X, %Y, 8), %Z

Format



Effect

$$u\left(\mathrm{M}_4\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 8\right) \bmod 2^{64}$$

3.32 movzwq

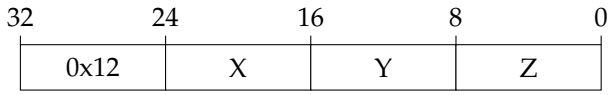
3.32.1 Assembly Notation

movzwq (%X, %Y, 1), %Z

Alternative Assembly Notation

movzwq (%X, %Y), %Z

Format



Effect

$$u\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y)\right) \bmod 2^{64}$$

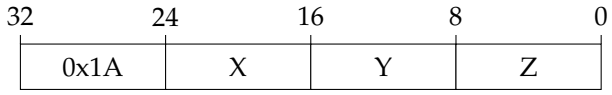
3.32.2 Assembly Notation

movzwq X(%Y), %Z

Alternative Assembly Notation

movzwq (%Y), %Z

Format



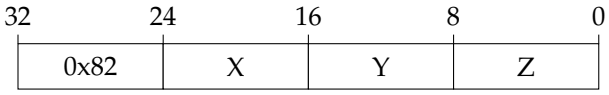
Effect

$$u\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(s(X) + u(\%Y)\right) \bmod 2^{64}$$

3.32.3 Assembly Notation

movzwq (%X, %Y, 2), %Z

Format



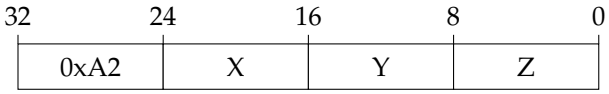
Effect

$$u\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 2\right) \bmod 2^{64}$$

3.32.4 Assembly Notation

movzwq (%X, %Y, 4), %Z

Format



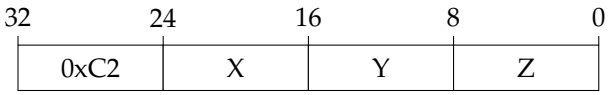
Effect

$$u\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 4\right) \bmod 2^{64}$$

3.32.5 Assembly Notation

movzwq (%X, %Y, 8), %Z

Format



Effect

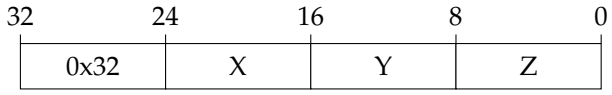
$$u\left(\mathrm{M}_2\left(addr\right)\right) \rightarrow u\left(\%Z\right) \text{ with } addr = \left(u(\%X) + u(\%Y) \cdot 8\right) \bmod 2^{64}$$

3.33 mulq

3.33.1 Assembly Notation

mulq %X, %Y, %Z

Format



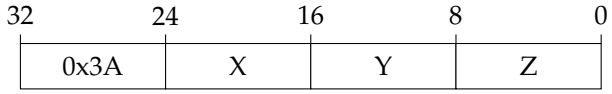
Effect

TODO: ulm_mul128

3.33.2 Assembly Notation

mulq X, %Y, %Z

Format



Effect

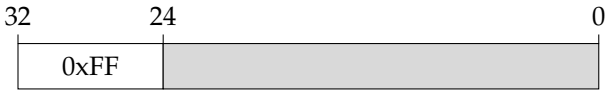
TODO: ulm_mul128

3.34 nop

3.34.1 Assembly Notation

nop

Format



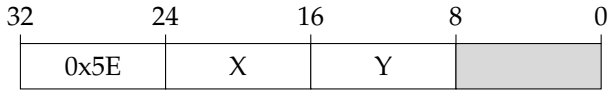
Effect

3.35 notq

3.35.1 Assembly Notation

notq %X, %Y

Format



Effect

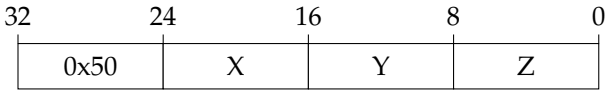
TODO: ulm_not64

3.36 orq

3.36.1 Assembly Notation

orq %X, %Y, %Z

Format



Effect

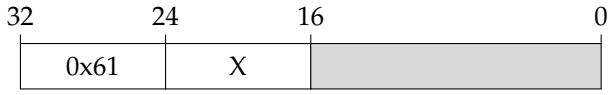
TODO: ulm_or64

3.37 putc

3.37.1 Assembly Notation

putc %X

Format



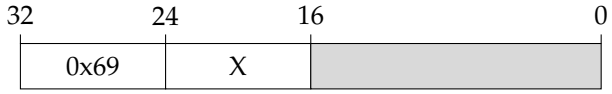
Effect

ulm_printChar(%X)

3.37.2 Assembly Notation

putc X

Format



Effect

ulm_printChar(X)

3.38 salq

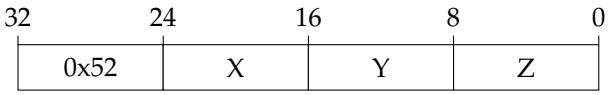
3.38.1 Assembly Notation

salq %X, %Y, %Z

Alternative Assembly Notation

shlq %X, %Y, %Z

Format



Effect

TODO: ulm_shiftLeft64

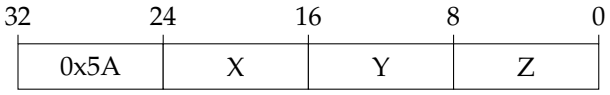
3.38.2 Assembly Notation

salq X, %Y, %Z

Alternative Assembly Notation

shlq X, %Y, %Z

Format



Effect

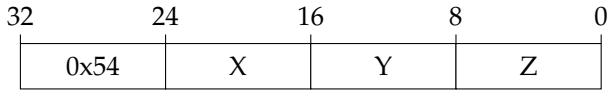
TODO: ulm_shiftLeft64

3.39 sarq

3.39.1 Assembly Notation

sarq %X, %Y, %Z

Format



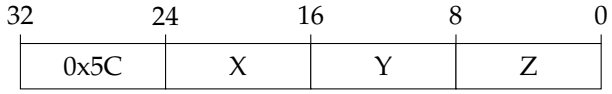
Effect

TODO: ulm_shiftRightSigned64

3.39.2 Assembly Notation

sarq X, %Y, %Z

Format



Effect

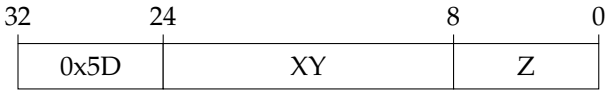
TODO: ulm_shiftRightSigned64

3.40 shldwq

3.40.1 Assembly Notation

shldwq XY, %Z

Format



Effect

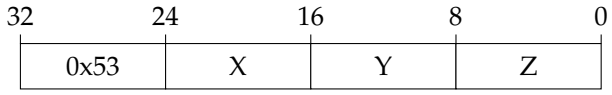
$$u(\%Z) << 16 | u(XY) \bmod 2^{64} \rightarrow u\,(\%Z)$$

3.41 shrq

3.41.1 Assembly Notation

shrq %X, %Y, %Z

Format



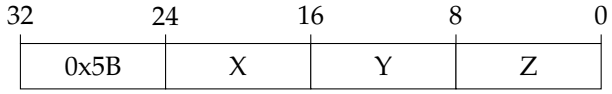
Effect

TODO: ulm_shiftRightUnsigned64

3.41.2 Assembly Notation

shrq X, %Y, %Z

Format



Effect

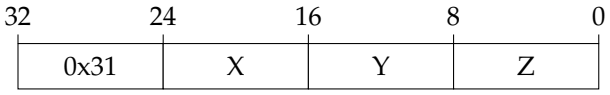
TODO: ulm_shiftRightUnsigned64

3.42 subq

3.42.1 Assembly Notation

subq %X, %Y, %Z

Format



Effect

$$(u(\%Y) - u(\%X)) \bmod 2^{64} \rightarrow u\,(\%Z)$$

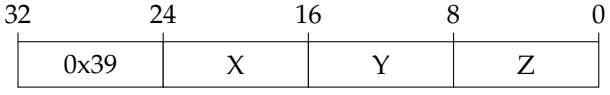
Updates the status flags:

Flag	Condition
ZF	$u(\%Y) - u(\%X) = 0$
CF	$u(\%Y) - u(\%X) < 0$
OF	$s(\%Y) - s(\%X) \notin \{-2^{63}, \ldots, 2^{63} - 1\}$
SF	$s(\%Y) - s(\%X) < 0$

3.42.2 Assembly Notation

subq X, %Y, %Z

Format



Effect

$$(u(\%Y) - u(X)) \bmod 2^{64} \rightarrow u\,(\%Z)$$

Updates the status flags:

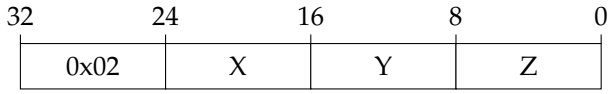
Flag	Condition
ZF	$u(\%Y) - u(X) = 0$
CF	$u(\%Y) - u(X) < 0$
OF	$s(\%Y) - s(X) \notin \{-2^{63}, \ldots, 2^{63} - 1\}$
SF	$s(\%Y) - s(X) < 0$

3.43 trap

3.43.1 Assembly Notation

trap %X, %Y, %Z

Format



Effect

TODO: ulm_trap

Chapter 4

ISA Source File for the ULM Generator

```
# Description of the hardware
# NUM_REGISTERS: 256

# For the description of the assembly notation
# LPAREN_MEM: (
# REGISTER_TOKEN: %

#
# Fmt {(Field Type Size)}
#
# First Field is used as opCode and is required to have some fixed size and type
#
E (OP u 8)
J26 (OP u 8) (XYZ j 24)
S16R (OP u 8) (XY s 16) (Z u 8)
U16R (OP u 8) (XY u 16) (Z u 8)
R (OP u 8) (X u 8)
U (OP u 8) (X u 8)
RR (OP u 8) (X u 8) (Y u 8)
RRR (OP u 8) (X u 8) (Y u 8) (Z u 8)
SRR (OP u 8) (X s 8) (Y u 8) (Z u 8)
URR (OP u 8) (X u 8) (Y u 8) (Z u 8)
R_MRR (OP u 8) (X u 8) (Y u 8) (Z u 8)
R_MSR (OP u 8) (X u 8) (Y s 8) (Z u 8)
MRR_R (OP u 8) (X u 8) (Y u 8) (Z u 8)
MSR_R (OP u 8) (X s 8) (Y u 8) (Z u 8)

#
# Opcode [Fmt]
# : Assembly notatation
# Code block

0x06 RRR
: hm3Add X, %Y, %Z
      ulm_add64(X*3, ulm_regVal(Y), Z);
```

```
36
37  #-------------------------------------------------------------------------
38
39  0x30 RRR
40  # Adds register X with register Y. Stores the result in register Z.
41  : addq %X, %Y, %Z
42        ulm_add64(ulm_regVal(X), ulm_regVal(Y), Z);
43
44  0x38 URR
45  # Adds the unsigned immediate value X with the unsigned integer %Y. Stores
46  # the result in %Z. Updates the status flags.
47  : addq X, %Y, %Z
48        ulm_add64(X, ulm_regVal(Y), Z);
49
50
51  @addq
52  # Integer addition.
53  # Adds either the content of register X or the unsigend immediate value X to
54  # register Y. The result is stored in register Z.
55
56  #-------------------------------------------------------------------------
57
58  0x37 RRR
59  : adcq %X, %Y, %Z
60        ulm_add64(ulm_regVal(X) + ulm_statusReg[ULM_ZF], ulm_regVal(Y), Z);
61
62  #-------------------------------------------------------------------------
63
64  0x51 RRR
65  : andq %X, %Y, %Z
66        ulm_and64(ulm_regVal(X), ulm_regVal(Y), Z);
67
68  #-------------------------------------------------------------------------
69
70  0x33 RRR
71  : divq %X, %Y, %Z
72        ulm_div128(ulm_regVal(X), ulm_regVal(Y), ulm_regVal(Y + 1),
73                   Z, Z + 1, Z + 2);
74  0x3B URR
75  : divq X, %Y, %Z
76        ulm_div128(X, ulm_regVal(Y), ulm_regVal(Y + 1), Z, Z + 1, Z + 2);
77
78  #-------------------------------------------------------------------------
79
80  0x60 R
81  : getc %X
82        ulm_setReg(ulm_readChar() & 0xFF, X);
83
```

```
84  #-------------------------------------------------------------------------------
85
86  0x01 R
87  : halt %X
88        ulm_halt(ulm_regVal(X));
89  0x09 U
90  : halt X
91        ulm_halt(X);
92
93  #-------------------------------------------------------------------------------
94
95  0x35 RRR
96  : idivq %X, %Y, %Z
97        ulm_idiv64(ulm_regVal(X), ulm_regVal(Y), Z);
98  0x3D URR
99  : idivq X, %Y, %Z
100       ulm_idiv64(X, ulm_regVal(Y), Z);
101
102 #-------------------------------------------------------------------------------
103
104 0x34 RRR
105 : imulq %X, %Y, %Z
106       ulm_mul64(ulm_regVal(X), ulm_regVal(Y), Z);
107 0x3C SRR
108 : imulq X, %Y, %Z
109       ulm_mul64(X, ulm_regVal(Y), Z);
110
111 #-------------------------------------------------------------------------------
112
113 0x4B J26
114 : ja XYZ
115 : jnbe XYZ
116       ulm_conditionalRelJump(ulm_statusReg[ULM_CF] == 0 &&
117                              ulm_statusReg[ULM_ZF] == 0,
118                              XYZ);
119 #-------------------------------------------------------------------------------
120
121 0x49 J26
122 : jae XYZ
123 : jnb XYZ
124       ulm_conditionalRelJump(ulm_statusReg[ULM_CF] == 0, XYZ);
125
126 #-------------------------------------------------------------------------------
127
128 0x48 J26
129 : jb XYZ
130 : jnae XYZ
```

```
131     ulm_conditionalRelJump(ulm_statusReg[ULM_CF] == 1, XYZ);
132
133 #-------------------------------------------------------------------------------
134
135 0x4A J26
136 : jna XYZ
137 : jbe XYZ
138     ulm_conditionalRelJump(ulm_statusReg[ULM_CF] == 1 ||
139                            ulm_statusReg[ULM_ZF] == 1,
140                            XYZ);
141
142 #-------------------------------------------------------------------------------
143
144 0x42 J26
145 : jz XYZ
146 : je XYZ
147     ulm_conditionalRelJump(ulm_statusReg[ULM_ZF] == 1, XYZ);
148
149 #-------------------------------------------------------------------------------
150
151 0x47 J26
152 : jg XYZ
153 : jnle XYZ
154     ulm_conditionalRelJump(ulm_statusReg[ULM_ZF] == 0 &&
155                            ulm_statusReg[ULM_SF] == ulm_statusReg[ULM_OF],
156                            XYZ);
157
158 #-------------------------------------------------------------------------------
159
160 0x45 J26
161 : jge XYZ
162 : jnl XYZ
163     ulm_conditionalRelJump(ulm_statusReg[ULM_SF] == ulm_statusReg[ULM_OF],
164                            XYZ);
165
166 #-------------------------------------------------------------------------------
167
168 0x44 J26
169 : jnge XYZ
170 : jl XYZ
171     ulm_conditionalRelJump(ulm_statusReg[ULM_SF] != ulm_statusReg[ULM_OF],
172                            XYZ);
173
174 #-------------------------------------------------------------------------------
175
176 0x46 J26
177 : jng XYZ
```

```
: jle XYZ
    ulm_conditionalRelJump(ulm_statusReg[ULM_ZF] == 1 ||
                           ulm_statusReg[ULM_SF] != ulm_statusReg[ULM_OF],
                           XYZ);

#-------------------------------------------------------------------------------


0x40 RR
: jmp %X, %Y
    ulm_absJump(ulm_regVal(X), Y);
0x41 J26
: jmp XYZ
    ulm_unconditionalRelJump(XYZ);

#-------------------------------------------------------------------------------


0x43 J26
: jne XYZ
: jnz XYZ
    ulm_conditionalRelJump(ulm_statusReg[ULM_ZF] == 0, XYZ);

#-------------------------------------------------------------------------------


0x57 S16R
: ldswq XY, %Z
    ulm_setReg(XY, Z);

@ldswq
# Load a signed word into a register

#-------------------------------------------------------------------------------


0x56 U16R
: ldzwq XY, %Z
    ulm_setReg(XY, Z);

@ldzwq
# Load a unsigned word into a register

#-------------------------------------------------------------------------------


0x23 R_MRR
: movb %X, (%Y, %Z)
    ulm_store64(0, Y, Z, 1, 1, X);

0x2B R_MSR
: movb %X, Y(%Z)
: movb %X, (%Z)
```

```
226         ulm_store64(Y, Z, 0, 1, 1, X);
227
228     0x93 R_MRR
229     : movb %X, (%Y, %Z, 2)
230         ulm_store64(0, Y, Z, 2, 1, X);
231
232     0xB3 R_MRR
233     : movb %X, (%Y, %Z, 4)
234         ulm_store64(0, Y, Z, 4, 1, X);
235
236     0xD3 R_MRR
237     : movb %X, (%Y, %Z, 8)
238         ulm_store64(0, Y, Z, 8, 1, X);
239
240     0x21 R_MRR
241     : movl %X, (%Y, %Z)
242         ulm_store64(0, Y, Z, 1, 4, X);
243
244     0x29 R_MSR
245     : movl %X, Y(%Z)
246     : movl %X, (%Z)
247         ulm_store64(Y, Z, 0, 1, 4, X);
248
249     0x91 R_MRR
250     : movl %X, (%Y, %Z, 2)
251         ulm_store64(0, Y, Z, 2, 4, X);
252
253     0xB1 R_MRR
254     : movl %X, (%Y, %Z, 4)
255         ulm_store64(0, Y, Z, 4, 4, X);
256
257     0xD1 R_MRR
258     : movl %X, (%Y, %Z, 8)
259         ulm_store64(0, Y, Z, 8, 4, X);
260
261     0x10 MRR_R
262     : movq (%X, %Y), %Z
263         ulm_fetch64(0, X, Y, 1, ULM_ZERO_EXT, 8, Z);
264
265     0x18 MSR_R
266     : movq X(%Y), %Z
267     : movq (%Y), %Z
268         ulm_fetch64(X, Y, 0, 1, ULM_ZERO_EXT, 8, Z);
269
270     0x20 R_MRR
271     : movq %X, (%Y, %Z)
272         ulm_store64(0, Y, Z, 1, 8, X);
273
274     0x28 R_MSR
275     : movq %X, Y(%Z)
276     : movq %X, (%Z)
277         ulm_store64(Y, Z, 0, 1, 8, X);
278
```

```
0x80 MRR_R
: movq (%X, %Y, 2), %Z
      ulm_fetch64(0, X, Y, 2, ULM_ZERO_EXT, 8, Z);

0x90 R_MRR
: movq %X, (%Y, %Z, 2)
      ulm_store64(0, Y, Z, 2, 8, X);

0xA0 MRR_R
: movq (%X, %Y, 4), %Z
      ulm_fetch64(0, X, Y, 4, ULM_ZERO_EXT, 8, Z);

0xB0 R_MRR
: movq %X, (%Y, %Z, 4)
      ulm_store64(0, Y, Z, 4, 8, X);

0xC0 MRR_R
: movq (%X, %Y, 8), %Z
      ulm_fetch64(0, X, Y, 8, ULM_ZERO_EXT, 8, Z);

0xD0 R_MRR
: movq %X, (%Y, %Z, 8)
      ulm_store64(0, Y, Z, 8, 8, X);

0x17 MRR_R
: movsbq (%X, %Y), %Z
: movsbq (%X, %Y, 1), %Z
      ulm_fetch64(0, X, Y, 1, ULM_SIGN_EXT, 1, Z);

0x1F MSR_R
: movsbq (%Y), %Z
: movsbq X(%Y), %Z
      ulm_fetch64(X, Y, 0, 1, ULM_SIGN_EXT, 1, Z);

0x87 MRR_R
: movsbq (%X, %Y, 2), %Z
      ulm_fetch64(0, X, Y, 2, ULM_SIGN_EXT, 1, Z);

0xA7 MRR_R
: movsbq (%X, %Y, 4), %Z
      ulm_fetch64(0, X, Y, 4, ULM_SIGN_EXT, 1, Z);

0xC7 MRR_R
: movsbq (%X, %Y, 8), %Z
      ulm_fetch64(0, X, Y, 8, ULM_SIGN_EXT, 1, Z);

0x15 MRR_R
: movslq (%X, %Y, 1), %Z
: movslq (%X, %Y), %Z
      ulm_fetch64(0, X, Y, 1, ULM_SIGN_EXT, 4, Z);

0x1D MSR_R
: movslq X(%Y), %Z
```

```
: movslq (%Y), %Z
        ulm_fetch64(X, Y, 0, 1, ULM_SIGN_EXT, 4, Z);

0x85 MRR_R
: movslq (%X, %Y, 2), %Z
        ulm_fetch64(0, X, Y, 2, ULM_SIGN_EXT, 4, Z);

0xA5 MRR_R
: movslq (%X, %Y, 4), %Z
        ulm_fetch64(0, X, Y, 4, ULM_SIGN_EXT, 4, Z);

0xC5 MRR_R
: movslq (%X, %Y, 8), %Z
        ulm_fetch64(0, X, Y, 8, ULM_SIGN_EXT, 4, Z);

0x16 MRR_R
: movswq (%X, %Y, 1), %Z
: movswq (%X, %Y), %Z
        ulm_fetch64(0, X, Y, 1, ULM_SIGN_EXT, 2, Z);

0x1E MSR_R
: movswq (%Y), %Z
: movswq X(%Y), %Z
        ulm_fetch64(X, Y, 0, 1, ULM_SIGN_EXT, 2, Z);

0x86 MRR_R
: movswq (%X, %Y, 2), %Z
        ulm_fetch64(0, X, Y, 2, ULM_SIGN_EXT, 2, Z);

0xA6 MRR_R
: movswq (%X, %Y, 4), %Z
        ulm_fetch64(0, X, Y, 4, ULM_SIGN_EXT, 2, Z);

0xC6 MRR_R
: movswq (%X, %Y, 8), %Z
        ulm_fetch64(0, X, Y, 8, ULM_SIGN_EXT, 2, Z);


0x22 R_MRR
: movw %X, (%Y, %Z)
        ulm_store64(0, Y, Z, 1, 2, X);

0x2A R_MSR
: movw %X, Y(%Z)
: movw %X, (%Z)
        ulm_store64(Y, Z, 0, 1, 2, X);

0x92 R_MRR
: movw %X, (%Y, %Z, 2)
        ulm_store64(0, Y, Z, 2, 2, X);

0xB2 R_MRR
: movw %X, (%Y, %Z, 4)
```

```
385         ulm_store64(0, Y, Z, 4, 2, X);
386
387     0xD2 R_MRR
388     : movw %X, (%Y, %Z, 8)
389         ulm_store64(0, Y, Z, 8, 2, X);
390
391     0x13 MRR_R
392     : movzbq (%X, %Y, 1), %Z
393     : movzbq (%X, %Y), %Z
394         ulm_fetch64(0, X, Y, 1, ULM_ZERO_EXT, 1, Z);
395
396     0x1B MSR_R
397     : movzbq (%Y), %Z
398     : movzbq X(%Y), %Z
399         ulm_fetch64(X, Y, 0, 1, ULM_ZERO_EXT, 1, Z);
400
401     0x83 MRR_R
402     : movzbq (%X, %Y, 2), %Z
403         ulm_fetch64(0, X, Y, 2, ULM_ZERO_EXT, 1, Z);
404
405     0xA3 MRR_R
406     : movzbq (%X, %Y, 4), %Z
407         ulm_fetch64(0, X, Y, 4, ULM_ZERO_EXT, 1, Z);
408
409     0xC3 MRR_R
410     : movzbq (%X, %Y, 8), %Z
411         ulm_fetch64(0, X, Y, 8, ULM_ZERO_EXT, 1, Z);
412
413     0x11 MRR_R
414     : movzlq (%X, %Y, 1), %Z
415     : movzlq (%X, %Y), %Z
416         ulm_fetch64(0, X, Y, 1, ULM_ZERO_EXT, 4, Z);
417
418     0x19 MSR_R
419     : movzlq X(%Y), %Z
420     : movzlq (%Y), %Z
421         ulm_fetch64(X, Y, 0, 1, ULM_ZERO_EXT, 4, Z);
422
423     0x81 MRR_R
424     : movzlq (%X, %Y, 2), %Z
425         ulm_fetch64(0, X, Y, 2, ULM_ZERO_EXT, 4, Z);
426
427     0xA1 MRR_R
428     : movzlq (%X, %Y, 4), %Z
429         ulm_fetch64(0, X, Y, 4, ULM_ZERO_EXT, 4, Z);
430
431     0xC1 MRR_R
432     : movzlq (%X, %Y, 8), %Z
433         ulm_fetch64(0, X, Y, 8, ULM_ZERO_EXT, 4, Z);
434
435     0x12 MRR_R
436     : movzwq (%X, %Y, 1), %Z
437     : movzwq (%X, %Y), %Z
```

```
438     ulm_fetch64(0, X, Y, 1, ULM_ZERO_EXT, 2, Z);
439
440 0x1A MSR_R
441 : movzwq X(%Y), %Z
442 : movzwq (%Y), %Z
443     ulm_fetch64(X, Y, 0, 1, ULM_ZERO_EXT, 2, Z);
444
445 0x82 MRR_R
446 : movzwq (%X, %Y, 2), %Z
447     ulm_fetch64(0, X, Y, 2, ULM_ZERO_EXT, 2, Z);
448
449 0xA2 MRR_R
450 : movzwq (%X, %Y, 4), %Z
451     ulm_fetch64(0, X, Y, 4, ULM_ZERO_EXT, 2, Z);
452
453 0xC2 MRR_R
454 : movzwq (%X, %Y, 8), %Z
455     ulm_fetch64(0, X, Y, 8, ULM_ZERO_EXT, 2, Z);
456
457 0x32 RRR
458 : mulq %X, %Y, %Z
459     ulm_mul128(ulm_regVal(X), ulm_regVal(Y), Z, Z + 1);
460
461 0x3A URR
462 : mulq X, %Y, %Z
463     ulm_mul128(X, ulm_regVal(Y), Z, Z + 1);
464
465 0xFF E
466 : nop
467     /* nop */;
468
469 0x5E RR
470 : notq %X, %Y
471     ulm_not64(ulm_regVal(X), Y);
472
473 0x50 RRR
474 : orq %X, %Y, %Z
475     ulm_or64(ulm_regVal(X), ulm_regVal(Y), Z);
476
477 0x61 R
478 : putc %X
479     ulm_printChar(ulm_regVal(X));
480
481 0x69 R
482 : putc X
483     ulm_printChar(X);
484
485 0x52 RRR
486 : salq %X, %Y, %Z
487 : shlq %X, %Y, %Z
488     ulm_shiftLeft64(ulm_regVal(X), ulm_regVal(Y), Z);
489
490 0x5A RRR
```

```
: salq X, %Y, %Z
: shlq X, %Y, %Z
    ulm_shiftLeft64(X, ulm_regVal(Y), Z);

0x54 RRR
: sarq %X, %Y, %Z
    ulm_shiftRightSigned64(ulm_regVal(X), ulm_regVal(Y), Z);

0x5C URR
: sarq X, %Y, %Z
    ulm_shiftRightSigned64(X, ulm_regVal(Y), Z);

0x5D U16R
: shldwq XY, %Z
    ulm_setReg(ulm_regVal(Z) << 16 | XY, Z);

0x53 RRR
: shrq %X, %Y, %Z
    ulm_shiftRightUnsigned64(ulm_regVal(X), ulm_regVal(Y), Z);

0x5B URR
: shrq X, %Y, %Z
    ulm_shiftRightUnsigned64(X, ulm_regVal(Y), Z);

0x31 RRR
: subq %X, %Y, %Z
    ulm_sub64(ulm_regVal(X), ulm_regVal(Y), Z);

0x39 URR
: subq X, %Y, %Z
    ulm_sub64(X, ulm_regVal(Y), Z);

0x02 RRR
: trap %X, %Y, %Z
    ulm_trap(X, Y, Z);
```