Einheitlicher Sammenstand Elektronischer Rechengeräte

Specification

Foreword

This document is a translation of the original texts in both German and Ard English, which should be provided with this document, but solely for reference purposes. The texts were originally licensed under the Reichgemeinnutzerlaubnis (Reich Public Use Permissions) and the Mean Need Leave (General Usage License), which are equivalent to CC-BY-ND; they were later relicensed under the Reichgemeinnutz- und -verbreitungserlaubnis (Reich Public Use and Distribution Permissions) and the Mean Need and Spread Leave (General Usage and Distribution License), which are slightly more restrictive than CC-BY-SA, but allow translations.

Contents

Introduction

The OSER is an architecture for systems designed to facilitate data processing and exchange. An OSER system instance is made up of **primary memory**, one or more **processing units** and naught or more **dependent units**.

In the OSER, data processing and exchange occurs in **programs**. When a program is run, three main processes happen:

- access;
- interruption; and
- execution.

Access

In an access, a **datum** is transferred between a unit and primary memory.

Interruption

In an interruption, a unit causes a processing unit to run a new program.

Execution

In execution, a processing unit runs a program.

Part I

Access

Chapter 1

Primary Memory

Primary memory is made of **cells**.

Bitstrings

The cells hold **bitstrings**. In a bitstring, the bits are numbered from 1 and may take either 0 or 1 as values. A string of *n* bits is called an ***n*-string**.

Each cell in primary memory holds a single 8-string. A string longer than 8 bits is held by cutting it into 8-strings and holding the 8-strings in consecutive cells.

8-String

1 8
1

16-String

1 8	9 16
1	2

32-String

1 8	9 16	17 24	25 32
1	2	3	4

64-String

1 8	9 16	17 24	25 32	33 40	41 48	49 56	57 64
1	2	3	4	5	6	7	8

128-String

1 8	9 16	17 24	25 32	33 40	41 48	49 56	57 64
1	2	3	4	5	6	7	8
65 72	73 80	81 88	89 96	97 104	105 112	113 120	121 128
9	10	11	12	13	14	15	16

256-String

1 8	9 16	17 24	25 32	33 40	41 48	49 56	57 64
1	2	3	4	5	6	7	8
65 72	73 80	81 88	89 96	97 104	105 112	113 120	121 128
9	10	11	12	13	14	15	16
129 136	137 144	145 152	153 160	161 168	169 176	177 184	185 192
17	18	19	20	21	22	23	24
193 200	201 208	209 216	217 224	225 232	233 240	241 248	249 256
25	26	27	28	29	30	31	32

Addresses

Each cell is designated by a natural number called an **address**. A bitstring longer than 8 bits is designated by the address of its first 8-string.

Each address is a **physical address** or a **virtual address**.

Physical Address

A physical address designates a single cell in primary memory. Address 0 designates the first cell, and consecutive addresses designate successive cells.

Virtual Address

A virtual address has several formats.

1st Order Format

1 7	8 16	17 25	26 34	35 43	44 52	53 64
R	5-T	4-T	3-T	2-T	1-T	O

R Root Index

5-T 5th Order Key Index

4-T 4th Order Key Index

3-T 3rd Order Key Index

2-T 2nd Order Key Index

1-T 1st Order Key Index

O Offset

2nd Order Format

1 7	8 16	17 25	26 34	35 43	44 64
R	5-T	4-T	3-T	2-T	O

R Root Index

5-T 5th Order Key Index

4-T 4th Order Key Index

3-T 3rd Order Key Index

2-T 2nd Order Key Index

O Offset

3rd Order Format

1 7	8 16	17 25	26 34	35 64
R	5-T	4-T	3-T	O

R Root Index

5-T 5th Order Key Index

4-T 4th Order Key Index

3-T 3rd Order Key Index

O Offset

4th Order Format

1 7	8 16	17 25	26 64
R	5-T	4-T	O

R Root Index

5-T 5th Order Key Index

4-T 4th Order Key Index

O Offset

5th Order Format

1 7	8 16	17 64
R	5-T	O

R Root Index

5-T 5th Order Key Index

O Offset

6th Order Format

1 7	8 64
R	O

R Root Index

O Offset

Chapter 2

Units

A unit accesses the primary memory when a bitstring is transferred between this unit and designated cells.

1. The unit generates the physical address designating the first of the designated cells in primary memory.
2. The bitstring is transferred between the unit and the designated cells in primary memory.

Physical Address

When a bitstring in primary memory is addressed, all of its constituant 8-strings are also designated.

Transfer

The access is either a **read** or a **write**, depending on the direction of transfer.

Read

The access is a read when the string is sent from the unit to primary memory.

Write

The access is a write when the string is sent from primary memory to the unit.

Order of Accesses

For every unit and for every primary memory cell, a read from the cell by the unit will yield the value written by the last write to that same cell by that same unit.

Virtual Address

A virtual address is translated into a physical address.

Frame Table

The **frame table** holds the data for address translation.

Root Table

A root table holds 6th order frame descriptors, 6th order frame keys or 5th order frame table descriptors.

6th Order Frame Descriptor

1–7	8–57	58	59	60	61	62	63	64
T	00000000000000000000000000000000000000000000000000	M	S	1	W	R	X	0

- T Frame Base
- M Modified
- S Swapped
- W Write Permission
- R Read Permission
- X Execute Permission

6th Order Frame Key

1–59	60	61	62	63	64
T	0	W	R	X	1

- T Gate Pointer
- W Write Permission
- R Read Permission
- X Execute Permission

5th Order Frame Table Descriptor

1–52	53–64
T	000010000000

- T Table Base

6th Order Frame Gate

A 6th order frame gate is pointed to by a 6th order frame key.

1–7	8–57	58	59	60–64
T	00000000000000000000000000000000000000000000000001	M	S	00000

- T Frame Base
- M Modified
- S Swapped

5th Order Frame Table

A 5th order frame table holds 5th order frame descriptors, 5th order frame keys or 4th order frame table descriptors.

5th Order Frame Descriptor

1–16	17–57	58	59	60	61	62	63	64
T	00000000000000000000000000000000000000000	M	S	1	W	R	X	0

- T Frame Base
- M Modified
- S Swapped
- W Write Permission
- R Read Permission
- X Execute Permission

5th Order Frame Key

1–59	60	61	62	63	64
T	0	W	R	X	1

T Gate Pointer

W Write Permission

R Read Permission

X Execute Permission

4th Order Frame Table Descriptor

1–52	53–64
T	0 0 0 0 1 0 0 0 0 0 0 0

T Table Base

5th Order Frame Gate

A 5th order frame gate is pointed to by a 5th order frame key.

1–16	17–57	58	59	60–64
T	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	M	S	0 0 0 0 0

T Frame Base

M Modified

S Swapped

4th Order Frame Table

A 4th order frame table holds 4th order frame descriptors, 4th order frame keys or 3rd order frame table descriptors.

4th Order Frame Descriptor

1–25	26–57	58	59	60	61	62	63	64
T	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	M	S	1	W	R	X	0

T Frame Base

M Modified

S Swapped

W Write Permission

R Read Permission

X Execute Permission

4th Order Frame Key

1–59	60	61	62	63	64
T	0	W	R	X	1

T Gate Pointer

W Write Permission

R Read Permission

X Execute Permission

3rd Order Frame Table Descriptor

1–52	53–64
T	0 0 0 0 1 0 0 0 0 0 0 0

T Table Base

4th Order Frame Gate

A 4th order frame gate is pointed to by a 4th order frame key.

1–25	26–57	58	59	60–64
T	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	M	S	0 0 0 0 0

T Frame Base

M Modified

S Swapped

3rd Order Frame Table

A 3rd order frame table holds 3rd order frame descriptors, 3rd order frame keys or 2nd order frame table descriptors.

3rd Order Frame Descriptor

1–34	35–57	58	59	60	61	62	63	64
T	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	M	S	1	W	R	X	0

T Frame Base

M Modified

S Swapped

W Write Permission

R Read Permission

X Execute Permission

3rd Order Frame Key

1–59	60	61	62	63	64
T	0	W	R	X	1

T Gate Pointer

W Write Permission

R Read Permission

X Execute Permission

2nd Order Frame Table Descriptor

1–52	53–64
T	0 0 0 0 1 0 0 0 0 0 0 0

T Table Base

3rd Order Frame Gate

A 3rd order frame gate is pointed to by a 3rd order frame key.

1–34	35–57	58	59	60–64
T	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	M	S	0 0 0 0 0

T Frame Base

M Modified

S Swapped

2nd Order Frame Table

A 2nd order frame table holds 2nd order frame descriptors, 2nd order frame keys or 1st order frame table descriptors.

2nd Order Frame Descriptor

1 – 43	44 – 57	58	59	60	61	62	63	64
T	0 0 0 0 0 0 0 0 0 0 0 0 0 0	M	S	1	W	R	X	0

T Frame Base

M Modified

S Swapped

W Write Permission

R Read Permission

X Execute Permission

2nd Order Frame Key

1 – 59	60	61	62	63	64
T	0	W	R	X	1

T Gate Pointer

W Write Permission

R Read Permission

X Execute Permission

1st Order Frame Table Descriptor

1 – 52	53 – 64
T	0 0 0 0 1 0 0 0 0 0 0 0

T Table Base

2nd Order Frame Gate

A 2nd order frame gate is pointed to by a 2nd order frame key.

1 – 43	44 – 57	58	59	60 – 64
T	0 0 0 0 0 0 0 0 0 0 0 0 0 1	M	S	0 0 0 0 0

T Frame Base

M Modified

S Swapped

1st Order Frame Table

A 1st order frame table holds 1st order frame descriptors or 1st order frame keys.

1st order frame descriptor

1–52	53–57	58	59	60	61	62	63	64
T	0 0 0 0 0	M	S	1	W	R	X	0

- T Frame Base
- M Modified
- S Swapped
- W Write Permission
- R Read Permission
- X Execute Permission

1st Order Frame Key

1–59	60	61	62	63	64
T	0	W	R	X	1

- T Gate Pointer
- W Write Permission
- R Read Permission
- X Execute Permission

1st Order Frame Gate

A 1st order frame gate is pointed to by a 1st order frame key.

1–52	53–57	58	59	60–64
T	0 0 0 0 1	M	S	0 0 0 0 0

- T Frame Base
- M Modified
- S Swapped

Procedure

The translation runs in two phases.

Frame Table Walk

The frame table walk yields either a frame key or a frame descriptor from a virtual address.

1. The unit reads from the root table the entry which the root index designates.
2. If the entry has none of the aforementioned formats, then condition **ZEE** happens.

   Otherwise, if the entry is either a 6th order frame key or a 6th order frame descriptor, then the unit performs the 6th order frame access with the entry.

   Otherwise, the unit reads from the 5th order frame table the entry which the 5th order table index designates.
3. If the entry has none of the aforementioned formats, then condition **ZEE** happens.

   Otherwise, if the entry is either a 5th order frame key or a 5th order frame descriptor, then the unit performs the 5th order frame access with the entry.

   Otherwise, the unit reads from the 4th order frame table the entry which the 4th order table index designates.
4. If the entry has none of the aforementioned formats, then condition **ZEE** happens.

   Otherwise, if the entry is either a 4th order frame key or a 4th order frame descriptor, then the unit performs the 4th order frame access with the entry.

   Otherwise, the unit reads from the 3rd order frame table the entry which the 3rd order table index designates.

5. If the entry has none of the aforementioned formats, then condition **ZEE** happens.

   Otherwise, if the entry is either a 3rd order frame key or a 3rd order frame descriptor, then the unit performs the 3rd order frame access with the entry.

   Otherwise, the unit reads from the 2nd order frame table the entry which the 2nd order table index designates.

6. If the entry has none of the aforementioned formats, then condition **ZEE** happens.

   Otherwise, if the entry is either a 2nd order frame key or a 2nd order frame descriptor, then the unit performs the 2nd order frame access with the entry.

   Otherwise, the unit reads from the 1st order frame table the entry which the 1st order table index designates.

7. If the entry has none of the aforementioned formats, then condition **ZEE** happens.

   Otherwise, the unit performs the 1st order frame access with the entry.

*n*th Order Frame Access

The *n*th order frame access gives a physical address needed in step 1 of primary memory access from an *n*th order frame key or an *n*th order frame descriptor.

1. If the entry does not allow the access, then condition **ZEE** happens.

   The entry allows the access if one and only one of the following conditions are met.

   - The access is a write and the W-bit is 1.
   - The access is a read and the R-bit is 1.
   - The access is a read of a instruction (as described in step (1) of the instruction cycle) and the X-bit is 1.
   - The access is a read of a root table descriptor (as described with instruction BRS) and all 3 bits are 0.

2. If the entry is a key, then the unit reads the gate to which the key's gate pointer points.
3. If the S-bit is 1, then condition **ZSW** happens.
4. If the access is a write, then the unit sets the E-bit of the entry in memory to 1.
5. The unit adds the offset to the frame pointer to give the physical address.

Part II

Interruptions

Chapter 3

Structures

In an interruption, data is exchanged between units.

Target

Each dependent unit is a target.

Gate

An interrupt gate is a cell in primary memory which is bound to a target.

Entry

An interrupt entry holds a virtual address to data and a pointer to the root table with which the address must be translated.

1 … 52	53 … 64
T	0 0 0 0 0 0 0 0 0 0 0 0

65 … 128
B

T Root Table Pointer

B Data Virtual Address

Source

A source is a unit which initiates an interruption.

Chapter 4

Operation

An interruption happens when the source writes an interrupt entry to the interrupt gate of the target.

1. The source writes the entry to the target's gate.
2. The target translates the entry's virtual address with the root table pointed to by the entry's root table pointer.
3. The target process the data pointed to by the aforegiven physical address.

Part III

Execution

Chapter 5

Operation

In the OSER, processing units follow a fixed operation.

Instruction Cycle

A processing unit follows the *instruction cycle*.

1. The unit reads an instruction.
2. If the instruction is not recognized, then condition **AEA** happens.
3. The unit performs all accesses which the instruction depends on.

Condition

When a condition happens, the unit executes a **trap**.

A trap is a special program which gets the unit's state at the time of the condition as input.

Chapter 6

Processing Units

In the OSER, the processing units are built to a particular architecture.

Register File

The register file does not depend on primary memory.

0-15 – Data

1 … 64
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

IP – Instruction Pointer

1 … 63	64
T	S

T Pointer

S State

- 0 Running
- 1 Stopped

SR – Status Register

1–52	53–61	62–63	64
R	0 0 0 0 0 0 0 0 0	C	0

65–128
IP

R Root Table Pointer

C Condition code

IP Instruction Pointer

Data

Data can be either a fixed point number or a floating point number.

Fixed Point Numbers

Let $Z_U(S, M)$ be the number expressed by a bitstring S with lowest power M in an unsigned interpretation.

$$Z_U(S, M) = 2^M \sum_{i=1}^{|S|} 2^{|S|-i} S_i$$

Let $Z_Z(S, M)$ be the number expressed by a bitstring S with lowest power M in a signed interpretation.

$$Z_Z(S, M) = 2^M \left(-2^{|S|-1} S_1 + \sum_{i=2}^{|S|} 2^{|S|-i} S_i \right)$$

U/Z1

1–8
S

U/Z2

1–16
S

U/Z4

1–32
S

U/Z8

1–64
S

Floating Point Numbers

Let $X(V, M, B)$ be the number expressed by a sign bit V, an exponent bitstring M and a fraction bitstring B.

$$X(V, M, B) = (-1)^V 2^{Z_U(M,0)-2^{|M|-1}} Z_U(B, -1-|B|)$$

X4

1	2–9	10–32
V	M	B

X8

1	2 – 12	13 – 64
V	M	B

X16

1	2 – 16	17 – 64
V	M	B

65 – 128
B

X32

1	2 – 20	21 – 64
V	M	B

65 – 128
B

129 – 192
B

193 – 256
B

Instructions

A instruction decribes a change in the program's state. Each instruction has various **formats**.

Control Instruction

A control instruction changes the run of the program when its condition is fulfilled. Each bit of the instruction condition designates a value of the SR-condition field, and the instruction condition is fulfilled if the SR-condition has a value whose bit in the instruction's condition is 1.

A control instruction has three formats.

Constant Format

1 – 3	4	5 – 8	9 – 12	13 – 32
0 0 0	*	B	R	W

In this format:

- the target is register *R*;
- the source is *W*.

Memory Format

1 – 3	4	5 – 8	9 – 12	13 – 16	17 – 32
0 1 0	*	B	R	Q	A

In this format:

- the target is register *R*;
- the source is the primary memory at the address given by the sum of *A* and register *Q*.

Register Format

1 – 3	4	5 – 8	9 – 12	13 – 16
1 1 0	*	B	R	Q

In this format, the target and source are registers *R* and *Q*.

SCI – Skip

After this instruction:

- *IP* is set to the sum of *IP* and the source;
- the target is set to *IP* before the instruction.

1 3	4	5 32
* * *	0	* *

1 3	4	5 16
* * *	0	* * * * * * * * * * * *

JMP – Jump

After this instruction:

- *IP* is set to the source;
- the target is set to *IP* before the instruction.

1 3	4	5 32
* * *	1	* *

1 3	4	5 16
* * *	1	* * * * * * * * * * * *

Move Instruction

A move instruction describes a movement of data between primary memory and the register file.

STR – Move

After this instruction, the target is set to the source.

This instruction has four formats.

Constant Format

1 8	9 12	13 32
0 0 1 1 0 0 0 0	Z	W

In this format:

- the target is register *Z*;
- the source is *W*.

Memory Format

1 8	9 12	13 16	17	18 20	21 32
0 1 1 1 0 0 0 0	Z	Q	Ä	G	A

In this format:

- the target is register *Z*;
- if the Ä-bit is 0, then the source is the primary memory at the address given by the sum of *A* and register *Q*; otherwise:
  - the source is the primary memory cell which register *Q* designates;
  - *A* is added to register *Q* after the instruction.

1 8	9 12	13 16	17	18 20	21 32
1 0 1 1 0 0 0 0	Z	Q	Ä	G	A

In this format:

- if the Ä-bit is 0, then the target is the primary memory at the address given by the sum of *A* and register *Z*; otherwise:

  - *A* is added to register *Z* before the instruction;
  - the target is the primary memory at the address in register *Z*;
- the source is register *Q*.

Register Format

1–8	9–12	13–16
1 1 1 1 0 0 0 0	Z	Q

In this format, the target and source are registers *Z* and *Q*.

USS – Atomic Move

This instruction's execution depends on its format.

This instruction has two formats.

1–8	9–12	13–16	17	18–20	21–32
0 1 1 1 0 0 0 1	Z	Q	0	G	A

In this format:

- the target is register *Z*;
- the source is the primary memory at the address given by the sum of *A* and register *Q*.

1–8	9–12	13–16	17	18–20	21–32
1 0 1 1 0 0 0 1	Z	Q	0	G	A

In this format:

- the target is the primary memory at the address given by the sum of *A* and register *Z*;
- the source is register *Q*.

In particular, this instruction will trap if a write to the target has happened after the last **USS** thereto.

BRS – Context Switch

This instruction's execution depends on its format.

This instruction has two formats.

1–8	9–12	13–16	17–32
0 1 1 1 0 0 1 0	R	Q	A

In this format:

- *SR* is written to primary memory at the address in register *R*;
- a new status register is read from primary memory at the address given by the sum of *A* and register *Q*.

1–8	9–12	13–16
1 1 1 1 0 0 1 0	R	Q

In this format:

- *SR* is written to primary memory at the address in register *R*;
- a new status register is read from primary memory at the address given by the sum of *A* and register *Q*.

UDB – Interruption

1–8	9–12	13–15	16	17–32
1 0 1 1 0 0 1 0	Z	Q	0	A

In this instruction:

- the source is the register pair $(Q, Q+1)$;
- the target is the primary memory at the address given by the sum of *A* and register *Z*.

Bitwise Instruction

A bitwise instruction describes a calculation over individual bits.

Each bitwise instruction has four formats.

Constant Format

1 4	5 8	9 12	13 32
0 0 1 0	* * * *	Z	W

In this format:

- the target is register *Z*;
- the source is *W*.

Memory Format

1 4	5 8	9 12	13 16	17	18 20	21 32
0 1 1 0	* * * *	Z	Q	Ä	G	A

In this format:

- the target is register *Z*;
- if the Ä-bit is 0, then the source is the primary memory whose address is given by the sum of *A* and register *Q*;

  otherwise:

  - the source is the primary memory at the address in register *Q*;
  - *A* is added to register *Q* after the instruction.

1 4	5 8	9 12	13 16	17	18 20	21 32
1 0 1 0	* * * *	Z	Q	Ä	G	A

In this format:

- if the Ä-bit is 0, then the target is the primary memory at the address given by the sum of *A* and register *Z*;

  otherwise:

  - *A* is added to register *Z* before the instruction;
  - the target is the primary memory at the address in register *Z*;
- the source is register *Q*.

Register Format

1 4	5 8	9 12	13 16
1 1 1 0	* * * *	Z	Q

In this format, the target and source are registers *Z* and *Q*.

LSH – Left Shift

After this instruction, the target is shifted to the left by the number of bits given by the source.

1 4	5 8	9 32
* * * *	0 0 0 0	* *

1 4	5 8	9 16
* * * *	0 0 0 0	* * * * * * * *

RNS – Logical Right Shift

After this instruction, the target is shifted to the right by the number of bits given by the source. The leftmost bits are set to 0.

1 4	5 8	9 32
* * * *	0 0 0 1	* *

1 4	5 8	9 16
* * * *	0 0 0 1	* * * * * * * *

RFS – Arithmetic Right Shift

After this instruction, the target is shifted to the right by the number of bits given by the source. The leftmost bits are set to the target's first bit before the instruction.

1 4	5 8	9 32
* * * *	0 0 1 0	* *

1 4	5 8	9 16
* * * *	0 0 1 0	* * * * * * * *

THR – Rotate

After this instruction, the target is rotated to the right by the number of bits given by the source.

1 4	5 8	9 32
* * * *	0 0 1 1	* *

1 4	5 8	9 16
* * * *	0 0 1 1	* * * * * * * *

AND – Conjunction

After this instruction, each of the target's bits will be set to the conjunction of the matching bits of the target and the source.

1 4	5 8	9 32
* * * *	0 1 0 0	* *

1 4	5 8	9 16
* * * *	0 1 0 0	* * * * * * * *

OR – Union

After this instruction, each of the target's bits will be set to the union of the matching bits of the target and the source.

1 4	5 8	9 32
* * * *	0 1 0 1	* *

1 4	5 8	9 16
* * * *	0 1 0 1	* * * * * * * *

NOT – Complement

After this instruction, each of the target's bits will be set to the complement of the matching bits of the source.

1 4	5 8	9 32
* * * *	0 1 1 0	* *

1 4	5 8	9 16
* * * *	0 1 1 0	* * * * * * * *

SSW – Sign Swap

After this instruction, the target will be set to the negative of the source.

In the costant format, the source is zero-extended.

1 4	5 8	9 32
* * * *	0 1 1 1	* *

1 4	5 8	9 16
* * * *	0 1 1 1	* * * * * * * *

Arithmetic Instruction

An arithmetic instruction describes an arithmetic calculation.

All arithmetic instructions calculate over fixed point numbers.

Each arithmetic instruction has four formats.

Constant Format

1 4	5 8	9 12	13 32
0 0 1 0	* * * *	Z	W

In this format:

- the target is register *Z*;
- the source is *W*.

Memory Format

1 4	5 8	9 12	13 16	17	18 20	21 32
0 1 1 0	* * * *	Z	Q	Ä	G	A

In this format:

- the target is register *Z*;
- if the Ä-bit is 0, then the source is the primary memory at the address given by the sum of *A* and register *Q*; otherwise:
  - the source is the primary memory at the address in *Q*;
  - *A* is added to register *Q* after the instruction.

1 4	5 8	9 12	13 16	17	18 20	21 32
1 0 1 0	* * * *	Z	Q	Ä	G	A

In this format:

- if the Ä-bit is 0, then the target is the primary memory at the address given by the summ of *A* and register *Z*; otherwise:
  - *A* is added to register*Z* before the instruction;
  - the target is the primary memory at the address in register *Z*;
- the source is register *Q*.

Register Format

1 4	5 8	9 12	13 16
1 1 1 0	* * * *	Z	Q

In this format, the target and source are registers *Z* and *Q*.

GIV – Add

After this instruction, the source is added to the target.

In the constant format, the source is sign-extended.

1 4	5 8	9 32
* * * *	1 0 0 0	* *

1 4	5 8	9 16
* * * *	1 0 0 0	* * * * * * * *

TAC – Subtract

After this instruction, the source is subtracted from the target.

In the constant format, the source is sign-extended.

1 4	5 8	9 32
* * * *	1 0 0 1	* *

1 4	5 8	9 16
* * * *	1 0 0 1	* * * * * * * *

FLD – Multiply

After this instruction, the source is multiplied by the target.

In the constant format, the source is sign-extended.

1 4	5 8	9 32
* * * *	1 0 1 0	* *

1 4	5 8	9 16
* * * *	1 0 1 0	* * * * * * * *

CUT – Divide

After this instruction, the source is divided by the target.

In the constant format, the source is sign-extended.

1 4	5 8	9 32
* * * *	1 0 1 1	* *

1 4	5 8	9 16
* * * *	1 0 1 1	* * * * * * * *