

Einheitlicher  
Sammenstand  
Elektronischer  
Rechengeräte

Specification

---

Daniel CAMPOS DO NASCIMENTO © 2020

Licensed under Creative Commons CC-BY-SA 4.0.

### **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 Reichsgemeinnutzerlaubnis (Reich Public Use Permissions) and the Mean Need Leave (General Usage License) respectively, which are equivalent to CC-BY-ND; they were later relicensed under the Reichsgemeinnutz- 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.

Reichstandortsgemeinschaft – Rsg

Reichsforschungsgemeinschaft für Rechenwissenschaft und -lehre – Rfg-r

British Forsee Fellowship for Recon- and Telllore

Norræn Rannsóknar Samfélag

© 1963-1970, 1972-1992

Left under the Mean Need Leave.

© 1993-

Left under the Mean Need and Spread Leave.

# Contents

<b>I</b>	<b>Access</b>	<b>3</b>
<b>1</b>	<b>Primary Memory</b>	<b>5</b>
	Bitstrings . . . . .	5
	8-String . . . . .	5
	16-String . . . . .	5
	32-String . . . . .	5
	64-String . . . . .	5
	128-String . . . . .	6
	256-String . . . . .	6
	Addresses . . . . .	6
	Physical Address . . . . .	6
	Virtual Address . . . . .	6
	1st Order Format . . . . .	6
	2nd Order Format . . . . .	7
<b>2</b>	<b>Units</b>	<b>9</b>
	Physical Address . . . . .	9
	Transfer . . . . .	9
	Read . . . . .	9
	Write . . . . .	9
	Order of Accesses . . . . .	9
	Virtual Address . . . . .	9
	Frame Table . . . . .	9
	Root Table . . . . .	9
	2nd Order Frame Gate . . . . .	10
	1st Order Frame Table . . . . .	10
	1st Order Frame Gate . . . . .	11
	Procedure . . . . .	11
	Frame Table Walk . . . . .	11
	nth Order Frame Access . . . . .	11
<b>II</b>	<b>Interruptions</b>	<b>13</b>
<b>3</b>	<b>Structures</b>	<b>15</b>
	Target . . . . .	15
	Gate . . . . .	15
	Entry . . . . .	15
	Source . . . . .	15
<b>4</b>	<b>Operation</b>	<b>17</b>
<b>III</b>	<b>Execution</b>	<b>19</b>
<b>5</b>	<b>Operation</b>	<b>21</b>
	Instruction Cycle . . . . .	21
	Conditions . . . . .	21

<b>6 Processing Units</b>	<b>23</b>
Register File	23
0-15 – Data	23
IP – Instruction Pointer	23
SR – Status Register	24
Data	24
Fixed Point Numbers	24
U/Z1	24
U/Z2	24
U/Z4	24
Floating Point Numbers	24
X4	24
X8	24
X16	25
X32	25
Instructions	25
Control Instruction	25
SCI – Skip	26
JMP – Jump	26
Move Instruction	26
STR – Move	26
USS – Atomic Move	27
BRS – Context Switch	27
UDB – Interruption	28
Bitwise Instruction	28
LSH – Left Shift	28
RNS – Logical Right Shift	29
RFS – Arithmetic Right Shift	29
THR – Rotate	29
AND – Conjunction	29
OR – Union	29
NOT – Complement	29
SSW – Sign Swap	30
Arithmetic Instruction	30
GIV – Add	31
TAC – Subtract	31
FLD – Multiply	31
CUT – Divide	31
<b>7 Channel Units</b>	<b>33</b>
Register File	33
IP – Instruction Pointer	33
IGP – Interrupt Gate Pointer	33
IEP – Interrupt Entry Pointer	33
SR – Status Register	33
Instructions	34
Control Instruction	34
Transfer Instruction	34
Read	35
Write	35
Operation	35
Programs	35
Traps	35

# 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 **channel 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
1	2	3	4	
33	40 41	48 49	56 57	64
5	6	7	8	

## 128-String

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

## 256-String

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

### Virtual Address

A virtual address has two formats.

#### 1st Order Format

1	10 11	20 21	32
R	1-T	O	

R Root Index

1-T 1st Order Key Index

O Offset

**2nd Order Format**

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 cell 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 constituent 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 2nd order frame descriptors, 2nd order frame keys or 1st order frame table descriptors.

## 2nd Order Frame Descriptor

1	10	11	26	27	28	29	30	31	32												
T	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	M	S	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	28	29	30	31	32
T	W	R	X	1	

T Gate Pointer

W Write Permission

R Read Permission

X Execute Permission

## 1st Order Frame Table Descriptor

1	20	21	32
T		1 0 0 0 0 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	10	11	26	27	28	29	32							
T			000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000											

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																				20	21									26	27	28	29	30	31	32
T																				1		0	0	0	0	0	M	S	W	R	X	0				

T Frame Base

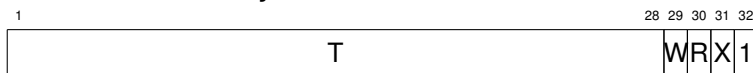
M Modified

S Swapped

W Write Permission

R Read Permission

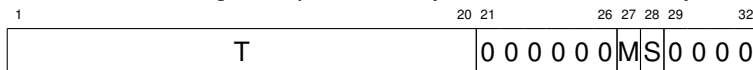
X Execute Permission

**1st Order Frame Key**

- 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.



- T Frame Pointer
- 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 2nd order frame key or a 2nd order frame bewrit, 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 index designates.
3. If the entry has none of the forbewritten formats, then condition **ZEE** happens.  
 Otherwise, the unit undertaces the 1st order frame grasp 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

A target is a processing unit.

### 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.



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.

The target must start the instruction cycle after having processed the interrupt entry.



## **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 must happen before the instruction.
4. The unit executes the instruction.
5. The unit performs all accesses which must happen after the instruction.

### Conditions

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		32
	0	
	1	
	2	
	3	
	4	
	5	
	6	
	7	
	8	
	9	
	10	
	11	
	12	
	13	
	14	
	15	

#### IP – Instruction Pointer

1		31	32
	T		S

T Pointer

S State

0 Running

1 Stopped





- 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	0	0	Z	W

In this format:

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

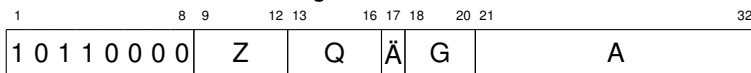
### Memory Format

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  $\ddot{A}$ -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;

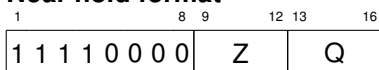
- $A$  is added to register  $Q$  after the instruction.



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  $Q$ .

### Near hold format

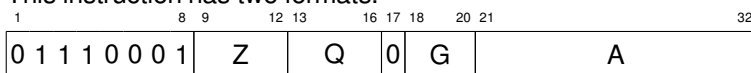


In this format, the target and source are registers  $Z$  and  $Q$ .

### USS – Atomic Move

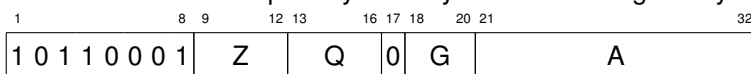
This instruction's execution depends on its format.

This instruction has two formats.



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  $Q$ .



In this format:

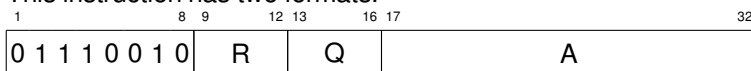
- the target is the primary memory at the address given by the sum of  $A$  and register  $Z$ ;
- the source is register  $Q$ .

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

### BRS – Context Switch

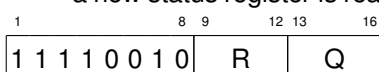
This instruction's execution depends on its format.

This instruction has two formats.



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  $Q$ .



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**

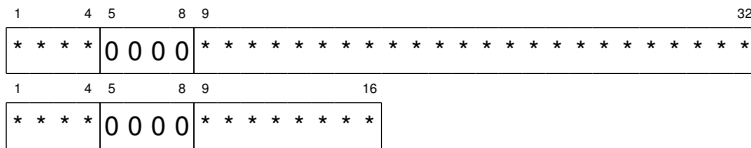
1	4	5	8	9	12	13	16		
1	1	1	0	*	*	*	*	Z	Q

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

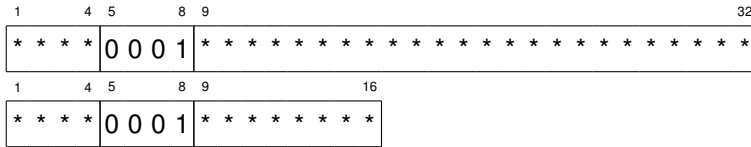
**LSH – Left Shift**

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

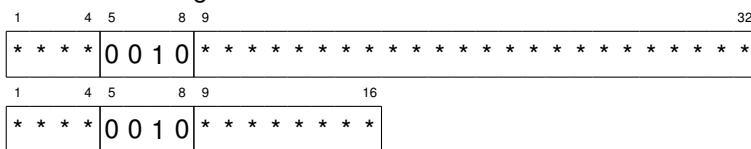


**RNS – Logical Right Shift**

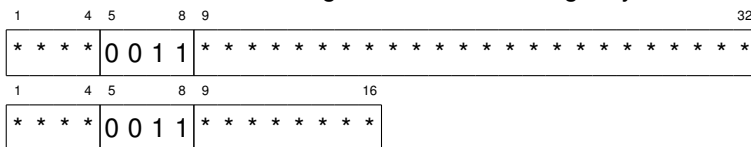
After this instruction, the target is shifted to the right by the number of bits given by the source. The leftmost bits are set to 0.

**RFS – Arithmetic Right Shift**

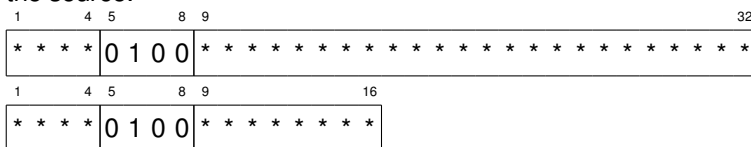
After this instruction, the target is shifted to the right by the number of bits given by the source. The leftmost bits are set to the target's first bit before the instruction.

**THR – Rotate**

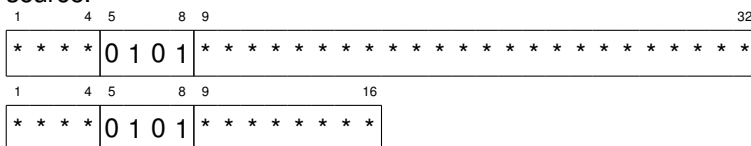
After this instruction, the target is rotated to the right by the number of bits given by the source.

**AND – Conjunction**

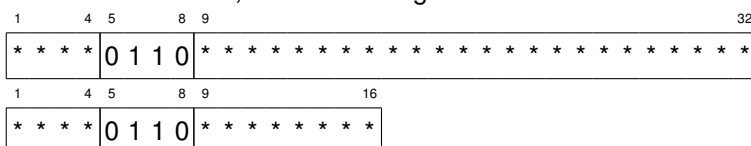
After this instruction, each of the target's bits will be set to the conjunction of the matching bits of the target and the source.

**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.

**NOT – Complement**

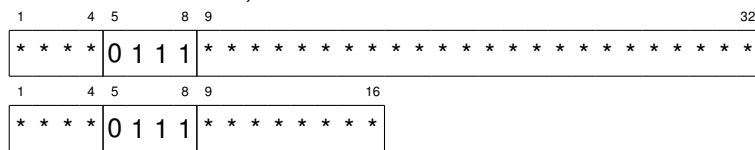
After this instruction, each of the target's bits will be set to the complement of the matching bits of the source.



## SSW – Sign Swap

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

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



## Arithmetic Instruction

An arithmetic instruction describes an arithmetic calculation.

All arithmetic instructions calculate over fixed point numbers.

Each arithmetic instruction has four formats.

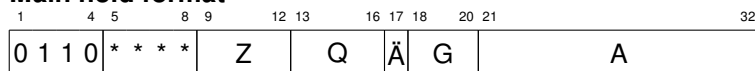
### Constant Format



In this format:

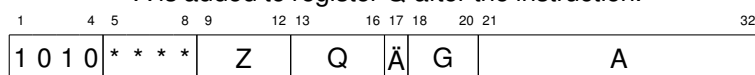
- the target is register  $Z$ ;
- the source is  $W$ .

### Main hold format



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.



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  $Q$ .

### Near hold format

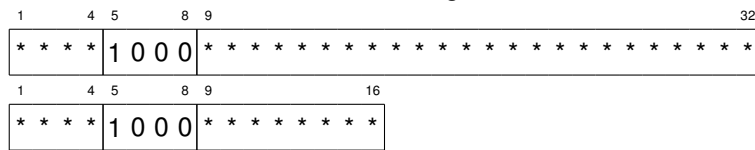


In this format, the target and source are registers  $Z$  and  $Q$ .

**GIV – Add**

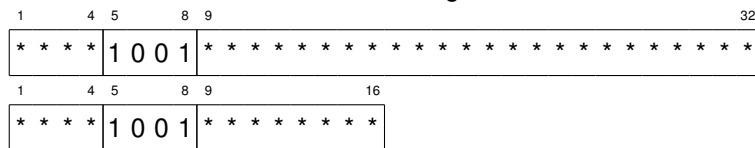
After this instruction, the source is added to the target.

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

**TAC – Subtract**

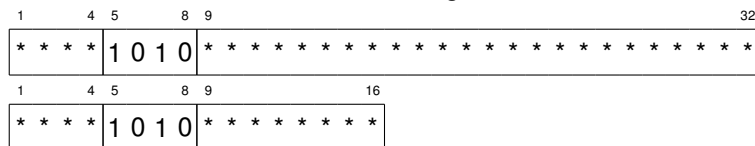
After this instruction, the source is subtracted from the target.

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

**FLD – Multiply**

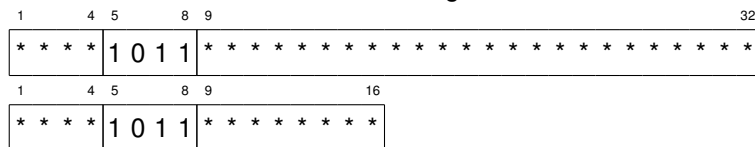
After this instruction, the source is multiplied by the target.

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

**CUT – Divide**

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

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





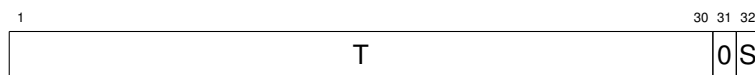
# Chapter 7

## Channel Units

The channel units facilitate data transfer into and out of primary memory. Each channel unit is made up of a *device* and a *execution unit*. Here will be described the execution unit's architecture.

### Register File

#### IP – Instruction Pointer



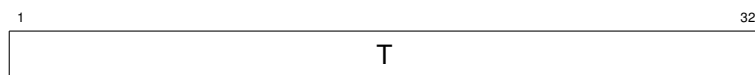
T Pointer

S State

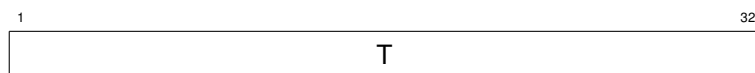
0 Running

1 Stopped

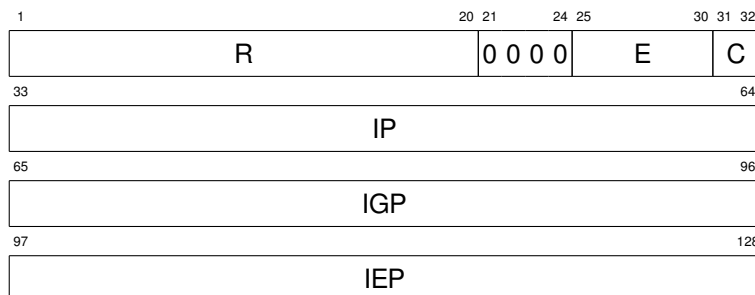
#### IGP – Interrupt Gate Pointer



#### IEP – Interrupt Entry Pointer



#### SR – Status Register



R Root Table Pointer

E Exception Code

C Condition Code

IP Instruction Pointer  
IGP Interrupt Gate Pointer  
IEP Interrupt Entry Pointer

