The cLEMENCy Architecture

Lightning
Legitimate Business Syndicate

July 2017

The cLEMENCy Architecture

Legitimate Business Syndicate

Deadwood Fuzyll Gyno HJ Hoju Jetboy Jymbolia Lightning Selir Sirgoon Thing2 Vito

Compiled from revision `c9a31bdf07c810d0a42efd3a6cf86b8012ba28f4` on July 27, 2017 .

Contents

List of Tables

Chapter 1

Basic Architecture

cLEMENCy is the LEgitbs Middle ENdian Computer architecture developed by Lightning for DEF CON CTF.

Each byte is 9 bits of data, bit 0 is the left most significant bit. Middle-Endian data stores bits 9 to 17, followed by bits 0 to 8, then bits 18 to 27 in memory when handling three bytes. Two bytes of data will have bits 9-17 then bits 0 to 8 written to memory.

Register XXYYZZ → Memory YYXXZZ

Register XXYY → Memory YYXX

1.1 Registers

cLEMECy has 32 general purpose 27-bit wide registers that serve both floating point and integer math operations along with a separate flag register. The multi-register format allows for a 54-bit value to be used during math operations by using two registers side by side while the starting register can be any register. If an instruction goes past the PC register while accessing multiple registers then access continues to R0. Attempts to write to PC by a load instruction are ignored.

The multi-register format and floating point operations are optional components of the processor, checking the processor features is recommended before attempts are made in using such instructions.

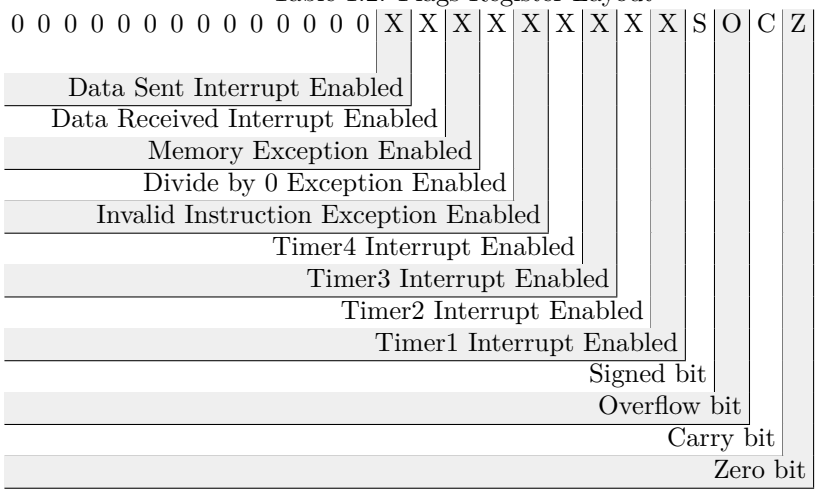
The ST, RA, and PC registers have a special purpose while all other registers are general use. The following table is a recommended register setup for compilers:

Table 1.1: Registers

Register Name	Register Number	Notes
R0	0	General purpose, parameter 1 and function return value
R1 to R8	1 to 8	General purpose, parameters 2 to 8
R9 to R28	9 to 28	General purpose, saved between function calls
ST	29	Pointer into the current memory location holding the current end of the stack
RA	30	Return address register, filled in by the call instruction, used when a return is executed
PC	31	Program counter, register is read-only
FL		Current flag and interrupt state

The flags register has the following layout:

Table 1.2: Flags Register Layout



1.2 Stack

Due to no specific stack based instructions, there is no expected direction the stack should grow, however during an interrupt the processor will subtract 99 bytes of data from the stack pointer when storing all registers to the stack and the interrupt return will read 99 bytes from the current stack pointer. If an implementation needs the interrupts to add to the stack instead of subtracting

then changing the 'Interrupt stack direction flag' bit in the features area of the processor will accomplish this.

1.3 Relative Memory Reference

All relative references are from the beginning of the instruction handling the relative reference. A relative branch will adjust PC by the amount in the relative offset without including the branch instruction size. Only if the branch is not taken is PC adjusted by the instruction size.

Chapter 2

Memory layout and IO

There are 2 main areas of memory, the RAM area and the DMA mapped areas. Processor execution starts at memory offset 0 and all DMA memory has the high bit of the memory address set. The following table provides the memory mapping for cLEMENCy processors:

Table 2.1: Memory Mapping

Memory Start	Memory End	Information
0000000	3FFFFFF	Main Program Memory
4000000	400001D	Clock IO
4010000	4010FFF	Flag IO
5000000	5001FFF	Data Received
5002000	5002002	Data Received Size
5010000	5011FFF	Data Sent
5012000	5012002	Data Sent Size
6000000	67FFFFF	Shared Memory
6800000	6FFFFFF	NVRAM Memory
7FFFF00	7FFFF1B	Interrupt Pointers
7FFFF80	7FFFFFF	Processor Identification and Features

2.1 Memory Protection

Memory is broken up into 1024 byte pages. Each page can have 1 of 4 states applied to it.

Table 2.2: Memory Protection States

State	Meaning
0	No Access
1	Read Only
2	Read/Write
3	Read/Execute

Attempting to interact with memory that is inconsistent with its current state will result in a memory exception occurring. If the exception is turned off and the attempt is execution then the processor will halt. The memory protection flags are from the processor only allowing for external IO controllers to modify any memory they are associated with even if the memory protection flags are set on their regions to Read Only or No Access.

2.2 Clock IO

There are 6 bytes per timer with 4 timers maximum. A 0 for the timer delay disables that specific timer. Each timer has a 1 millisecond accuracy.

Table 2.3: Clock and Timer Parameters

Memory Start	Bytes	Details
4000000	3	Timer 1 Delay
4000003	3	Number of milliseconds left for Timer 1
4000006	3	Timer 2 Delay
4000009	3	Number of milliseconds left for Timer 2
400000C	3	Timer 3 Delay
400000F	3	Number of milliseconds left for Timer 3
4000012	3	Timer 4 Delay
4000015	3	Number of milliseconds left for Timer 4
4000018	6	Number of seconds since Aug. 02, 2013 09:00 PST
400001E	3	Number of processing ticks since processor start

2.3 Flag IO

This memory area contains the flag of the current instance of the running firmware. Its default is readable and writable however writes are ignored. The processor enforces a minimum readable setting.

2.4 Data Received

When the Data Received interrupt fires, this area has been filled in with network related traffic. No more data can be received until the value stored in the Data Received Size area is set to 0.

2.5 Data Received Size

This area ia a 3 byte value storing the size of data received. If this value is non-zero then no more data can be received from the network.

2.6 Data Sent

This area is a buffer to write data to that is to be sent over the network. The data is not sent until the Data Sent Size value is specified.

2.7 Data Sent Size

This area is a 3 byte value storing the size of data being sent. When a value is written to this memory location the amount of data specified will be sent over the network. Upon completion the value is set to 0 and the Data Sent interrupt is fired to indicate success.

2.8 Shared Memory

This area of memory is an optional component that allows a processor to have a shared memory region with other processors on the same bus. If this area is detected on initialization then it will be marked Read/Write. Care must be taken in communicating with other processors as data may be overwritten.

2.9 NVRAM Memory

This area of memory is an optional component that allows a processor to have storage between a full power cycle. If this area is detected on initialization then it will be marked Read/Write.

2.10 Interrupt Pointers

Each interrupt has 3 bytes to indicate the area of memory to jump to upon the interrupt firing. If the value is 0 then the interrupt is not fired. It is possible for an interrupt to fire while another interrupt is processing so disabling and enabling interrupts is highly recommended to avoid conflicts.

Table 2.4: Interrupt Pointer Addresses

Memory Start	Interrupt
7FFFF00	Timer 1
7FFFF03	Timer 2
7FFFF06	Timer 3
7FFFF09	Timer 4
7FFFF0C	Invalid Instruction
7FFFF0F	Divide by 0
7FFFF12	Memory Exception
7FFFF15	Data Received
7FFFF18	Data Sent

2.11 Processor Identification

The last 128 bytes of memory are used for processor identification and information of supported functionality. Writes to this area are ignored with the exception of the Interrupt Stack Direction Flag.

Table 2.5: Processor Identification Attributes

Memory Start	Bytes	Details
7FFFF80	20	Processor name
7FFFFA0	3	Processor version
7FFFFA3	3	Processor functionality flags
7FFFFA6	4A	For future use
7FFFFF0	1	Interrupt stack direction flag
7FFFFF1	F	For future use

It is recommended to implementors that the processor version contains a Major, Minor, and Revision value, one value per byte entry.

The functionality flags has the following information:

Table 2.6: Processor Functionality Flags

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	X	X	X
	Interrupts are able to flip stack storage direction		
		FPU built into the processor	
			Processor supports 54-bit math

The low bit of the interrupt stack direction flag dictates the direction the interrupt writes to the stack. A value of 0, the default, results in the interrupt subtracting 99 bytes from the stack pointer then storing all registers in the 99 byte buffer starting with register 0. A value of 1 results in the interrupt storing

and incrementing the stack pointer starting with register 0. The interrupt return behaves in the opposite manner to restore the registers.

Chapter 3

Interrupts and Exceptions

When any interrupt is fired, all 32 general purpose registers and low 4 bits of the flags register are stored to the current stack before the processor begins executing the specified interrupt routine. Upon returning from an interrupt, all registers and low bits of the flag register are restored from the stack. The Disable Interrupts, DI, and Enable Interrupts, EI, instructions are used to temporarily disable an interrupt. Any interrupt with a value of 0 will not be called and ignored.

3.1 Timer 1 to 4 Interrupts

Each timer has an accuracy of 1 millisecond and can be configured through the Clock IO.

3.2 Invalid Instruction Exception

When an invalid instruction is detected this interrupt is fired. If this interrupt is disabled with DI, (the "Disable Interrupts" instruction,) or by having this value be 0 then the processor will halt.

3.3 Divide by 0 Exception

Division with a divisor of 0 will trigger this interrupt. Additionally, all other floating point exceptions will also trigger this interrupt.

3.4 Memory Exception

An attempt to read, write, or execute memory with invalid permission bits will cause this interrupt to trigger.

3.5 Data Received Interrupt

When data is received over the network this interrupt is fired.

3.6 Data Sent Interrupt

When data is fully sent over the network this interrupt is fired.

3.7 Exceptions

Upon an exception occurring, all registers are moved to the stack, R0 is set to the PC location that failed, R1 is set to one of the following IDs indicating the type of exception, and R2 is a value specific to the exception type. If an exception occurs while the interrupt handling the exception is still active then the processor will halt.

Table 3.1: Exception Identifiers

Exception	ID	Value Meaning
Memory Read	0	Address that failed to be read
Memory Write	1	Address that failed to be written
Memory Execute	2	Address that failed to execute
Invalid Instruction	3	0
Floating Point Error	4	0
Divide By 0	5	0

If the exception is disabled or the interrupt has no registered handler then the exception is ignored and the result of the operation that failed is undefined. The only case this is problematic is upon an instruction fault or execution in non-executable memory. If no exception is registered it will cause the CPU to halt due to not advancing the PC which would otherwise cause an infinite fault loop. If multiple faults can occur on the same instruction then only one fault will occur although no guarantee of which fault takes priority.

Chapter 4

Instruction Set

Unless specified otherwise, all math and immediate values are unsigned for integer arithmetic while all floating point math is signed. All rX values can reference a general purpose register from 0 to 31. Any time the format rX:rX+Y is seen, the instruction will work on registers rX through and including rX+Y based on the value of Y. If present, the UF field controls if the flags get updated for the instruction.

4.1 AD: Add

0 6	7 11	12 16	17 21	22 25	26
0000000	rA	rB	rC	0000	UF

Format: `AD rA, rB, rC`

Purpose: Add two 27-bit integer registers together

Description: The 27-bit value in rC is added to the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB + rC`

Flags affected: `Z C O S`

4.2 ADC: Add With Carry

0 6	7 11	12 16	17 21	22 25	26
0100000	rA	rB	rC	0000	UF

Format: `ADC rA, rB, rC`

Purpose: Add two 27-bit integer registers together including the carry bit

Description: The 27-bit value in rC is added to the 27-bit value in rB, the carry bit from any previous operation is added to the result. The result is placed in rA.

Operation: `rA ← rB + rC + Carry_Bit`

Flags affected: `Z C O S`

4.3 ADCI: Add Immediate With Carry

0 6	7 11	12 16	17 23	24 25	26
0100000	rA	rB	Immediate	01	UF

Format: `ADCI rA, rB, IMM`

Purpose: Add a 7-bit immediate value to a 27-bit integer register including the carry bit

Description: The 7-bit immediate value is added to the 27-bit value in rB, the carry bit from any previous operation is added to the result. The result is placed in rA.

Operation: `rA ← rB + IMM + Carry_Bit`

Flags affected: `Z C O S`

4.4 ADCIM: Add Immediate Multi Reg With Carry

0 6	7 11	12 16	17 23	24 25	26
0100010	rA	rB	Immediate	01	UF

Format: `ADCIM rA, rB, IMM`

Purpose: Add a 7-bit immediate value to a 54-bit integer register including the carry bit

Description: The 7-bit immediate value is added to the 54-bit value in rB:rB+1, the carry bit from any previous operation is added to the result. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 + IMM + Carry_Bit`

Flags affected: `Z C O S`

4.5 ADCM: Add Multi Reg With Carry

0 6	7 11	12 16	17 21	22 25	26
0100010	rA	rB	rC	0000	UF

Format: `ADCM rA, rB, rC`

Purpose: Add two 54-bit integer registers together including the carry bit

Description: The 54-bit value in rC:rC+1 is added to the 54-bit value in rB:rB+1, the carry bit from any previous operation is added to the result. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 + rC:rC+1 + Carry_Bit`

Flags affected: `Z C O S`

4.6 ADF: Add Floating Point

0 6	7 11	12 16	17 21	22 25	26
0000001	rA	rB	rC	0000	UF

Format: `ADF rA, rB, rC`

Purpose: Add two 27-bit floating point registers together

Description: The 27-bit floating point value in rC is added to the 27-bit floating point value in rB, the result is placed in rA.

Operation: `rA ← rB + rC`

Flags affected: `Z C O S`

4.7 ADFM: Add Floating Point Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0000011	rA	rB	rC	0000	UF

Format: `ADFM rA, rB, rC`

Purpose: Add two 54-bit floating point registers together

Description: The 54-bit floating point value in rC:rC+1 is added to the 54-bit floating point value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 + rC:rC+1`

Flags affected: `Z C O S`

4.8 ADI: Add Immediate

0 6	7 11	12 16	17 23	24 25	26
0000000	rA	rB	Immediate	01	UF

Format: `ADI rA, rB, IMM`

Purpose: Add a 7-bit immediate value to a 27-bit integer register

Description: The 7-bit immediate value is added to the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB + IMM`

Flags affected: `Z C O S`

4.9 ADIM: Add Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0000010	rA	rB	Immediate	01	UF

Format: `ADIM rA, rB, IMM`

Purpose: Add a 7-bit immediate value to a 54-bit integer register

Description: The 7-bit immediate value is added to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 + IMM`

Flags affected: `Z C O S`

4.10 ADM: Add Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0000010	rA	rB	rC	0000	UF

Format: `ADM rA, rB, rC`

Purpose: Add two 54-bit integer registers together

Description: The 54-bit value in rC:rC+1 is added to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 + rC:rC+1`

Flags affected: `Z C O S`

4.11 AN: And

0 6	7 11	12 16	17 21	22 25	26
0010100	rA	rB	rC	0000	UF

Format: `AN rA, rB, rC`

Purpose: Bit-wise AND two 27-bit integer registers together

Description: The 27-bit value in rC is bit-wise AND to the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB & rC`

Flags affected: `Z C O S`

4.12 ANI: And Immediate

0 6	7 11	12 16	17 23	24 25	26
0010100	rA	rB	Immediate	01	UF

Format: `ANI rA, rB, IMM`

Purpose: Bit-wise AND a 27-bit integer register and 7-bit immediate together

Description: The 7-bit immediate value is bit-wise AND to the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB & IMM`

Flags affected: `Z C O S`

4.13 ANM: And Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0010110	rA	rB	rC	0000	UF

Format: `ANM rA, rB, rC`

Purpose: Bit-wise AND two 54-bit integer registers together

Description: The 54-bit value in rC:rC+1 is bit-wise AND to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 & rC:rC+1`

Flags affected: `Z C O S`

4.14 B: Branch Conditional

0 5	6 9	10 26
110000	Condition	Offset

Format: `Bcc Offset`

Purpose: Conditional branch to an offset

Description: If the specified condition is true then the sign extended offset is added to the current program counter.

The condition table is as follows:

cc	Value	Description	Flag State Checked
`n`	`0000`	Not Equal / Not Zero	Z == 0
`e`	`0001`	Equal / Zero	Z == 1
`l`	`0010`	Less Than	C == 1 AND Z == 0
`le`	`0011`	Less Than or Equal	C == 1 OR Z == 1
`g`	`0100`	Greater Than	C == 0 AND Z == 0
`ge`	`0101`	Greater Than or Equal	C == 0 OR Z == 1
`no`	`0110`	Not Overflow	O == 0
`o`	`0111`	Overflow	O == 1
`ns`	`1000`	Not Signed	S == 0
`s`	`1001`	Signed	S == 1
`sl`	`1010`	Signed Less Than	S != O
`sle`	`1011`	Signed Less Than or Equal	S != O OR Z == 1
`sg`	`1100`	Signed Greater Than	S == O AND Z == 0
`sge`	`1101`	Signed Greater Than or Equal	S == O
	1111	Always	

Operation:

```
if Flags == Condition then
  PC += (signed)Offset
```

Flags affected: *None*

4.15 BF: Bit Flip

0 8	9 13	14 18	19 25	26
101001100	rA	rB	1000000	UF

Format: `BF rA, rB`

Purpose: Bit flip a 27-bit register

Description: Invert all bits of the 27-bit rB register value and store the result in rA.

Operation: `rA ← ~rB`

Flags affected: `Z C O S`

4.16 BFM: Bit Flip Multi Reg

0 8	9 13	14 18	19 25	26
101001110	rA	rB	1000000	UF

Format: `BFM rA, rB`

Purpose: Bit flip a 54-bit register

Description: Invert all bits of the 54-bit rB:rB+1 register value and store the result in rA:rA+1.

Operation: `rA:rA+1 ← ~rB:rB+1`

Flags affected: `Z C O S`

4.17 BR: Branch Register Conditional

0 5	6 9	10 14	15 17
110010	Condition	rA	000

Format: `BRcc rA`

Purpose: Conditional branch to a register

Description: If the specified condition is true then the value in rA is placed into the program counter.

The condition table is as follows:

cc	Value	Description	Flag State Checked
n	0000	Not Equal / Not Zero	Z == 0
e	0001	Equal / Zero	Z == 1
l	0010	Less Than	C == 1 AND Z == 0
le	0011	Less Than or Equal	C == 1 OR Z == 1
g	0100	Greater Than	C == 0 AND Z == 0
ge	0101	Greater Than or Equal	C == 0 OR Z == 1
no	0110	Not Overflow	O == 0
o	0111	Overflow	O == 1
ns	1000	Not Signed	S == 0
s	1001	Signed	S == 1
sl	1010	Signed Less Than	S != O
sle	1011	Signed Less Than or Equal	S != O OR Z == 1
sg	1100	Signed Greater Than	S == O AND Z == 0
sge	1101	Signed Greater Than or Equal	S == O
	1111	Always	

Operation:

```
if Flags == Condition then
  PC = rA
```

Flags affected: *None*

4.18 BRA: Branch Absolute

0	8	9	35
111000100		Location	

Format: `BRA Location`

Purpose: Branch to a set absolute location

Description: The program counter is set to the specified 27-bit location.

Operation: `PC = Location`

Flags affected: *None*

4.19 BRR: Branch Relative

0	8	9	35
111000000		Offset	

Format: `BRR Offset`

Purpose: Branch to a relative offset

Description: The 27-bit offset is added to the current program counter.

Operation: `PC = PC + Offset`

Flags affected: *None*

4.20 C: Call Conditional

0 5	6 9	10 26
110101	Condition	Offset

Format: `Ccc Offset`

Purpose: Conditional call to an offset

Description: If the specified condition is true then the current program counter + 3 is saved into the RA register and the sign extended offset is added to the current program counter.

The condition table is as follows:

cc	Value	Description	Flag State Checked
`n`	`0000`	Not Equal / Not Zero	Z == 0
`e`	`0001`	Equal / Zero	Z == 1
`l`	`0010`	Less Than	C == 1 AND Z == 0
`le`	`0011`	Less Than or Equal	C == 1 OR Z == 1
`g`	`0100`	Greater Than	C == 0 AND Z == 0
`ge`	`0101`	Greater Than or Equal	C == 0 OR Z == 1
`no`	`0110`	Not Overflow	O == 0
`o`	`0111`	Overflow	O == 1
`ns`	`1000`	Not Signed	S == 0
`s`	`1001`	Signed	S == 1
`sl`	`1010`	Signed Less Than	S != O
`sle`	`1011`	Signed Less Than or Equal	S != O OR Z == 1
`sg`	`1100`	Signed Greater Than	S == O AND Z == 0
`sge`	`1101`	Signed Greater Than or Equal	S == O
	`1111`	Always	

Operation:

```
if Flags == Condition then
  RA = PC + 3
  PC += (signed)Offset
```

Flags affected: *None*

4.21 CAA: Call Absolute

0	8	9	35
111001100		Location	

Format: `CAA Location`

Purpose: Call to a set absolute location

Description: The current program counter + 4 is stored into RA and the program counter is set to the specified 27-bit location.

Operation:

```
RA = PC + 4
PC = Location
```

Flags affected: *None*

4.22 CAR: Call Relative

0	8	9	35
111001000		Offset	

Format: `CAR Offset`

Purpose: Call to a relative offset

Description: The current program counter + 4 is stored into RA and the 27-bit offset is added to the current program counter.

Operation:

```
RA = PC + 4
PC = PC + Offset
```

Flags affected: *None*

4.23 CM: Compare

0 7	8 12	13 17
10111000	rA	rB

Format: `CM rA, rB`

Purpose: Compare two registers

Description: The 27-bit values in rB is subtracted from rA. The flags are set appropriately.

Operation:

```
FLAGS = FLAGS & ~0xF
28-bit temp = rA - rB
if temp == 0 then
  ZERO FLAG = 1
if temp & 0x4000000 then
  SIGNED FLAG = 1
if (temp & 0x4000000) != (rA & 0x4000000) then
  OVERFLOW FLAG = 1
if (temp & 0x8000000) then
  CARRY FLAG = 1
```

Flags affected: `Z S O C`

4.24 CMF: Compare Floating Point

0 7	8 12	13 17
10111010	rA	rB

Format: `CMF rA, rB`

Purpose: Compare two floating point registers

Description: The 27-bit floating point value in rB is subtracted from the floating point rA. The flags are set appropriately.

Operation:

```
FLAGS = FLAGS & ~0xF
temp = rA - rB
if temp == 0.0 then
  ZERO FLAG = 1
if temp < 0.0 then
  SIGNED FLAG = 1
if (temp < 0.0) != (rA < 0.0) then
  OVERFLOW FLAG = 1
```

Flags affected: `Z S O`

4.25 CMFM: Compare Floating Point Multi Reg

0 7	8 12	13 17
10111110	rA	rB

Format: `CMFM rA, rB`

Purpose: Compare two floating point registers

Description: The 54-bit floating point value in rB is subtracted from the floating point rA. The flags are set appropriately.

Operation:

```
FLAGS = FLAGS & ~0xF
temp = rA:rA+1 - rB:rB+1
if temp == 0 then
  ZERO FLAG = 1
if temp < 0.0 then
  SIGNED FLAG = 1
if (temp < 0.0) != (rA:rA+1 < 0.0) then
  OVERFLOW FLAG = 1
```

Flags affected: `Z S O`

4.26 CMI: Compare Immediate

0 7	8 12	13 26
10111001	rA	Immediate

Format: `CMI rA, IMM`

Purpose: Compare a register and immediate value

Description: The sign extended immediate value is subtracted from the 27-bit rA. The flags are set appropriately.

Operation:

```
FLAGS = FLAGS & ~0xF
28-bit temp = rA - (signed)IMM
if temp == 0 then
  ZERO FLAG = 1
if temp & 0x4000000 then
  SIGNED FLAG = 1
if (temp & 0x4000000) != (rA & 0x4000000) then
  OVERFLOW FLAG = 1
if (temp & 0x8000000) then
  CARRY FLAG = 1
```

Flags affected: `Z S O C`

4.27 CMIM: Compare Immediate Multi Reg

0 7	8 12	13 26
10111101	rA	Immediate

Format: `CMIM rA, IMM`

Purpose: Compare a register and immediate value

Description: The sign extended immediate value is subtracted from the 54-bit rA:rA+1. The flags are set appropriately.

Operation:

```
FLAGS = FLAGS & ~0xF
55-bit temp = rA:rA+1 - (signed)IMM
if temp == 0 then
  ZERO FLAG = 1
if temp & 0x20000000000000 then
  SIGNED FLAG = 1
if ((temp & 0x20000000000000) !=
    (rA:rA+1 & 0x20000000000000)) then
  OVERFLOW FLAG = 1
if (temp & 0x40000000000000) then
  CARRY FLAG = 1
```

Flags affected: `Z S O C`

4.28 CMM: Compare Multi Reg

0 7	8 12	13 17
10111100	rA	rB

Format: `CMM rA, rB`

Purpose: Compare two registers

Description: The 54-bit values in rB:rB+1 is subtracted from rA:rA+1. The flags are set appropriately.

Operation:

```
FLAGS = FLAGS & ~0xF
55-bit temp = rA:rA+1 - rB:rB+1
if temp == 0 then
  ZERO FLAG = 1
if temp & 0x20000000000000 then
  SIGNED FLAG = 1
if ((temp & 0x20000000000000) !=
    (rA:rA+1 & 0x20000000000000)) then
  OVERFLOW FLAG = 1
if (temp & 0x40000000000000) then
  CARRY FLAG = 1
```

Flags affected: `Z S O C`

4.29 CR: Call Register Conditional

0 5	6 9	10 14	15 17
110111	Condition	rA	000

Format: `CRcc rA`

Purpose: Conditional call to a register

Description: If the specified condition is true then the current program counter + 2 is saved into the RA register and the program counter is set to the value in rA.

The condition table is as follows:

cc	Value	Description	Flag State Checked
n	0000	Not Equal / Not Zero	Z == 0
e	0001	Equal / Zero	Z == 1
l	0010	Less Than	C == 1 AND Z == 0
le	0011	Less Than or Equal	C == 1 OR Z == 1
g	0100	Greater Than	C == 0 AND Z == 0
ge	0101	Greater Than or Equal	C == 0 OR Z == 1
no	0110	Not Overflow	O == 0
o	0111	Overflow	O == 1
ns	1000	Not Signed	S == 0
s	1001	Signed	S == 1
sl	1010	Signed Less Than	S != O
sle	1011	Signed Less Than or Equal	S != O OR Z == 1
sg	1100	Signed Greater Than	S == O AND Z == 0
sge	1101	Signed Greater Than or Equal	S == O
	1111	Always	

Operation:

```
if Flags == Condition then
  RA = PC + 2
  PC = rA
```

Flags affected: *None*

4.30 DBRK: Debug Break

0	17
11111111111111111	

Format: `DBRK`

Purpose: Debug break

Description: A special instruction for emulators to allow forcing a break in the emulation. Normal execution will cause an illegal instruction fault.

Operation:

Flags affected: *None*

4.31 DI: Disable Interrupts

0	11	12	16	17
101000000101		rA		0

Format: `DI rA`

Purpose: Disable interrupts based on rA

Description: Disable interrupts based on the mask in rA.

Operation: `FLAGS = (FLAGS & 0x7FFE00F) | ((~rA << 4) & 0x1FF0)`

Flags affected: *None*

4.32 DMT: Direct Memory Transfer

0 6	7 11	12 16	17 21	22 26
0110100	rA	rB	rC	00000

Format: `DMT rA, rB, rC`

Purpose: Directly transfer memory between locations

Description: Directly transfer rC bytes of memory from location pointed to by rB to memory location rA.

Operation:

```
P = 0
while(P < rC)
  (byte)[rA + P] = (byte)[rB + P]
  P = P + 1
```

Flags affected: *None*

4.33 DV: Divide

0 6	7 11	12 16	17 21	22 25	26
0001100	rA	rB	rC	0000	UF

Format: `DV rA, rB, rC`

Purpose: Divide two 27-bit integer registers

Description: The 27-bit value in rB is divided with the 27-bit value in rC, the result is placed in rA.

Operation: `rA` $\leftarrow$ `rB / rC`

Flags affected: `Z C O S`

4.34 DVF: Divide Floating Point

0 6	7 11	12 16	17 21	22 25	26
0001101	rA	rB	rC	0000	UF

Format: `DVF rA, rB, rC`

Purpose: Divide two 27-bit floating point registers

Description: The 27-bit floating point value in rB is divided with the 27-bit floating point value in rC, the result is placed in rA.

Operation: `rA ← rB / rC`

Flags affected: `Z C O S`

4.35 DVFM: Divide Floating Point Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0001111	rA	rB	rC	0000	UF

Format: `DVFM rA, rB, rC`

Purpose: Divide two 54-bit floating point registers

Description: The 54-bit floating point value in rB:rB+1 is divided with the 54-bit floating point value in rC:rC+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 / rC:rC+1`

Flags affected: `Z C O S`

4.36 DVI: Divide Immediate

0 6	7 11	12 16	17 23	24 25	26
0001100	rA	rB	Immediate	01	UF

Format: `DVI rA, rB, IMM`

Purpose: Divide a 27-bit integer register by a 7-bit immediate

Description: The 27-bit value in rB is divided with the 7-bit immediate value, the result is placed in rA.

Operation: `rA ← rB / IMM`

Flags affected: `Z C O S`

4.37 DVIM: Divide Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0001110	rA	rB	Immediate	01	UF

Format: `DVIM rA, rB, IMM`

Purpose: Divide a 54-bit integer register by a 7-bit immediate

Description: The 54-bit value in rB:rB+1 is divided with the 7-bit immediate value, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 / IMM`

Flags affected: `Z C O S`

4.38 DVIS: Divide Immediate Signed

0 6	7 11	12 16	17 23	24 25	26
0001100	rA	rB	Immediate	11	UF

Format: `DVIS rA, rB, IMM`

Purpose: Divide a signed 27-bit integer register by a signed 7-bit immediate

Description: The signed 27-bit value in rB is divided with the signed 7-bit immediate value, the result is placed in rA.

Operation: `rA ← (signed)rB / (signed)IMM`

Flags affected: `Z C O S`

4.39 DVISM: Divide Immediate Signed Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0001110	rA	rB	Immediate	11	UF

Format: `DVISM rA, rB, IMM`

Purpose: Divide a signed 54-bit integer register by a signed 7-bit immediate

Description: The signed 54-bit value in rB:rB+1 is divided with the signed 7-bit immediate value, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (signed)rB:rB+1 / (signed)IMM`

Flags affected: `Z C O S`

4.40 DVM: Divide Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0001110	rA	rB	rC	0000	UF

Format: `DVM rA, rB, rC`

Purpose: Divide two 54-bit integer registers

Description: The 54-bit value in rB:rB+1 is divided with the 54-bit value in rC:rC+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 / rC:rC+1`

Flags affected: `Z C O S`

4.41 DVS: Divide Signed

0 6	7 11	12 16	17 21	22 25	26
0001100	rA	rB	rC	0010	UF

Format: `DVS rA, rB, rC`

Purpose: Divide two signed 27-bit integer registers

Description: The signed 27-bit value in rB is divided with the signed 27-bit value in rC, the result is placed in rA.

Operation: `rA ← (signed)rB / (signed)rC`

Flags affected: `Z C O S`

4.42 DVSM: Divide Signed Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0001110	rA	rB	rC	0010	UF

Format: `DVSM rA, rB, rC`

Purpose: Divide two 54-bit integer registers

Description: The 54-bit value in rB:rB+1 is divided with the 54-bit value in rC:rC+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (signed)rB:rB+1 / (signed)rC:rC+1`

Flags affected: `Z C O S`

4.43 EI: Enable Interrupts

0 11	12 16	17
101000000100	rA	0

Format: `EI rA`

Purpose: Enable interrupts base on rA

Description: Enable interrupts based on the mask in rA.

Operation: `FLAGS = (FLAGS & 0x7FFE00F) | ((rA << 4) & 0x1FF0)`

Flags affected: *None*

4.44 FTI: Float to Integer

0 8	9 13	14 18	19 26
101000101	rA	rB	00000000

Format: `FTI rA, rB`

Purpose: Convert a 27-bit float to integer

Description: The 27-bit float in rB in converted to a 27-bit integer value and stored in rA.

Operation: `rA ← int(rB)`

Flags affected: `Z C O S`

4.45 FTIM: Float to Integer Multi Reg

0 8	9 13	14 18	19 26
101000111	rA	rB	00000000

Format: `FTIM rA, rB`

Purpose: Convert a 54-bit float to integer

Description: The 54-bit float in rB:rB+1 in converted to a 54-bit integer value and stored in rA:rA+1.

Operation: `rA:rA+1 ← int(rB:rB+1)`

Flags affected: `Z C O S`

4.46 HT: Halt

```
0                 17
1010000000011000000
```

Format: `HT`

Purpose: Halt the processor

Description: Disables all interrupts and stops the processor from responding to any more instructions.

Operation:

Flags affected: *None*

4.47 IR: Interrupt Return

```
0                 17
1010000000001000000
```

Format: `IR`

Purpose: Return from an interrupt

Description: Return from an interrupt routine.

Operation:

```
if Stack_Direction_Flag then
  SP -= 0x63
R0:R31 ← [SP:SP+0x60]
FLAGS ← [SP+0x60:SP+0x63]
if not Stack_Direction_Flag then
  SP += 0x63
```

Flags affected: `Z C O S`

4.48 ITF: Integer to Float

0	8	9	13	14	18	19	26
101000100		rA		rB		00000000	

Format: `ITF rA, rB`

Purpose: Convert a 27-bit integer to float

Description: The 27-bit integer in rB in converted to a 27-bit float value and stored in rA.

Operation: `rA` ← `float(rB)`

Flags affected: `Z C O S`

4.49 ITFM: Integer to Float Multi Reg

0	8	9	13	14	18	19	26
101000110		rA		rB		00000000	

Format: `ITFM rA, rB`

Purpose: Convert a 54-bit integer to float

Description: The 54-bit integer in rB:rB+1 in converted to a 54-bit float value and stored in rA:rA+1.

Operation: `rA:rA+1` ← `float(rB:rB+1)`

Flags affected: `Z C O S`

4.50 LDS: Load Single

0 6	7 11	12 16	17 21	22 23	24 50	51 53
1010100	rA	rB	Register Count	Adjust rB	Memory Offset	000

Format: `LDSm rA, [rB + Offset, RegCount]`

Purpose: Load a single byte from memory

Description: The value in rB is added to Offset and a single byte per register is loaded into the low 9 bits of rA to rA+RegCount. The top 18 bits are set to zero for each affected register.

m refers to the Adjust RB value which can have the following designations to indicate the mode:

m	Mode	Description
	0	rB is not adjusted after load
I	1	rB has the number of bytes read added to the start value after all data is read
D	2	rB has the number of bytes to read subtracted from the start value after all data is read

Operation:

```
StartReg = rA
RegCount = RegCount + 1
CurCount = RegCount
Temp = rB
TempPC = PC
if Mode is 2 then
  Temp = Temp - CurCount
MemLocation = (Temp + Offset)
While CurCount is not 0
  Registers[StartReg] = Memory[MemLocation]
  MemLocation += 1
  StartReg = (StartReg + 1) % 32
  CurCount = CurCount - 1
if Mode is 1 then
  rB = rB + RegCount
if Mode is 2 then
  rB = Temp
PC = TempPC
```

Flags affected: *None*

4.51 LDT: Load Tri

0 6	7 11	12 16	17 21	22 23	24 50	51 53
1010110	rA	rB	Register Count	Adjust rB	Memory Offset	000

Format: `LDTm rA, [rB + Offset, RegCount]`

Purpose: Load three bytes from memory

Description: The value in rB is added to Offset and three bytes per register are loaded into rA to rA+RegCount.

m refers to the Adjust RB value which can have the following designations to indicate the mode:

m	Mode	Description
	0	rB is not adjusted after load
I	1	rB has the number of bytes read added to the start value after all data is read
D	2	rB has the number of bytes to read subtracted from the start value after all data is read

Operation:

```
StartReg = rA
RegCount = RegCount + 1
CurCount = RegCount
Temp = rB
TempPC = PC
if Mode is 2 then
  Temp = Temp - (CurCount * 3)
MemLocation = (Temp + Offset)
While CurCount is not 0
  Registers[StartReg] =
    (Memory[MemLocation] << 9) |
    (Memory[(MemLocation + 1)] << 18) |
    Memory[(MemLocation + 2)]
  MemLocation += 3
  StartReg = (StartReg + 1) % 32
  CurCount = CurCount - 1
if Mode is 1 then
  rB = rB + (RegCount * 3)
if Mode is 2 then
  rB = Temp
PC = TempPC
```

Flags affected: *None*

4.52 LDW: Load Word

0 6	7 11	12 16	17 21	22 23	24 50	51 53
1010101	rA	rB	Register Count	Adjust rB	Memory Offset	000

Format: `LDWm rA, [rB + Offset, RegCount]`

Purpose: Load two bytes from memory

Description: The value in rB is added to Offset and two bytes per register are loaded into the low 18 bits of rA to rA+RegCount. The top 9 bits are set to zero for each affected register.

m refers to the Adjust RB value which can have the following designations to indicate the mode:

m	Mode	Description
	0	rB is not adjusted after load
I	1	rB has the number of bytes read added to the start value after all data is read
D	2	rB has the number of bytes to read subtracted from the start value after all data is read

Operation:

```
StartReg = rA
RegCount = RegCount + 1
CurCount = RegCount
Temp = rB
TempPC = PC
if Mode is 2 then
  Temp = Temp - (CurCount * 2)
MemLocation = (Temp + Offset)
While CurCount is not 0
  Registers[StartReg] =
    (Memory[MemLocation] << 9) | Memory[(MemLocation + 1)]
  MemLocation += 2
  StartReg = (StartReg + 1) % 32
  CurCount = CurCount - 1
if Mode is 1 then
  rB = rB + (RegCount * 2)
if Mode is 2 then
  rB = Temp
PC = TempPC
```

Flags affected: *None*

4.53 MD: Modulus

0 6	7 11	12 16	17 21	22 25	26
0010000	rA	rB	rC	0000	UF

Format: `MD rA, rB, rC`

Purpose: Access the remainder of dividing two 27-bit integer registers

Description: The 27-bit value in rB is divided with the 27-bit value in rC, the remainder of the division is placed in rA.

Operation: `rA ← rB % rC`

Flags affected: `Z C O S`

4.54 MDF: Modulus Floating Point

0 6	7 11	12 16	17 21	22 25	26
0010001	rA	rB	rC	0000	UF

Format: `MDF rA, rB, rC`

Purpose: Access the remainder of dividing two 27-bit floating point registers

Description: The 27-bit floating point value in rB is divided with the 27-bit floating point value in rC, the remainder of the division is placed in rA.

Operation: `rA ← rB % rC`

Flags affected: `Z C O S`

4.55 MDFM: Modulus Floating Point Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0010011	rA	rB	rC	0000	UF

Format: `MDFM rA, rB, rC`

Purpose: Access the remainder of dividing two 54-bit floating point registers

Description: The 54-bit floating point value in rB:rB+1 is divided with the 54-bit floating point value in rC:rC+1, the remainder of the division is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 % rC:rC+1`

Flags affected: `Z C O S`

4.56 MDI: Modulus Immediate

0 6	7 11	12 16	17 23	24 25	26
0010000	rA	rB	Immediate	01	UF

Format: `MDI rA, rB, IMM`

Purpose: Access the remainder of dividing a 27-bit integer register by a 7-bit immediate

Description: The 27-bit value in rB is divided with a 7-bit immediate value, the remainder of the division is placed in rA.

Operation: `rA ← rB % IMM`

Flags affected: `Z C O S`

4.57 MDIM: Modulus Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0010010	rA	rB	Immediate	01	UF

Format: `MDIM rA, rB, IMM`

Purpose: Access the remainder of dividing a 54-bit integer register by a 7-bit immediate

Description: The 54-bit value in rB:rB+1 is divided with the 7-bit immediate value, the remainder of the division is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 % IMM`

Flags affected: `Z C O S`

4.58 MDIS: Modulus Immediate Signed

0 6	7 11	12 16	17 23	24 25	26
0010000	rA	rB	Immediate	11	UF

Format: `MDIS rA, rB, IMM`

Purpose: Access the remainder of dividing a signed 27-bit integer register by a signed 7-bit immediate

Description: The signed 27-bit value in rB is divided with a signed 7-bit immediate value, the remainder of the division is placed in rA.

Operation: `rA ← (signed)rB % (signed)IMM`

Flags affected: `Z C O S`

4.59 MDISM: Modulus Immediate Signed Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0010010	rA	rB	Immediate	11	UF

Format: `MDISM rA, rB, IMM`

Purpose: Access the remainder of dividing a signed 54-bit integer register by a signed 7-bit immediate

Description: The signed 54-bit value in rB:rB+1 is divided with the signed 7-bit immediate value, the remainder of the division is placed in rA:rA+1.

Operation: `rA:rA+1 ← (signed)rB:rB+1 % (signed)IMM`

Flags affected: `Z C O S`

4.60 MDM: Modulus Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0010010	rA	rB	rC	0000	UF

Format: `MDM rA, rB, rC`

Purpose: Access the remainder of dividing two 54-bit integer registers

Description: The 54-bit value in rB:rB+1 is divided with the 54-bit value in rC:rC+1, the remainder of the division is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 % rC:rC+1`

Flags affected: `Z C O S`

4.61 MDS: Modulus Signed

0 6	7 11	12 16	17 21	22 25	26
0010000	rA	rB	rC	0010	UF

Format: `MDS rA, rB, rC`

Purpose: Access the remainder of dividing two signed 27-bit integer registers

Description: The signed 27-bit value in rB is divided with the signed 27-bit value in rC, the remainder of the division is placed in rA.

Operation: `rA ← (signed)rB % (signed)rC`

Flags affected: `Z C O S`

4.62 MDSM: Modulus Signed Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0010010	rA	rB	rC	0010	UF

Format: `MDSM rA, rB, rC`

Purpose: Access the remainder of dividing two signed 54-bit integer registers

Description: The signed 54-bit value in rB:rB+1 is divided with the signed 54-bit value in rC:rC+1, the remainder of the division is placed in rA:rA+1.

Operation: `rA:rA+1 ← (signed)rB:rB+1 % (signed)rC:rC+1`

Flags affected: `Z C O S`

4.63 MH: Move High

0 4	5 9	10 26
10001	rA	Immediate

Format: `MH rA, IMM`

Purpose: Move an immediate value to the high bits of a register

Description: The high 17 bits of a 27-bit value in rA is set to the specified immediate value.

Operation: `rA ← (IMM << 10) | (rA & 0x3FF)`

Flags affected: `Z C O S`

4.64 ML: Move Low

0 4	5 9	10 26
10010	rA	Immediate

Format: `ML rA, IMM`

Purpose: Move a 17-bit immediate value to the register

Description: A 27-bit rA is set to the specified 17-bit immediate value. The high bits are zero'd out.

Operation: `rA ← IMM`

Flags affected: `Z C O S`

4.65 MS: Move Low Signed

0 4	5 9	10 26
10011	rA	Immediate

Format: `MS rA, IMM`

Purpose: Move a signed 17-bit immediate value to the register

Description: A 27-bit rA is set to the specified signed 17-bit immediate value. The high bits are set based on the signed bit of the immediate value specified.

Operation: `rA ← (signed)IMM`

Flags affected: `Z C O S`

4.66 MU: Multiply

0 6	7 11	12 16	17 21	22 25	26
0001000	rA	rB	rC	0000	UF

Format: `MU rA, rB, rC`

Purpose: Multiply two 27-bit integer registers together

Description: The 27-bit value in rC is multiplied with the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB * rC`

Flags affected: `Z C O S`

4.67 MUF: Multiply Floating Point

0 6	7 11	12 16	17 21	22 25	26
0001001	rA	rB	rC	0000	UF

Format: `MUF rA, rB, rC`

Purpose: Multiply two 27-bit floating point registers together

Description: The 27-bit floating point value in rC is multiplied with the 27-bit floating point value in rB, the result is placed in rA.

Operation: `rA ← rB * rC`

Flags affected: `Z C O S`

4.68 MUFM: Multiply Floating Point Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0001011	rA	rB	rC	0000	UF

Format: `MUFM rA, rB, rC`

Purpose: Multiply two 54-bit floating point registers together

Description: The 54-bit floating point value in rC:rC+1 is multiplied with the 54-bit floating point value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 * rC:rC+1`

Flags affected: `Z C O S`

4.69 MUI: Multiply Immediate

0 6	7 11	12 16	17 23	24 25	26
0001000	rA	rB	Immediate	01	UF

Format: `MUI rA, rB, IMM`

Purpose: Multiply a 27-bit integer register by a 7-bit immediate

Description: The 7-bit immediate value is multiplied with the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB * IMM`

Flags affected: `Z C O S`

4.70 MUIM: Multiply Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0001010	rA	rB	Immediate	01	UF

Format: `MUIM rA, rB, IMM`

Purpose: Multiply a 54-bit integer register by a 7-bit immediate

Description: The 7-bit immediate value is multiplied with the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 * IMM`

Flags affected: `Z C O S`

4.71 MUIS: Multiply Immediate Signed

0 6	7 11	12 16	17 23	24 25	26
0001000	rA	rB	Immediate	11	UF

Format: `MUIS rA, rB, IMM`

Purpose: Multiply a signed 27-bit integer register by a signed 7-bit immediate

Description: The signed 7-bit immediate value is multiplied with the signed 27-bit value in rB, the result is placed in rA.

Operation: `rA ← (signed)rB * (signed)IMM`

Flags affected: `Z C O S`

4.72 MUISM: Multiply Immediate Signed Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0001010	rA	rB	Immediate	11	UF

Format: `MUISM rA, rB, IMM`

Purpose: Multiply a signed 54-bit integer register by a signed 7-bit immediate

Description: The signed 7-bit immediate value is multiplied with the signed 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (signed)rB:rB+1 * (signed)IMM`

Flags affected: `Z C O S`

4.73 MUM: Multiply Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0001010	rA	rB	rC	0000	UF

Format: `MUM rA, rB, rC`

Purpose: Multiply two 54-bit integer registers together

Description: The 54-bit value in rC:rC+1 is multiplied with the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 * rC:rC+1`

Flags affected: `Z C O S`

4.74 MUS: Multiply Signed

0 6	7 11	12 16	17 21	22 25	26
0001000	rA	rB	rC	0010	UF

Format: `MUS rA, rB, rC`

Purpose: Multiply two signed 27-bit integer registers together

Description: The signed 27-bit value in rC is multiplied with the signed 27-bit value in rB, the result is placed in rA.

Operation: `rA ← (signed)rB * (signed)rC`

Flags affected: `Z C O S`

4.75 MUSM: Multiply Signed Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0001010	rA	rB	rC	0010	UF

Format: `MUSM rA, rB, rC`

Purpose: Multiply two 54-bit integer registers together

Description: The signed 54-bit value in rC:rC+1 is multiplied with the signed 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (signed)rB:rB+1 * (signed)rC:rC+1`

Flags affected: `Z C O S`

4.76 NG: Negate

0 8	9 13	14 18	19 25	26
101001100	rA	rB	0000000	UF

Format: `NG rA, rB`

Purpose: Negate a 27-bit register

Description: Negate the 27-bit rB register value and store the result in rA.

Operation: `rA ← -rB`

Flags affected: `Z C O S`

4.77 NGF: Negate Floating Point

0 8	9 13	14 18	19 25	26
101001101	rA	rB	0000000	UF

Format: `NGF rA, rB`

Purpose: Negate a 27-bit floating-point register

Description: Negate the 27-bit rB floating-point register value and store the result in rA.

Operation: `rA ← -rB`

Flags affected: `Z S`

4.78 NGFM: Negate Floating Point Multi Reg

0 8	9 13	14 18	19 25	26
101001111	rA	rB	0000000	UF

Format: `NGFM rA, rB`

Purpose: Negate a 54-bit floating-point register

Description: Negate the 54-bit rB:rB+1 floating-point register value and store the result in rA:rA+1.

Operation: `rA:rA+1 ← -rB:rB+1`

Flags affected: `Z S`

4.79 NGM: Negate Multi Reg

0 8	9 13	14 18	19 25	26
101001110	rA	rB	0000000	UF

Format: `NGM rA, rB`

Purpose: Negate a 54-bit register

Description: Negate the 54-bit rB:rB+1 register value and store the result in rA:rA+1.

Operation: `rA:rA+1 ← -rB:rB+1`

Flags affected: `Z C O S`

4.80 NT: Not

0 8	9 13	14 18	19 25	26
101001100	rA	rB	0100000	UF

Format: `NT rA, rB`

Purpose: Bit test a 27-bit register

Description: Test all bits of the 27-bit rB register value. If any bits are set then rA is set to 0. If all bits are off then rA is set to 1.

Operation: `rA ← !rB`

Flags affected: `Z C O S`

4.81 NTM: Not Multi Reg

0 8	9 13	14 18	19 25	26
101001110	rA	rB	0100000	UF

Format: `NTM rA, rB`

Purpose: Bit test a 54-bit register

Description: Test all bits of the 54-bit rB:rB+1 register value. If any bits are set then rA:rA+1 is set to 0. If all bits are off then rA:rA+1 is set to 1.

Operation: `rA:rA+1 ← !rB:rB+1`

Flags affected: `Z C O S`

4.82 OR: Or

0 6	7 11	12 16	17 21	22 25	26
0011000	rA	rB	rC	0000	UF

Format: `OR rA, rB, rC`

Purpose: Bit-wise OR two 27-bit integer registers together

Description: The 27-bit value in rC is bit-wise OR to the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB | rC`

Flags affected: `Z C O S`

4.83 ORI: Or Immediate

0 6	7 11	12 16	17 23	24 25	26
0011000	rA	rB	Immediate	01	UF

Format: `ORI rA, rB, IMM`

Purpose: Bit-wise OR a 27-bit integer register and 7-bit immediate together

Description: The 7-bit immediate value is bit-wise OR to the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB | IMM`

Flags affected: `Z C O S`

4.84 ORM: Or Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0011010	rA	rB	rC	0000	UF

Format: `ORM rA, rB, rC`

Purpose: Bit-wise OR two 54-bit integer registers together

Description: The 54-bit value in rC:rC+1 is bit-wise OR to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 | rC:rC+1`

Flags affected: `Z C O S`

4.85 RE: Return

0 17
101000000000000000

Format: `RE`

Purpose: To return from a call

Description: To return from a call.

Operation: `PC ← RETADDR`

Flags affected: *None*

4.86 RF: Read Flags

0 11	12 16	17
101000001100	rA	0

Format: `RF rA`

Purpose: Read flags into the rA register

Description: Set rA to the value in the flag register.

Operation: `rA = FLAGS`

Flags affected: *None*

4.87 RL: Rotate Left

0 6	7 11	12 16	17 21	22 25	26
0110000	rA	rB	rC	0000	UF

Format: `RL rA, rB, rC`

Purpose: Rotate left a 27-bit integer

Description: The 27-bit value in rB is left rotated the number of bits specified in rC. The result is placed in rA.

Operation: `rA ← (rB << rC) | (rB >> (27-rC))`

Flags affected: `Z C O S`

4.88 RLI: Rotate Left Immediate

0 6	7 11	12 16	17 23	24 25	26
1000000	rA	rB	Immediate	00	UF

Format: `RLI rA, rB, IMM`

Purpose: Rotate left a 27-bit integer by a 7-bit immediate value

Description: The 27-bit value in rB is left rotated by the 7-bit immediate value. The result is placed in rA.

Operation: `rA ← (rB << IMM) | (rB >> (27-IMM))`

Flags affected: `Z C O S`

4.89 RLIM: Rotate Left Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
1000010	rA	rB	Immediate	00	UF

Format: `RLIM rA, rB, IMM`

Purpose: Rotate left a 54-bit integer by a 7-bit immediate value

Description: The 54-bit value in rB:rB+1 is left rotated by the 7-bit immediate value. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (rB:rB+1 << IMM) | (rB:rB+1 >> (54-IMM))`

Flags affected: `Z C O S`

4.90 RLM: Rotate Left Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0110010	rA	rB	rC	0000	UF

Format: `RLM rA, rB, rC`

Purpose: Rotate left a 54-bit integer

Description: The 54-bit value in rB:rB+1 is left rotated the number of bits specified in rC. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (rB:rB+1 << rC) | (rB:rB+1 >> (54-rC))`

Flags affected: `Z C O S`

4.91 RMP: Read Memory Protection

0 6	7 11	12 16	17 26
1010010	rA	rB	0000000000

Format: `RMP rA, rB`

Purpose: Read the memory protections for a page of memory

Description: rA is set to the page protection flags of the pages of the specified memory range starting at the memory location specified by rA and moving forward rB pages. A set zero flag indicates success, upon failure rA holds the error reason. Every 2 bits indicates the memory protection state of a page. Only 13 page statuses can be returned at a time.

Errors:

Reason	Description
0	Memory start is not page aligned
1	Too many pages specified

Flags:

Flag	Description
0	No access
1	Read
2	Read/Write
3	Read/Execute

Operation:

```
rA ←
    Memory_Page_Flags[rA]:Memory_Page_Flags[rA+(rB*1024)]
```

Flags affected: `Z`

4.92 RND: Random

0 8	9 13	14 25	26
101001100	rA	000001100000	UF

Format: `RND rA`

Purpose: Generate a random value into rA

Description: A random value is placed into the 27-bit rA register.

Operation: `rA ← <random value>`

Flags affected: `Z C O S`

4.93 RNDM: Random Multi Reg

0 8	9 13	14 25	26
101001110	rA	000001100000	UF

Format: `RNDM rA`

Purpose: Generate a random value into rA:rA+1

Description: A random value is generated and placed into the 54-bit rA:rA+1 register.

Operation: `rA:rA+1 ← <random value>`

Flags affected: `Z C O S`

4.94 RR: Rotate Right

0 6	7 11	12 16	17 21	22 25	26
0110001	rA	rB	rC	0000	UF

Format: `RR rA, rB, rC`

Purpose: Rotate right a 27-bit integer

Description: The 27-bit value in rB is right rotated the number of bits specified in rC. The result is placed in rA.

Operation: `rA ← (rB >> rC) | (rB << (27-rC))`

Flags affected: `Z C O S`

4.95 RRI: Rotate Right Immediate

0 6	7 11	12 16	17 23	24 25	26
1000001	rA	rB	Immediate	00	UF

Format: `RRI rA, rB, IMM`

Purpose: Rotate right a 27-bit integer by a 7-bit immediate value

Description: The 27-bit value in rB is right rotated by the 7-bit immediate value. The result is placed in rA.

Operation: `rA ← (rB >> IMM) | (rB << (27-IMM))`

Flags affected: `Z C O S`

4.96 RRIM: Rotate Right Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
1000011	rA	rB	Immediate	00	UF

Format: `RRIM rA, rB, rC`

Purpose: Rotate right a 54-bit integer by an immediate value

Description: The 54-bit value in rB:rB+1 is right rotated by the 7-bit immediate value. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (rB:rB+1 >> IMM) | (rB:rB+1 << (54-IMM))`

Flags affected: `Z C O S`

4.97 RRM: Rotate Right Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0110011	rA	rB	rC	0000	UF

Format: `RRM rA, rB, rC`

Purpose: Rotate right a 54-bit integer

Description: The 54-bit value in rB:rB+1 is right rotated the number of bits specified in rC. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (rB:rB+1 >> rC) | (rB:rB+1 << (54-rC))`

Flags affected: `Z C O S`

4.98 SA: Shift Arithemetic Right

0 6	7 11	12 16	17 21	22 25	26
0101101	rA	rB	rC	0000	UF

Format: `SA rA, rB, rC`

Purpose: Shift right arithmetic a 27-bit integer

Description: The 27-bit value in rB is right arithmetic shifted the number of bits specified in rC. The result is placed in rA.

Operation: `rA ← (signed)rB >> rC`

Flags affected: `Z C O S`

4.99 SAI: Shift Arithemetic Right Immediate

0 6	7 11	12 16	17 23	24 25	26
0111101	rA	rB	Immediate	00	UF

Format: `SAI rA, rB, IMM`

Purpose: Shift right arithmetic a 27-bit integer by a 7-bit immediate value

Description: The 27-bit value in rB is right arithmetic shifted by the 7-bit immediate value. The result is placed in rA.

Operation: `rA ← (signed)rB >> IMM`

Flags affected: `Z C O S`

4.100 SAIM: Shift Arithemetic Right Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0111111	rA	rB	Immediate	00	UF

Format: `SAIM rA, rB, IMM`

Purpose: Shift right arithmetic a 54-bit integer by a 7-bit immediate value

Description: The 54-bit value in rB:rB+1 is right arithmetic shifted by the 7-bit immediate value. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (signed)rB:rB+1 >> IMM`

Flags affected: `Z C O S`

4.101 SAM: Shift Arithemetic Right Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0101111	rA	rB	rC	0000	UF

Format: `SAM rA, rB, rC`

Purpose: Shift right arithmetic a 54-bit integer

Description: The 54-bit value in rB:rB+1 is right arithmetic shifted the number of bits specified in rC. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← (signed)rB:rB+1 >> rC`

Flags affected: `Z C O S`

4.102 SB: Subtract

0 6	7 11	12 16	17 21	22 25	26
0000100	rA	rB	rC	0000	UF

Format: `SB rA, rB, rC`

Purpose: Subtract two 27-bit integer registers from each other

Description: The 27-bit value in rC is subtracted from the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB - rC`

Flags affected: `Z C O S`

4.103 SBC: Subtract With Carry

0 6	7 11	12 16	17 21	22 25	26
0100100	rA	rB	rC	0000	UF

Format: `SBC rA, rB, rC`

Purpose: Subtract two 27-bit integer registers from each other including the carry bit

Description: The 27-bit value in rC is subtracted from the 27-bit value in rB, the carry bit from any previous operation is subtracted from the result. The result is placed in rA.

Operation: `rA ← rB - rC - Carry_Bit`

Flags affected: `Z C O S`

4.104 SBCI: Subtract Immediate With Carry

0 6	7 11	12 16	17 23	24 25	26
0100100	rA	rB	Immediate	01	UF

Format: `SBCI rA, rB, IMM`

Purpose: Subtract a 7-bit immediate from a 27-bit integer register including the carry bit

Description: The 7-bit immediate value is subtracted from the 27-bit value in rB, the carry bit from any previous operation is subtracted from the result. The result is placed in rA.

Operation: `rA ← rB - IMM - Carry_Bit`

Flags affected: `Z C O S`

4.105 SBCIM: Subtract Immediate Multi Reg With Carry

0 6	7 11	12 16	17 23	24 25	26
0100110	rA	rB	Immediate	01	UF

Format: `SBCIM rA, rB, IMM`

Purpose: Subtract a 7-bit immediate from a 54-bit integer register including the carry bit

Description: The 7-bit immediate value is subtracted from the 54-bit value in rB:rB+1, the carry bit from any previous operation is subtracted from the result. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 - IMM - Carry_Bit`

Flags affected: `Z C O S`

4.106 SBCM: Subtract Multi Reg With Carry

0 6	7 11	12 16	17 21	22 25	26
0100110	rA	rB	rC	0000	UF

Format: `SBCM rA, rB, rC`

Purpose: Subtracted two 54-bit integer registers from each other including the carry bit

Description: The 54-bit value in rC:rC+1 is subtracted from the 54-bit value in rB:rB+1, the carry bit from any previous operation is subtracted from the result. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 - rC:rC+1 - Carry_Bit`

Flags affected: `Z C O S`

4.107 SBF: Subtract Floating Point

0 6	7 11	12 16	17 21	22 25	26
0000101	rA	rB	rC	0000	UF

Format: `SBF rA, rB, rC`

Purpose: Subtract two 27-bit floating point registers from each other

Description: The 27-bit floating point value in rC is subtracted from to the 27-bit floating point value in rB, the result is placed in rA.

Operation: `rA ← rB - rC`

Flags affected: `Z C O S`

4.108 SBFM: Subtract Floating Point Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0000111	rA	rB	rC	0000	UF

Format: `SBFM rA, rB, rC`

Purpose: Subtract two 54-bit floating point registers from each other

Description: The 54-bit floating point value in rC:rC+1 is subtracted from the 54-bit floating point value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 - rC:rC+1`

Flags affected: `Z C O S`

4.109 SBI: Subtract Immediate

0 6	7 11	12 16	17 23	24 25	26
0000100	rA	rB	Immediate	01	UF

Format: `SBI rA, rB, IMM`

Purpose: Subtract a 7-bit immediate value from a 27-bit integer register

Description: The 7-bit immediate value is subtracted from the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB - IMM`

Flags affected: `Z C O S`

4.110 SBIM: Subtract Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0000110	rA	rB	Immediate	01	UF

Format: `SBIM rA, rB, IMM`

Purpose: Subtract a 7-bit immediate value from a 54-bit integer register

Description: The 7-bit immediate value is subtracted from the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 - IMM`

Flags affected: `Z C O S`

4.111 SBM: Subtract Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0000110	rA	rB	rC	0000	UF

Format: `SBM rA, rB, rC`

Purpose: Subtracted two 54-bit integer registers from each other

Description: The 54-bit value in rC:rC+1 is subtracted from the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 - rC:rC+1`

Flags affected: `Z C O S`

4.112 SES: Sign Extend Single

0 11	12 16	17 21	22 26
101000000111	rA	rB	00000

Format: `SES rA, rB`

Purpose: Sign extend from a byte

Description: Sign extend a single byte from register rB into the 27-bit rA.

Operation: `rA ← (signed)(rB << 18) >> 18`

Flags affected: `Z C O S`

4.113 SEW: Sign Extend Word

0 11	12 16	17 21	22 26
101000001000	rA	rB	00000

Format: `SEW rA, rB`

Purpose: Sign extend from two bytes

Description: Sign extend a 18-bit value from register rB into the 27-bit rA.

Operation: `rA ← (signed)(rB << 9) >> 9`

Flags affected: `Z C O S`

4.114 SF: Set Flags

0 11	12 16	17
101000001011	rA	0

Format: `SF rA`

Purpose: Set flag register to rA

Description: Set the flag register to the value in rA.

Operation: `FLAGS = rA & 0x7FFFFFF`

Flags affected: `Z C O S`

4.115 SL: Shift Left

0 6	7 11	12 16	17 21	22 25	26
0101000	rA	rB	rC	0000	UF

Format: `SL rA, rB, rC`

Purpose: Shift left a 27-bit integer

Description: The 27-bit value in rB is left shifted the number of bits specified in rC. The result is placed in rA.

Operation: `rA ← rB << rC`

Flags affected: `Z C O S`

4.116 SLI: Shift Left Immediate

0 6	7 11	12 16	17 23	24 25	26
0111000	rA	rB	Immediate	00	UF

Format: `SLI rA, rB, IMM`

Purpose: Shift left a 27-bit integer by a 7-bit immediate value

Description: The 27-bit value in rB is left shifted by the 7-bit immediate value. The result is placed in rA.

Operation: `rA ← rB << IMM`

Flags affected: `Z C O S`

4.117 SLIM: Shift Left Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0111010	rA	rB	Immediate	00	UF

Format: `SLIM rA, rB, IMM`

Purpose: Shift left a 54-bit integer by a 7-bit immediate value

Description: The 54-bit value in rB is left shifted by the 7-bit immediate value. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 << IMM`

Flags affected: `Z C O S`

4.118 SLM: Shift Left Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0101010	rA	rB	rC	0000	UF

Format: `SLM rA, rB, rC`

Purpose: Shift left a 54-bit integer

Description: The 54-bit value in rB:rB+1 is left shifted the number of bits specified in rC. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 << rC`

Flags affected: `Z C O S`

4.119 SMP: Set Memory Protection

0 6	7 11	12 16	17	18 19	20 26
1010010	rA	rB	1	Memory Flags	0000000

Format: `SMP rA, rB, FLAGS`

Purpose: Set the memory protections for a range of pages of memory

Description: Memory starting at a page boundary specified by rA has its page protection flags set to FLAGS for rB pages. A set zero flag indicates success. Upon failure rA holds the error reason.

Errors:

Reason	Description
0	Memory start is not page aligned
1	Too many pages specified

Flags:

Flag	Description
0	No access
1	Read
2	Read/Write
3	Read/Execute

Operation:

Flags affected: `Z`

4.120 SR: Shift Right

0 6	7 11	12 16	17 21	22 25	26
0101001	rA	rB	rC	0000	UF

Format: `SR rA, rB, rC`

Purpose: Shift right a 27-bit integer

Description: The 27-bit value in rB is right shifted the number of bits specified in rC. The result is placed in rA.

Operation: `rA ← rB >> rC`

Flags affected: `Z C O S`

4.121 SRI: Shift Right Immediate

0 6	7 11	12 16	17 23	24 25	26
0111001	rA	rB	Immediate	00	UF

Format: `SRI rA, rB, IMM`

Purpose: Shift right a 27-bit integer by a 7-bit immediate value

Description: The 27-bit value in rB is right shifted by the 7-bit immediate value. The result is placed in rA.

Operation: `rA ← rB >> IMM`

Flags affected: `Z C O S`

4.122 SRIM: Shift Right Immediate Multi Reg

0 6	7 11	12 16	17 23	24 25	26
0111011	rA	rB	Immediate	00	UF

Format: `SRIM rA, rB, IMM`

Purpose: Shift right a 54-bit integer by a 7-bit immediate value

Description: The 54-bit value in rB:rB+1 is right shifted by the 7-bit immediate value. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 >> IMM`

Flags affected: `Z C O S`

4.123 SRM: Shift Right Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0101011	rA	rB	rC	0000	UF

Format: `SRM rA, rB, rC`

Purpose: Shift right a 54-bit integer

Description: The 54-bit value in rB:rB+1 is right shifted the number of bits specified in rC. The result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 >> rC`

Flags affected: `Z C O S`

4.124 STS: Store Single

0 6	7 11	12 16	17 21	22 23	24 50	51 53
1011000	rA	rB	Register Count	Adjust rB	Memory Offset	000

Format: `STSm rA, [rB + Offset, RegCount]`

Purpose: Store a single byte into memory

Description: The value in rB is added to Offset and a single byte per register is stored from the low 9 bits of each register from rA to rA+RegCount. m refers to the Adjust RB value which can have the following designations to indicate the mode:

m	Mode	Description
	0	rB is not adjusted after store
I	1	rB has the number of bytes written added to the start value after all data is written
D	2	rB has the number of bytes to write subtracted from the start value after all data is written

Operation:

```
StartReg = rA
RegCount = RegCount + 1
CurCount = RegCount
Temp = rB
if Mode is 2 then
  Temp = Temp - CurCount
MemLocation = (Temp + Offset)
While CurCount is not 0
  Memory[MemLocation] = Registers[StartReg] & 0x1ff
  MemLocation += 1
  StartReg = (StartReg + 1) % 32
  CurCount = CurCount - 1
if Mode is 1 then
  rB = rB + RegCount
if Mode is 2 then
  rB = Temp
```

Flags affected: *None*

4.125 STT: Store Tri

0 6	7 11	12 16	17 21	22 23	24 50	51 53
1011010	rA	rB	Register Count	Adjust rB	Memory Offset	000

Format: `STTm rA, [rB + Offset, RegCount]`

Purpose: Store three bytes into memory

Description: The value in rB is added to Offset and a three bytes per register are stored for each register from rA to rA+RegCount. m refers to the Adjust RB value which can have the following designations to indicate the mode:

m	Mode	Description
	0	rB is not adjusted after store
I	1	rB has the number of bytes written added to the start value after all data is written
D	2	rB has the number of bytes to write subtracted from the start value after all data is written

Operation:

```
StartReg = rA
RegCount = RegCount + 1
CurCount = RegCount
Temp = rB
if Mode is 2 then
  Temp = Temp - (CurCount * 3)
MemLocation = (Temp + Offset)
While CurCount is not 0
  Memory[MemLocation] =
    (Registers[StartReg] >> 9) & 0x1ff
  MemLocation += 1
  Memory[MemLocation] =
    (Registers[StartReg] >> 18) & 0x1ff
  MemLocation += 1
  Memory[MemLocation] =
    Registers[StartReg] &  0x1ff
  MemLocation += 1
  StartReg = (StartReg + 1) % 32
  CurCount = CurCount - 1
if Mode is 1 then
  rB = rB + (RegCount * 3)
if Mode is 2 then
  rB = Temp
```

Flags affected: *None*

4.126 STW: Store Word

0 6	7 11	12 16	17 21	22 23	24 50	51 53
1011001	rA	rB	Register Count	Adjust rB	Memory Offset	000

Format: `STWm rA, [rB + Offset, RegCount]`

Purpose: Store two bytes into memory

Description: The value in rB is added to Offset and a two bytes per register are stored from the low 18 bits of each register from rA to rA+RegCount. m refers to the Adjust RB value which can have the following designations to indicate the mode:

m	Mode	Description
	0	rB is not adjusted after store
I	1	rB has the number of bytes written added to the start value after all data is written
D	2	rB has the number of bytes to write subtracted from the start value after all data is written

Operation:

```
StartReg = rA
RegCount = RegCount + 1
CurCount = RegCount
Temp = rB
if Mode is 2 then
  Temp = Temp - (CurCount * 2)
MemLocation = (Temp + Offset)
While CurCount is not 0
  Memory[MemLocation] = (Registers[StartReg] >> 9) & 0x1ff
  MemLocation = (MemLocation + 1)
  Memory[MemLocation] = Registers[StartReg] & 0x1ff
  MemLocation = (MemLocation + 1)
  StartReg = (StartReg + 1) % 32
  CurCount = CurCount - 1
if Mode is 1 then
  rB = rB + (RegCount * 2)
if Mode is 2 then
  rB = Temp
```

Flags affected: *None*

4.127 WT: Wait

0	17
101000000010000000	

Format: `WT`

Purpose: Wait for interrupt

Description: Pauses the processor until an interrupt fires.

Operation:

Flags affected: *None*

4.128 XR: Xor

0 6	7 11	12 16	17 21	22 25	26
0011100	rA	rB	rC	0000	UF

Format: `XR rA, rB, rC`

Purpose: Bit-wise eXclusive OR two 27-bit integer registers together

Description: The 27-bit value in rC is bit-wise eXclusive OR to the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB ^ rC`

Flags affected: `Z C O S`

4.129 XRI: Xor Immediate

0 6	7 11	12 16	17 23	24 25	26
0011100	rA	rB	Immediate	01	UF

Format: `XRI rA, rB, IMM`

Purpose: Bit-wise Exclusive OR a 27-bit integer register and 7-bit immediate together

Description: The 7-bit immediate value is bit-wise Exclusive OR to the 27-bit value in rB, the result is placed in rA.

Operation: `rA ← rB ^ IMM`

Flags affected: `Z C O S`

4.130 XRM: Xor Multi Reg

0 6	7 11	12 16	17 21	22 25	26
0011110	rA	rB	rC	0000	UF

Format: `XRM rA, rB, rC`

Purpose: Bit-wise eXclusive OR two 54-bit integer registers together

Description: The 54-bit value in rC:rC+1 is bit-wise eXclusive OR to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: `rA:rA+1 ← rB:rB+1 ^ rC:rC+1`

Flags affected: `Z C O S`

4.131 ZES: Zero Extend Single

0 11	12 16	17 21	22 26
101000001001	rA	rB	00000

Format: `ZES rA, rB`

Purpose: Zero extend from a byte

Description: Zero extend a single byte from register rB into the 27-bit rA.

Operation: `rA ← rB & 0x00001FF`

Flags affected: `Z C O S`

4.132 ZEW: Zero Extend Word

0 11	12 16	17 21	22 26
101000001010	rA	rB	00000

Format: `ZEW rA, rB`

Purpose: Zero extend from two bytes

Description: Zero extend a 18-bit value from register rB into the 27-bit rA.

Operation: `rA ← rB & 0x003FFFF`

Flags affected: `Z C O S`