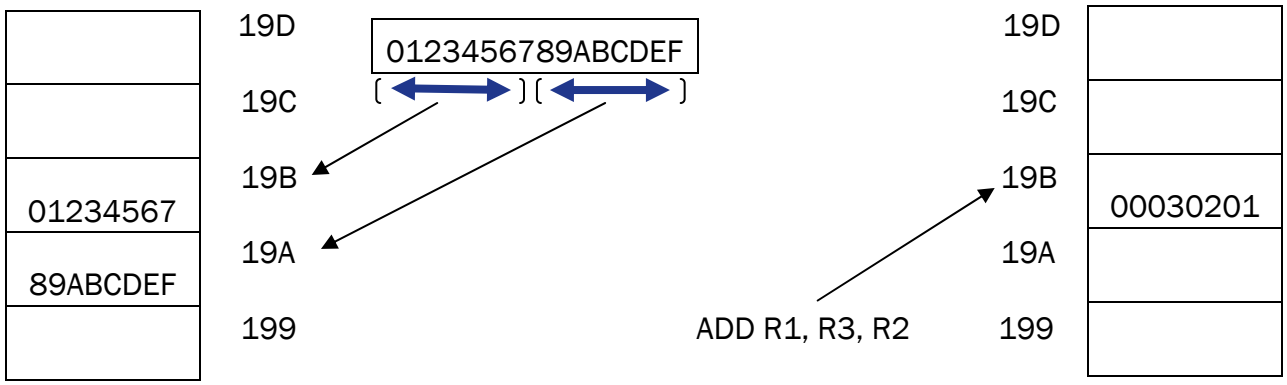
Table of Contents

Abstract

The Instruction Set Architecture (ISA) is the part of a computing machine visible for programming, including its native data types, instructions, registers, memory architecture, exception handling and external interfaces. This paper presents enhancements to a basic ISA, which will allow for rapid manipulation and alteration of memory in order to accelerate the building of images in a frame buffer intended for output to a display. The ISA is implementing vector, matrix, and (enhancements) instructions in order to accelerate geometric calculations such as the rotation and translation of vertices.

Memory Structure

The memory contains 64-bit address paces with a minimum of 1024 words for the simulation. Each memory location holds one 32-bit word. Thus, 64-bit memory operands will be stored in two consecutive memory locations in “little endian” format with the least significant 32-bits being stored at the lower address and the most significant 32-bits being stored at the higher address.



Processor Register Set

Processor register set contains all the registers that are available to the user. There 32 integer registers and 32 floating-point registers. NAME registers are mainly 64-bits.

Data Register

Data registers are used to store intermediate data values are results when any arithmetic operation is performed. The registers are allocated to the variables according to the type of operation. They can be used with any instruction that performs operations on data.

Floating-point Register

Data contained in the floating-point registers can be either integer or real type.

Vector Register

Data contained in the vector registers can be either integer or real type.

Address Register

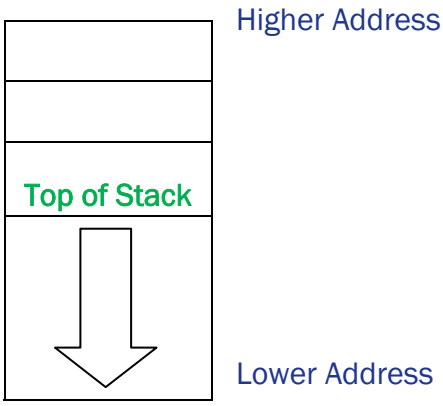
Address register are used to store the addresses of other registers and different memory locations. The either contain main memory addresses of data and instructions or they contain a portion of the address that is used in the calculation of the complete addresses.

Instruction Pointer

The instruction pointer, commonly called the program counter, is a 64-bit register that indicates where the computer is in the program sequence. The instruction pointer is incremented after fetching an instruction. It stores the memory address of the next instruction to be executed. Typically, the instructions are usually fetched from memory sequentially. There are some cases where the instruction pointer may not be sequential. Control transfer instructions such as jumps, subroutine calls, and returns places a new value inside the instruction pointer.

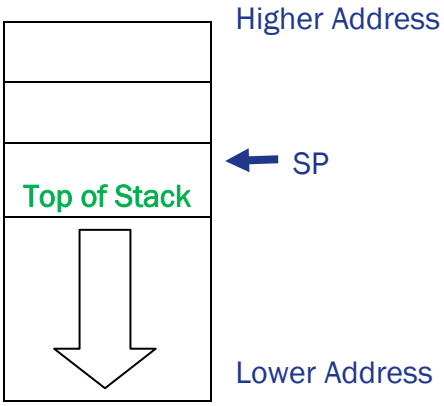
Stack

A Stack is an area in memory where data is added or removed in a last-in-first-out approach.



Stack Pointer

The Stack Pointer is a 64-bit register. The stack pointer's function is to point to the address of the top of the stack.



- When adding data onto the stack – push (Pre-decrement)
  SP = SP – 4
  Store data on stack at new SP
- When removing data from the stack – pop (Post-increment)
  Load data from stack at SP
  SP = SP + 4

Flags Register

The flag register is a 64-bit hardware register that indicates which state the processor is in. Specific bits inside the flags register indicates what condition the register is in. After an instruction is execute, the bits will trigger on or off.

Zero Flag

The zero flag indicates that the result of the arithmetic and logic operation was equal to zero.

Carry Flag

The carry flag indicates when arithmetic carries or borrows has been generated out of the most significant ALU bit position.

Negative/Signed Flag

The negative or signed flag indicates that the result of the arithmetic operation was negative.

Overflow Flag

The overflow flag indicates when an arithmetic overflow has occurred in an operation, which indicates the signed two's-complement result would not fit in the number of bits used for the operation.

Interrupt Flag

The interrupt flag may be set or cleared by certain instructions. If the flag is set to 1, maskable hardware interrupts will be handled otherwise they will be ignored when set to 0 or cleared.

Parity Flag

The parity flag indicates if the number of set bits is odd or even in the binary representation of the result of the last operation.

ADDRESSING MODE

Register

31-24	23-21	20-16	15-13	12-08	07-05	04-00
Opcode	Enhance	Source1 Reg	Enhance	Source2 Reg	Enhance	Dest Reg

Basic format for triple/double/single operations:

- All triple-op instructions are register based, which means no memory accesses are allowed. Thus, the "source1 reg" and "source2 reg" field are operated on and the result is returned to the register specified by the "dest reg" field.
- Double-op instructions have the same format as triple-op, but are encoded according to the following cases:
  - For load immediate, the 64-bit immediate value is transferred into "dest reg".
  - For load/input, the 64-bit word in memory or I/O location specified by the "source1 reg" is transferred into the register specified by the "dest reg" field.
  - For store/output, the 64-bit register specified by the "source1 reg" field is transferred to a memory or I/O locations specified by the "dest reg" field.
  - For exchange/compare/test, the two operands are registers specified by the "source1 reg" and "source2 reg" fields.
  - For copy, the "source1 reg" is copied into the "dest reg."
- Single-op instructions have the same format as above, but encoded according to these cases:
  - For Push, the register to store to memory, which is pointed at by stack pointer (SP), is specified by the "source1 reg" field. Thus, the register pushed will be store in two consecutive memory locations, using "little endian" format, starting with the address held in the SP.
  - For Pop, the register to be loaded from memory pointed at by SP is specified by the "dest reg" field. Thus, the 64-bit memory operand to be loaded is stored in two consecutive memory locations, using "little endian" format, starting with the address held in the SP.
  - For all other operations, the single source operand is a register specified by the "source1 reg" field. The result of the operation is to go back into the register specified by the "dest reg" field.

Register Indirect

31-24	23-21	20-16	15-00
Opcode	Enhancement	Source Reg	Enhancement

The "effective address" is found in the register specified by the "source reg" field. The contents of that register are transferred into the Instruction Pointer.

Immediate

31-24	23-21	20-16	15-08	07-05	04-00
Opcode	Enhancement	Source Reg	8-bit Imm.	Enhancement	Dest Reg

All immediate operand instructions are register based. No memory accesses are allowed. Thus, the source register, specified by the "source reg" and "8-bit immediate value" fields are operated on and the result is returned to the register specified by the "dest reg" field. The 8-bit immediate value is a two's complement number that must be sign-extended to become a 64-bit signed integer for the respective operation.

Jump

31-24	23-00
Opcode	24-bit Signed Displacement

The "effective address" of these "jump" instructions is calculated by adding the 64-bit "sign-extension" of a 24-bit signed displacement to the Instruction Pointer.

Flags/Processor

31-24	23-00
Opcode	Zero Fill

Bits 23 to 0 are filled with zeros, and the Opcode in bits 31 to 24 will specify which of the six flaps/processor instructions to execute.

Floating Point

31-24	23-21	20-16	15-13	12-08	07-05	04-00
Opcode	Enhance	Source1 Reg	Enhance	Source2 Reg	Enhance	Dest Reg

The registers used depend on the instructions being executed.

- The **addition**, **subtraction**, **multiply**, and **divide** instructions are all register based, which means no memory accesses are allowed. Thus, all the register fields will be operated on and the result is returned to the floating point register specified by the destination register field.
- The **increment** and **decrement** instructions are both register based. The single operand is a floating point register specified by the "source1 reg" field with the results going back into the same register, thus the "dest reg" field will have the same register code as the "source1 reg" field.
- The **zero** and **one** instructions result in either a "0.0" or "1.0" value being stored into the 'dest reg" field.
- For **load immediate**, the 64-bit immediate value is transferred into the floating pint register specified by the "dest reg" field.
- For load, the 64-bit value in two consecutive memory locations pointed at by the integer register specified by the "source1 reg" field is transferred into the floating point register specified by the "dest reg" field.
- For **store**, the 64-bit floating point register specified by the "source1 reg" field is transferred to the two memory locations pointed at by the integer register specified by the "dest reg" field.

Instruction Set Classification

Program Status Register

C: Carry Flag
Z: Zero Flag
N: Negative Flag
P: Parity Flag
O: Overflow Flag

Registers

src1_reg: Source 1
src2_reg: Source 2
dest_reg: Destination
K: Constant Value
PC: Program Counter
SP: Stack Pointer

Flags

⇔: Flags affected by instruction
0: Flag cleared by instruction
1: Flag set by instruction
-: Flag not affected by instruction

ADD

Description:
Performs an arithmetic addition between the contents of two registers and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg + src2_reg

Instruction Format:
ADD dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0000	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Instruction	Contents Hex	Binary
ADD R0, R2, R5	R2 = 0x02F11058	0000_0010_1111_0001_0001_0000_0101_1000
	R5 = 0x1A353201	0001_1010_0011_0101_0011_0010_0000_0001

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0000	000	00010	000	00101	000	00000

```
          1 1 111      1    11                    ←   Carries
   0000_0010_1111_0001_0001_0000_0101_1000
+  0001_1010_0011_0101_0011_0010_0000_0001
   0001_1101_0010_0110_0100_0010_0101_1001   →   0x1D264259
```

R0 ← 0x1D264259

C	N	O	P	Z
X	X	X	0	X

EXAMPLE 2

Instruction	Contents Hex	Binary
ADD R1, R3, R6	R3 = 0xC2D5945A	1100_0010_1101_0101_1001_0100_0101_1010
	R6 = 0xD2353205	1101_0010_0011_0101_0011_0010_0000_0101

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0000	000	00011	000	00110	000	00001

```
  11       1 1  111   1 1    11                    ←   Carries
   1100_0010_1101_0101_1001_0100_0101_1010
+  1101_0010_0011_0101_0011_0010_0000_0101
   1001_0101_0000_1010_1100_0110_0101_1111   →   0x950AC65F
```

R1 ← 0x950AC65F

C	N	O	P	Z
1	1	X	1	X

SUBTRACT

Description:
Performs an arithmetic subtraction between the contents of two registers and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg – src2_reg

Instruction Format:
SUB dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Instruction	Contents Hex	Contents Binary
SUB R1, R3, R6	R3 = 0x53F35676	0101_0011_1111_0011_0101_0110_0111_0110
	R6 = 0x161D2037	0001_0110_0001_1101_0010_0000_0011_0111

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0001	000	00011	000	00110	000	00001

```
      1 10  1          10                    10 10 1 10 10
    0 ~~10~~ 0  ~~10~~10      0 ~~0~~  10 10      0 10          0 ~~0~~ ~~0~~  ~~10~~ ~~0~~ ~~0~~ 10      ←   Borrows
  0~~101~~_0~~0~~11_1~~111~~_~~00~~11_0~~10~~1_0110_0~~111~~_~~011~~0
- 0001_0110_0001_1101_0010_0000_0011_0111
  0011_1101_1101_0110_0011_0110_0011_1111                  →   0x3DD6363F
```

R1 ← 0x3DD6363F

C	N	O	P	Z
X	X	X	0	X

EXAMPLE 2

Instruction	Contents Hex	Contents Binary
SUB R5, R6, R7	R6 = 0x081E146A	0000_1000_0001_1110_0001_0100_0110_1010
	R7 = 0x081E146A	0000_1000_0001_1110_0001_0100_0110_1010

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0001	000	00110	000	00111	000	00101

```
                                                           ←   Borrows
  0000_1000_0001_1110_0001_0100_0110_1010
- 0000_1000_0001_1110_0001_0100_0110_1010
  0000_0000_0000_0000_0000_0000_0000_0000                  →   0x00000000
```

R5 ← 0x00000000

C	N	O	P	Z
X	X	X	0	1

MULTIPLY

Description:
Performs an arithmetic multiplication between the contents of two registers and places the lower 64-bit result into the dest_reg.

Operation:
Product[127:64]:dest_reg ← src1_reg * src2_reg

Instruction Format:
MUL dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

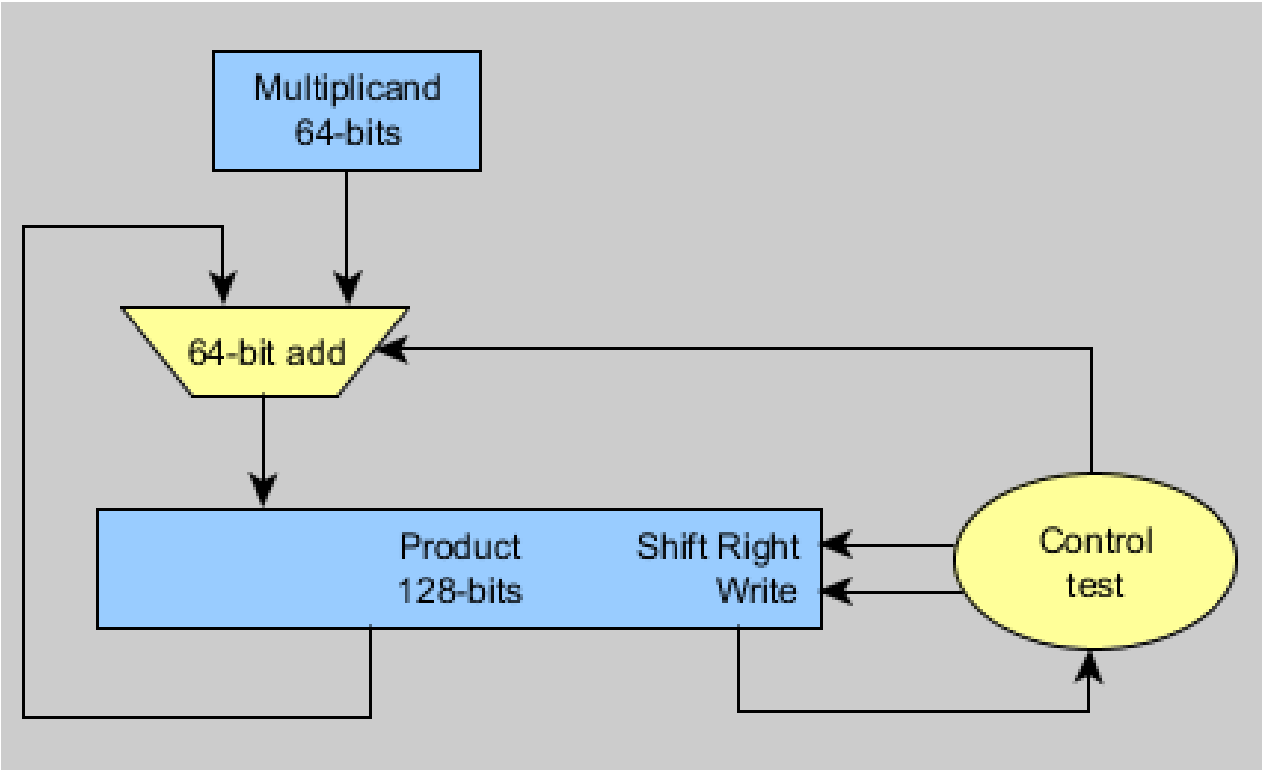
C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

ILLUSTRATION



DIVIDE

Description:
Performs an arithmetic division between the contents of two registers and places the lower 64-bit result into the dest_reg.

Operation:
Quotient[127:64]:dest_reg ← src1_reg / src1_reg

Instruction Format:
DIV dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0011	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

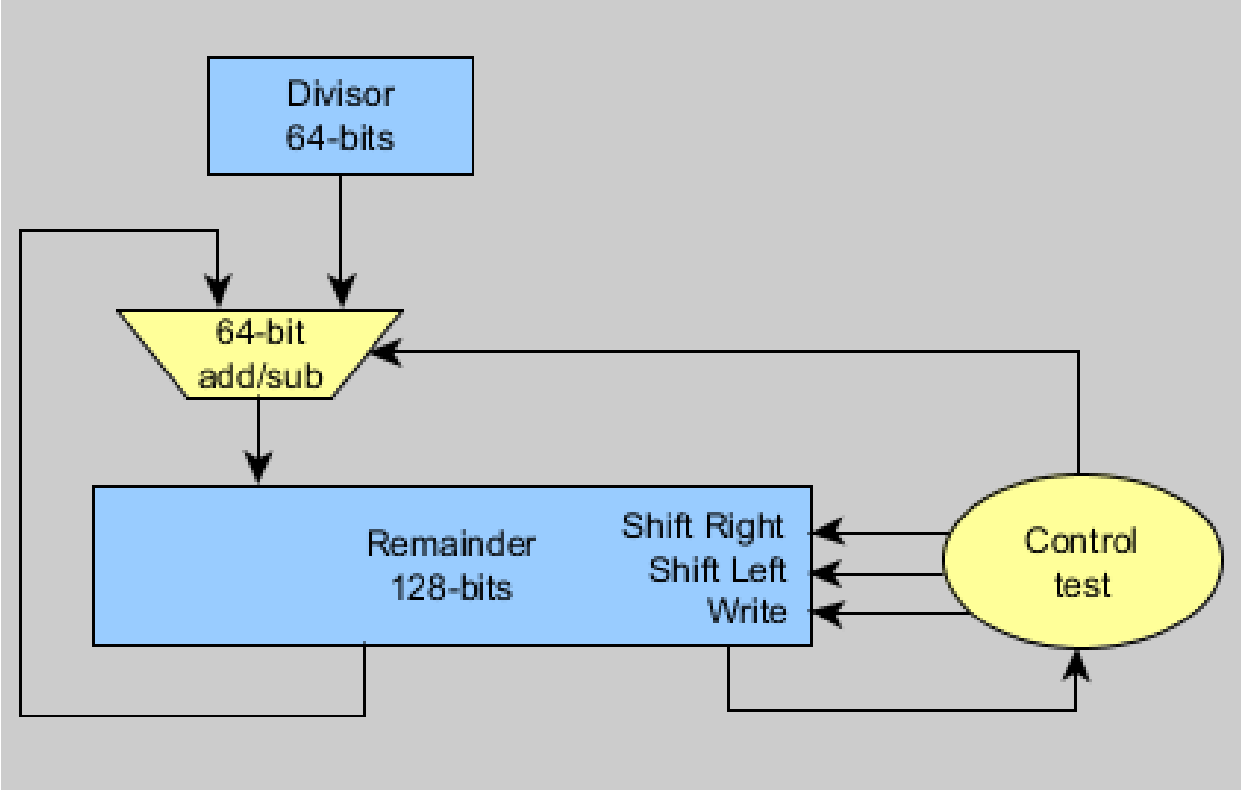
C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

ILLUSTRATION



AND

Description:
Performs a logical AND between the contents of two registers and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg & src2_reg

Instruction Format:
AND dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0100	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	⇔	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Instruction	Contents Hex	Binary
AND R0, R2, R7	R2 = 0x654A3C51	0110_0101_0100_1010_0011_1100_0101_0001
	R7 = 0xF0EE3269	1111_0000_1110_1110_0011_0010_0110_1001

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0100	000	00010	000	00111	000	00000

```
  0110_0101_0100_1010_0011_1100_0101_0001
& 1111_0000_1110_1110_0011_0010_0110_1001
  0110_0000_0100_1010_0011_0000_0100_0001  →  0x604A3041
```

R0 ← 0x604A3041

C	N	O	P	Z
X	X	X	0	0

EXAMPLE 2

Instruction	Contents Hex	Binary
AND R4, R2, R1	R2 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
	R1 = 0x5A5A5A5A	0101_1010_0101_1010_0101_1010_0101_1010

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0100	000	00010	000	00001	000	00100

```
  1010_0101_1010_0101_1010_0101_1010_0101
& 0101_1010_0101_1010_0101_1010_0101_1010
  0000_0000_0000_0000_0000_0000_0000_0000  →  0x00000000
```

R4 ← 0x00000000

C	N	O	P	Z
X	X	X	0	1

OR

Description:
Performs a logical OR between the contents of two registers and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg | src2_reg

Instruction Format:
OR dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0101	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	⇔	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Instruction	Contents Hex	Binary
OR R0, R2, R7	R2 = 0x654A3C51	0110_0101_0100_1010_0011_1100_0101_0001
	R7 = 0xF0EE3269	1111_0000_1110_1110_0011_0010_0110_1001

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0101	000	00010	000	00101	000	00000

```
  0110_0101_0100_1010_0011_1100_0101_0001
| 1111_0000_1110_1110_0011_0010_0110_1001
  1111_0101_1110_1110_0011_1110_0111_1001  →  0xF5EE3E79
```

R0 ← 0x F5EE3E79

C	N	O	P	Z
X	1	X	0	0

EXAMPLE 2

Instruction	Contents Hex	Binary
OR R4, R2, R1	R2 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
	R1 = 0x5A5A5A5A	0101_1010_0101_1010_0101_1010_0101_1010

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0101	000	00010	000	00001	000	00100

```
  1010_0101_1010_0101_1010_0101_1010_0101
| 0101_1010_0101_1010_0101_1010_0101_1010
  1111_1111_1111_1111_1111_1111_1111_1111  →  0xFFFFFFFF
```

R4 ← 0xFFFFFFFF

C	N	O	P	Z
X	1	X	0	0

XOR

Description:
Performs the logical XOR between the contents of two registers and places the result into dest_reg.

Operation:
dest_reg ← src1_reg ^ src2_reg

Instruction Format:
XOR dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0110	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	⇔	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Instruction	Contents Hex	Contents Binary
XOR R3, R5, R7	R5 = 0x5A36752C	0101_1010_0011_0110_0111_0101_0010_1100
	R7 = 0x65298B63	0110_0101_0010_1001_1000_1011_0110_0011

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0110	000	00101	000	00111	000	00011

```
  0101_1010_0011_0110_0111_0101_0010_1100
⊕ 0110_0101_0010_1001_1000_1011_0110_0011
  0011_1111_0001_1111_1111_1110_0100_1111  →  0x3F1FFE4F
```

R3 ← 0x3F1FFE4F

C	N	O	P	Z
X	X	X	0	X

EXAMPLE 2

Instruction	Contents Hex	Contents Binary
XOR R5, R9, R8	R9 = 0xA5A5A5A5	1111_0101_1010_0101_1010_0101_1010_1111
	R8 = 0x5A5A5A5A	0101_1010_0101_1010_0101_1010_0101_1010

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0110	000	01001	000	01000	000	00101

```
  1111_0101_1010_0101_1010_0101_1010_1111
⊕ 0101_1010_0101_1010_0101_1010_0101_1010
  1110_1111_1111_1111_1111_1111_1111_0111  →  0xEFFFFFF7
```

R5 ← 0xEFFFFFF7

C	N	O	P	Z
X	1	X	0	X

LOAD IMMEDIATE

Description:
Places the content of an immediate value into the dest_reg.

Operation:
dest_reg ← K

Instruction Format:
LDI dest_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0111	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1 Contents

Instruction
LDI R5, 0x00000089

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0111	000	src1_reg	0x00000089	000	dest_reg

R5 ← 0x00000089 R5 = 0000_0000_0000_0000_0000_0000_1000_1001

EXAMPLE 2 Contents

Instruction
LDI R0, 0x000000FF

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0111	000	src1_reg	8 – bit immediate value	000	dest_reg

R0 ← 0xFFFF0000 R5 = 1111_1111_1111_1111_0000_0000_0000_0000

LOAD

Description:
Copies the content from a memory and places it into the dest_reg.

Operation:
dest_reg ← [src1_reg]

Instruction Format:
LD dest_reg, src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1000	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

	Hex	Binary
	R5 = 0x40000004	0100_0000_0000_0000_0000_0000_0000_0100
Instruction	Memory Location	Contents
LD R4, [R5]	0x40000000	0x00005301
	0x40000004	0x500149A9
	0x40000008	0x06B43700

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1000	000	00101	000	00000	000	00100

R4 ← 0x500149A9

EXAMPLE 2

	Hex	Binary
	R6= 0x40000008	0100_0000_0000_0000_0000_0000_0000_1000
Instruction	Memory Location	Contents
LD R3, [R6]	0x40000000	0x00005301
	0x40000004	0x500149A9
	0x40000008	0x06B43700

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1000	000	00110	000	0000	000	00011

R3 ← 0x06B43700

STORE

Description:
Copies content of source register Rs and stores value into memory.

Operation:
[src2_reg] ← src1_reg

Instruction Format:
ST [src2_reg], src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

	Hex	Binary
	R2 = 0xAB394461	1010_1011_0011_1001_0100_0100_0110_0001
Instruction	R4 = 0x40000004	0100_0000_0000_0000_0000_0000_0000_0100
ST [R4], R2	**Memory Location**	**Contents**
	0x40000000	0x00005301
	0x40000004	~~0x500149A9~~ 0xAB394461
	0x40000008	0x06B43700

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1001	000	00010	000	00000	000	00100

0x40000004 ← 0xAB394461

EXAMPLE 2

	Hex	Binary
	R1 = 0x00000000	0000_0000_0000_0000_0000_0000_0000_0000
Instruction	R6 = 0x40000008	0100_0000_0000_0000_0000_0000_0000_1000
ST [R6], R1	**Memory Location**	**Contents**
	0x40000000	0x11004401
	0x40000004	0xE00100A9
	0x40000008	~~0x95637411~~ 0x00000000

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1001	000	00001	000	00000	000	00110

0x40000008 ← 0x00000000

COPY

Description:
Copies the content in register src1_reg and stores the value in register dest_reg.

Operation:
dest_reg ← src1_dest

Instruction Format:
CPY dest_reg, src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Instruction	Contents Hex	Binary
CPY R8, R9	R8 = 0x5995331A	0101_1001_1001_0101_0011_0011_0001_1010
	R9 = 0x6510E364	0110_0101_0001_0000_1110_0011_0110_0100

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1010	000	01000	000	01001	000	00000

R8 ← 0x6510E364

EXAMPLE 2

Instruction	Hex	Binary
	R3 = 0xFFF63441	1111_1111_1111_0110_0011_0100_0100_0001
CPY R3, [R6]	R6 = 0x40000000	0100_0000_0000_0000_0000_0000_0000_000

Memory Location	Contents
0x40000000	0x00005301
0x40000004	0x500149A9
0x40000008	0x06B43700

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0000	000	00011	000	00110	000	00000

R3 ← 0x00005301

EXCHANGE

Description:
Performs a swap between the contents of two registers.

Operation:
src1_reg ← dest_reg, dest_reg ← src1_reg

Instruction Format:
XCH src1_reg, dest_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1011	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Instruction	Contents Hex	Contents Binary
XCH R3, R6	R3 = 0x00001541	0000_0000_0000_0000_0001_0101_0100_0001
	R6 = 0x2F631100	0110_0101_0010_1001_1000_1011_0110_0011

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1011	000	00011	000	00110	000	00000

TEMP ← R3
R3 ← R6
R6 ← R3

TEMP ← 0x00001541
R3 ← 0x2F631100
R6 ← 0x00001541

EXAMPLE 2

Instruction	Contents Hex	Contents Binary
XCH R2, R7	R2 = 0x00001541	0000_0000_0000_0000_0001_0101_0100_0001
	R7 = 0x00000000	0000_0000_0000_0000_0000_0000_0000_0000

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1011	000	00010	000	00111	000	00000

TEMP ← R2
R2 ← R7
R7 ← TEMP

TEMP ← 0x00001541
R2 ← 0x00000000
R7 = 0x00001541

INPUT

Description:
Loads data from the I/O Space (Ports, Timers, Configuration Registers etc.) into register dest_reg in the Register File.

Operation:
dest_reg ← I/O (A)

Instruction Format:
IN dest_reg, A

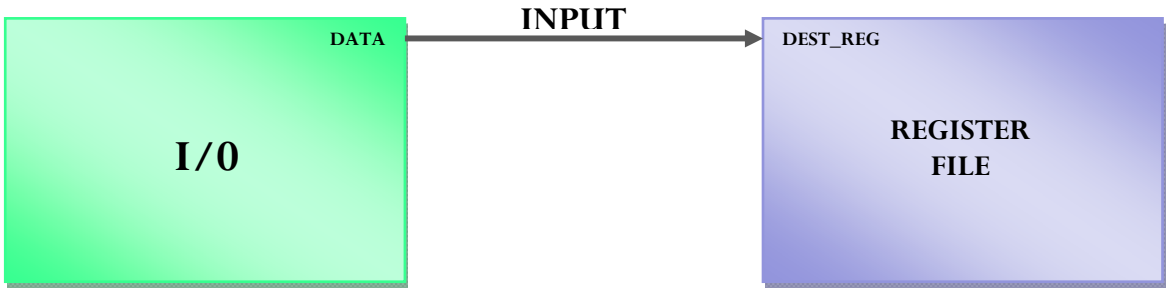
32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1100	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE



OUTPUT

Description:
Stores data from register src1_reg in the Register File into I/O Space (Ports, Timers, Configuration Registers, etc.).

Operation:
I/O (A) ← dest_reg

Instruction Format:
OUT A, dest_reg

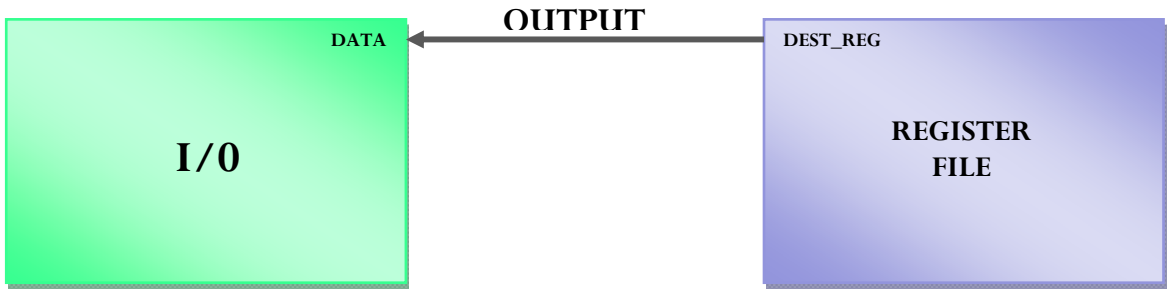
32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1101	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE



COMPARE

Description:
This instruction performs a compare between two registers src2_reg and src1_reg. Subtracts src1_reg from src2_reg and compares the result to zero, but does not store the result. This instruction updates the Program Status Register to use for jump statements.

Operation:
src1_reg – src2_reg == 0

Instruction Format:
CP src2_reg, src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1110	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

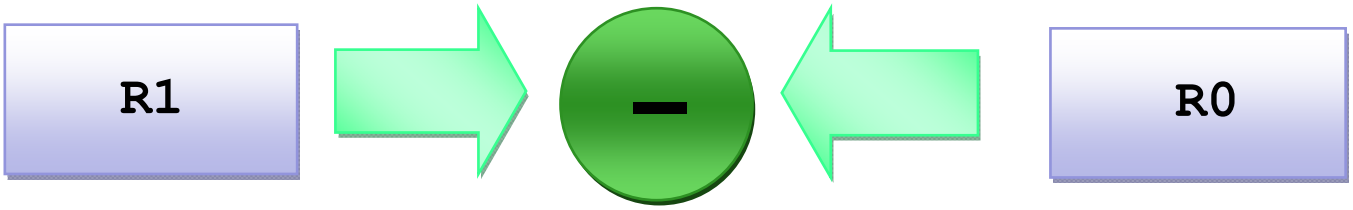
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
CP R0, R1

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1110	000	00000	000	00001	000	00000



TEST

Description:
Tests if a register is zero or negative. Performs a logical AND between a register and itself. The register will remain unchanged. This instruction updates the Program Status Register to use for jump statements.

Operation:
src1_reg & src1_reg

Instruction Format:
TST src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_1111	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	0	⇔	⇔

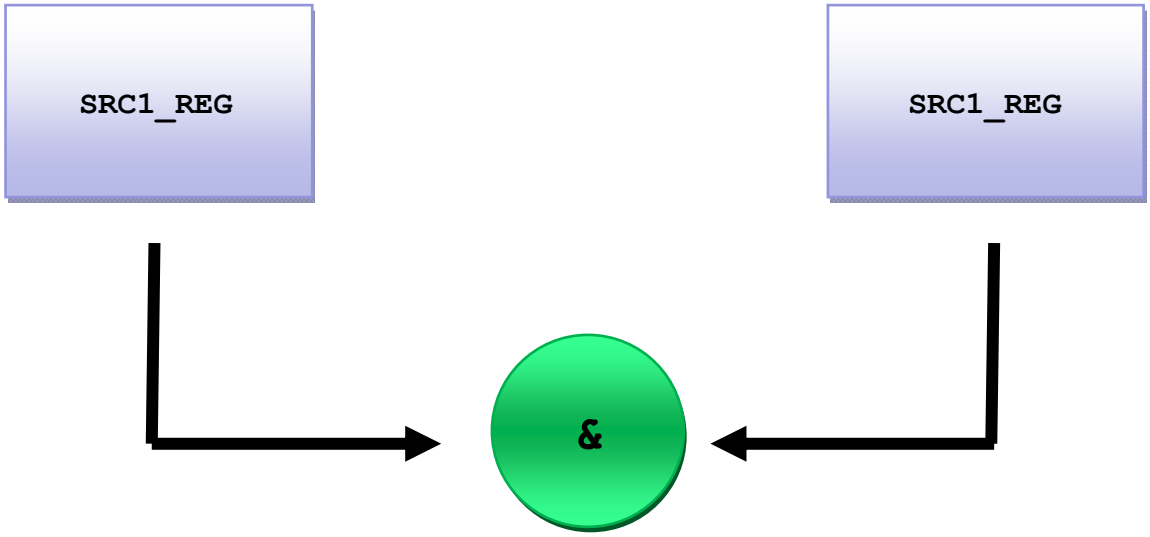
N: Set if MSB of the result is set; cleared otherwise.

O: 0
Cleared

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE



C	N	O	P	Z
-	⇔	0	⇔	⇔

UPDATED STATUS REGISTERS

PUSH

Description:
This instruction stores the content of the src1_reg on the STACK. The STACK POINTER is then pre-decremented by 1 after the push.

Operation:
STACK ← src1_dest

Instruction Format:
PUSH src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0000	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

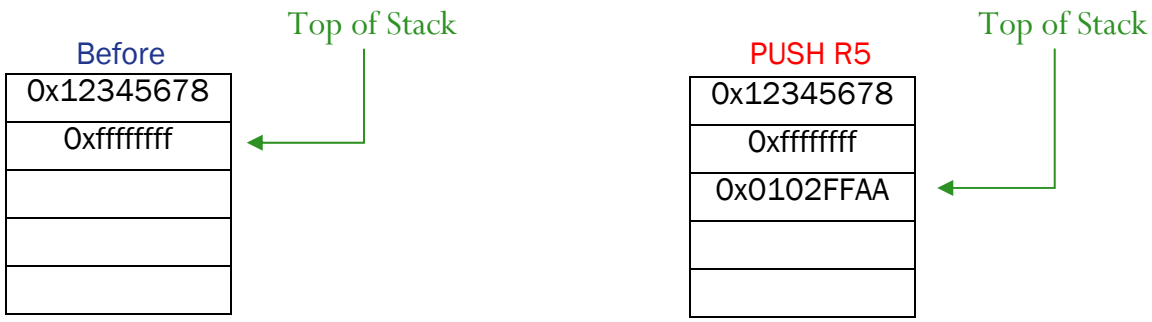
Instruction
PUSH R5

Contents

Hex	Binary
R5 = 0x0102FFAA	0000_0001_0000_0001_1111_1111_1010_1010

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0000	000	00101	000	00000	000	00000



EXAMPLE 2

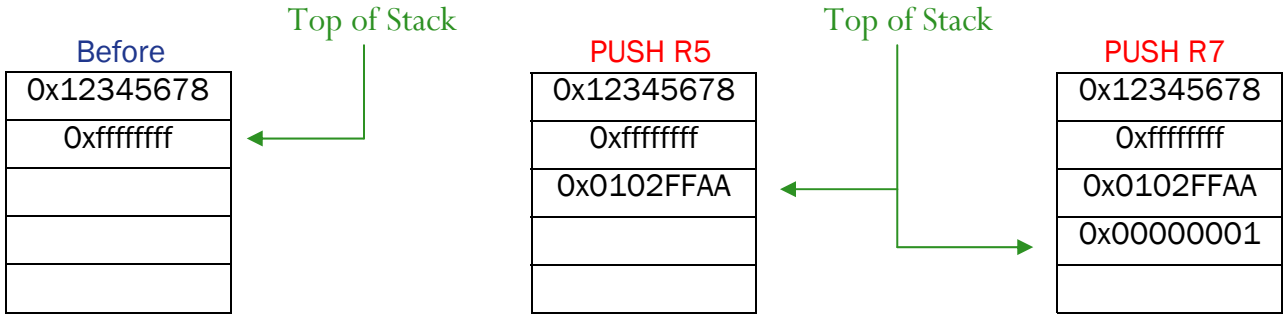
Instruction
PUSH R5
PUSH R7

Contents

Hex	Binary
R5 = 0x0102FFAA	0000_0001_0000_0001_1111_1111_1010_1010
R7 = 0x00000001	0000_0000_0000_0000_0000_0000_0000_0001

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0000	000	00111	000	00000	000	00000



POP

Description:
This instruction loads register dest_reg with the value from the top of the stack. The Stack Pointer is pre-decremented by 1 before the pop.

Operation:
src1_reg ← STACK

Instruction Format:
POP src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

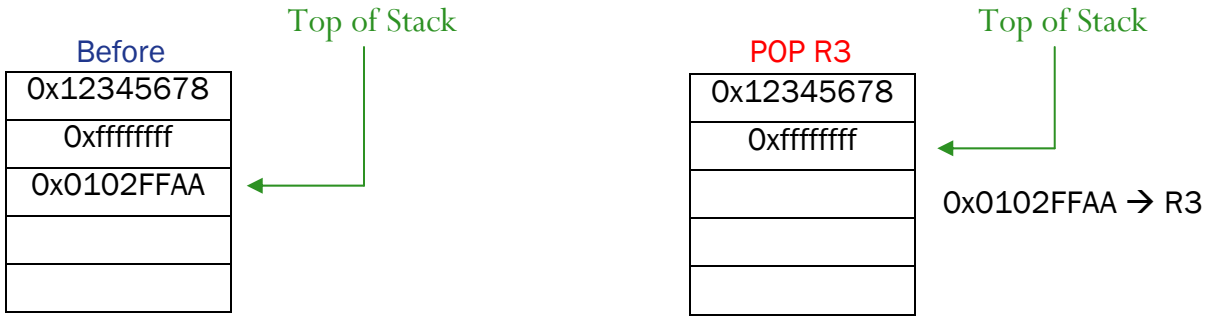
Instruction
POP R3

Contents

Hex	Binary
R3 = 0x7FFFFFFF	0111_1111_1111_1111_1111_1111_1111_1111

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0001	000	00011	000	00000	000	00000



EXAMPLE 2

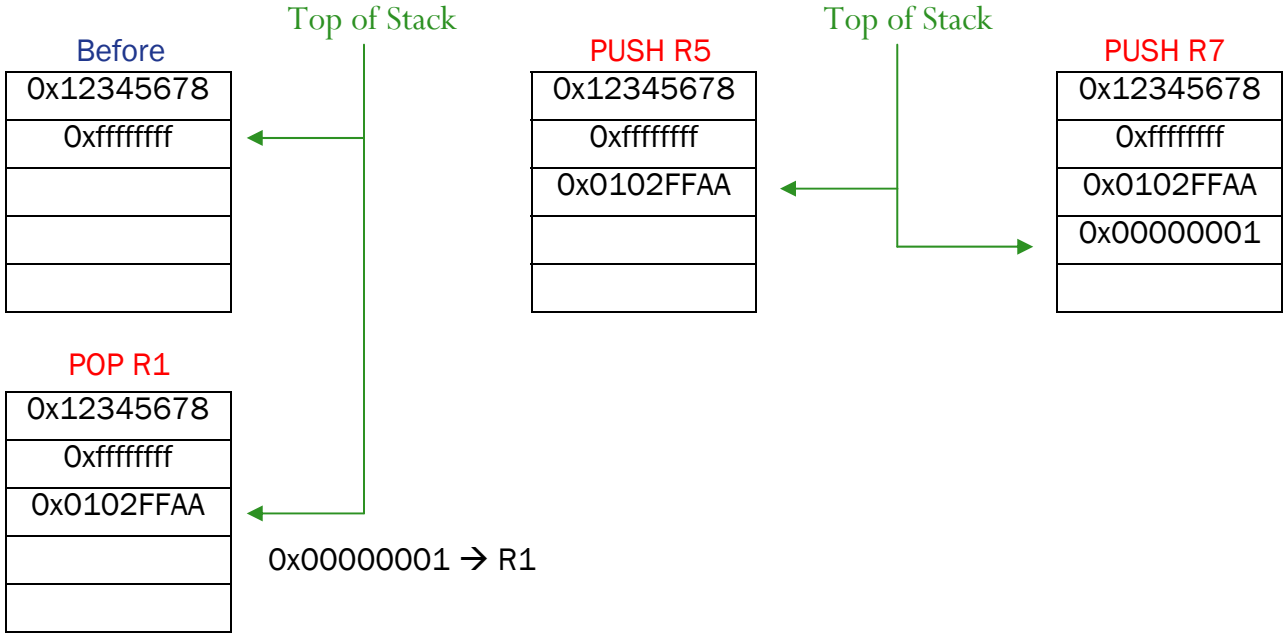
Instruction
PUSH R5
PUSH R7
POP R1

Contents

Hex	Binary
R5 = 0x0102FFAA	0000_0001_0000_0001_1111_1111_1010_1010
R7 = 0x00000001	0000_0000_0000_0000_0000_0000_0000_0001

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0001	000	00001	000	00000	000	00000



NEGATE

Description:
Performs a 2's compliment (NOT each bit, then add 1) instruction on the content of the src1_reg and places the result in the dest_reg.

Operation:
dest_reg ← ~src1_reg + 1

Instruction Format:
NEG src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0010_0010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
0	⇔	-	⇔	⇔

C: Cleared.

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

	Hex	Binary
Instruction	R2 = 0xA643E174	1010_0110_0100_0011_1110_0001_0111_0100
NEG R2		

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0010_0001	000	00010	000	00000	000	00010

~1010_0110_0100_0011_1110_0001_0111_0100 ← R2

0101_1001_1011_1100_0001_1110_1000_1011 ← Complimented
+ 0000_0000_0000_0000_0000_0000_0000_0001 ← Adding 1
0101_1001_1011_1100_0001_1110_1000_1100 → R2

R2 ← 0x59BC1E8C

C	N	O	P	Z
X	X	X	X	1

EXAMPLE 2

Contents

	Hex	Binary
Instruction	R3 = 0x5A5A5A5A	0101_1010_0101_1010_0101_1010_0101_1010
NEG R3		

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0010_0010	000	00011	000	00000	000	00011

~0101_1010_0101_1010_0101_1010_0101_1010 ← R3

1010_0101_1010_0101_1010_0101_1010_0101 ← Complimented
+ 0000_0000_0000_0000_0000_0000_0000_0001 ← Adding 1
1010_0101_1010_0101_1010_0101_1010_0110 → R3

R2 ← 0xA5A5A5A6

C	N	O	P	Z
X	X	X	X	1

NOT

Description:
Performs the logical negation of the contents in the src1_reg and places the result in the dest_reg.

Operation:
dest_reg ← ~src1_reg

Instruction Format:
NOT src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0011	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	⇔	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

	Hex	Binary
Instruction	R5 = 0x0000FFFF	0000_0000_0000_0000_1111_1111_1111_1111
NOT R5		

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0011	000	00101	000	00000	000	00101

~0000_0000_0000_0000_1111_1111_1111_1111 ← R5

1111_1111_1111_1111_0000_0000_0000_0000 ← Complimented
1111_1111_1111_1111_0000_0000_0000_0000 → R2

R5 ← 0xFFFF0000

C	N	O	P	Z
X	X	X	1	1

EXAMPLE 2

Contents

	Hex	Binary
Instruction	R0 = 0x5A5A5A5A	0101_1010_0101_1010_0101_1010_0101_1010
NOT R0		

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0011	000	00000	000	00000	000	00000

~0101_1010_0101_1010_0101_1010_0101_1010 ← R0

1010_0101_1010_0101_1010_0101_1010_0101 ← Complimented
1010_0101_1010_0101_1010_0101_1010_0101 → R0

R0 ← 0xA5A5A5A5

C	N	O	P	Z
X	X	X	1	1

INCREMENT

Description:
Adds one -1- from the contents of the src1_reg and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg + 1

Instruction Format:
INC src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0100	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	⇔	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

	Hex	Binary
Instruction	R1 = 0xA25509F1	1010_0010_0101_0101_0000_1001_1111_0001
INC R1		

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0100	000	00001	000	00000	000	00001

```
                                          1   ←   Carry
    1010_0010_0101_0101_0000_1001_1111_0001
+   0000_0000_0000_0000_0000_0000_0000_0001
    1010_0010_0101_0101_0000_1001_1111_0010   →   0xA25509F2
```

R1 ← A25509F2

C	N	O	P	Z
X	1	X	0	X

EXAMPLE 2

Contents

	Hex	Binary
Instruction	R2 = 0xFFFFFFFF	1111_1111_1111_1111_1111_1111_1111_1111
INC R2		

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0100	000	00010	000	00000	000	00010

```
  1 1111   1111   1111   1111   1111   1111   1111   111   ←   Carry
    1111_1111_1111_1111_1111_1111_1111_1111
+   0000_0000_0000_0000_0000_0000_0000_0001
  1 _0000_0000_0000_0000_0000_0000_0000_0000   →   0x00000000
```

R2 ← 00000000

C	N	O	P	Z
1	X	1	0	X

DECREMENT

Description:
Subtracts one -1- from the contents of the src1_reg and places the result into the dest_reg.

Operation:
src1_reg ← src1_reg – 1

Instruction Format:
DEC src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0101	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	⇔	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Instruction	Contents Hex	Contents Binary
DEC R4	R4 = 0xA25509F1	1010_0010_0101_0101_0000_1001_1111_0001

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0101	000	00100	000	00000	000	00100

```
                                        ←   Borrow
  1010_0010_0101_0101_0000_1001_1111_0001
- 0000_0000_0000_0000_0000_0000_0000_0001
  1010_0010_0101_0101_0000_1001_1111_0000   →   0xA25509F0
```

R4 ← A25509F0

C	N	O	P	Z
X	1	X	0	X

EXAMPLE 2

Instruction	Contents Hex	Contents Binary
DEC R3	R3 = 0xFFFFFFFF	1111_1111_1111_1111_1111_1111_1111_1111

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_0101	000	00011	000	00000	000	00011

```
                                        ←   Borrow
  1111_1111_1111_1111_1111_1111_1111_1111
- 0000_0000_0000_0000_0000_0000_0000_0001
  1111_1111_1111_1111_1111_1111_1111_1110   →   0xFFFFFFFE
```

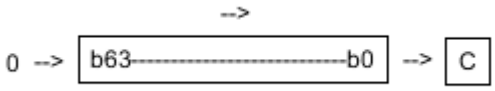
R3 ← FFFFFFFE

C	N	O	P	Z
X	1	X	0	X

LOGICAL SHIFT RIGHT

Description:
Shifts all bits in the src1_reg one place to the right. Bit 63 is cleared. Bit 0 is loaded into the C Flag of the Status Register. This operation divides an unsigned value by two

Operation:



Instruction Format:
SRL src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1000	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	0	⇔	⇔	⇔

C: Set if, before the shift, the LSB of Src1_reg was set; cleared otherwise.

N: 0

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

	Hex	Binary
Instruction	R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
SRL R5		

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1000	000	00101	000	00000	000	00000

1010_0101_1010_0101_1010_0101_1010_0101 ← R5

→ → → → → → → →
0101_1010_0101_1010_0101_1010_0101_1010 1 → 0x5A5A5A5A

R5 ← 0x5A5A5A5A

C	N	O	P	Z
1	0	X	1	X

EXAMPLE 2

Contents

	Hex	Binary
Instruction	R8 = 0xFFFFFFFF	1111_1111_1111_1111_1111_1111_1111_1111
SRL R8		

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1000	000	01000	000	00000	000	00000

1111_1111_1111_1111_1111_1111_1111_1111 ← R8

→ → → → → → → →
0111_1111_1111_1111_1111_1111_1111_1111 1 → 0x7FFFFFFF

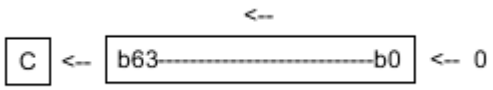
R8 ← 0x7FFFFFFF

C	N	O	P	Z
1	0	X	0	X

LOGICAL SHIFT LEFT

Description:
Shifts all bits in the src1_reg one place to the left. Bit 63 is cleared. Bit 0 is loaded into the C Flag of the Status Register. This operation multiplies an unsigned value by two.

Operation:



Instruction Format:
SLL src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if, before the shift, the MSB of Src1_reg was set; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Instruction
SLL R5

Contents

Hex	Binary
R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1001	000	00101	000	00000	000	00000

1010_0101_1010_0101_1010_0101_1010_0101 ← R5

← ← ← ← ← ← ← ←

1 0101_1010_0101_1010_0101_1010_0101_1010 → 0x5A5A5A5A

R5 ← 0x5A5A5A5A

C	N	O	P	Z
1	0	X	1	X

EXAMPLE 2

Instruction
SLL R8

Contents

Hex	Binary
R8 = 0xFFFFFFFF	1111_1111_1111_1111_1111_1111_1111_1111

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1001	000	01000	000	00000	000	00000

1111_1111_1111_1111_1111_1111_1111_1111 ← R8

← ← ← ← ← ← ← ←

1 1111_1111_1111_1111_1111_1111_1111_1110 → 0xFFFFFFFE

R8 ← 0xFFFFFFFE

C	N	O	P	Z
1	1	X	0	X

ARITHMETIC SHIFT RIGHT

Description:
Shifts all bits of register src1_reg one place to the right. Copies the content in the MSB before the shift and places it in the MSB after the shift. Bit 0 is loaded into the C Flag of the Status Register.

Operation:

Instruction Format:
SRA src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if, before the shift, the LSB of src1_reg was set; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

Hex	Binary
R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Instruction
SRA R5

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1010	000	00101	000	00000	000	00101

1010_0101_1010_0101_1010_0101_1010_0101 ← R5
↓
↓ → → → → → → → →
1101_1010_0101_1010_0101_1010_0101_1010 1 → 0xDA5A5A5A

R5 ← 0xDA5A5A5A

C	N	O	P	Z
1	1	X	0	X

EXAMPLE 2

Contents

Hex	Binary
R8 = 0xFFFFFFFF	1111_1111_1111_1111_1111_1111_1111_1111

Instruction
SRA R8

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1010	000	01000	000	00000	000	01000

1111_1111_1111_1111_1111_1111_1111_1111 ← R8
↓
↓ → → → → → → → →
1111_1111_1111_1111_1111_1111_1111_1111 1 → 0xFFFFFFFF

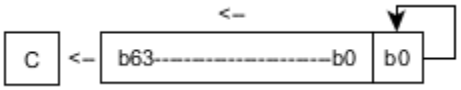
R8 ← 0xFFFFFFFF

C	N	O	P	Z
1	1	X	0	X

ARITHMETIC SHIFT LEFT

Description:
Shifts all bits of register src1_reg one place to the left. Copies the content in the LSB before the shift and places it in the LSB after the shift. Bit 63 is loaded into the C Flag of the Status Register.

Operation:



Instruction Format:
SLA src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1011	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if, before the shift, the MSB of Src1_reg was set; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

Hex	Binary
R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Instruction
SLA R5

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1011	000	00101	000	00000	000	00101

1010_0101_1010_0101_1010_0101_1010_0101 ← R5

↓

← ← ← ← ← ← ← ← ↓

1 0101_1010_0101_1010_0101_1010_0101_1011 → 0x5A5A5A5B

R5 ← 0x5A5A5A5B

C	N	O	P	Z
1	X	X	0	X

EXAMPLE 2

Contents

Hex	Binary
R8 = 0xFFFFFFFF	1111_1111_1111_1111_1111_1111_1111_1111

Instruction
SLA R8

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1011	000	01000	000	00000	000	00000

1111_1111_1111_1111_1111_1111_1111_1111 ← R8

↓

← ← ← ← ← ← ← ← ↓

1 1111_1111_1111_1111_1111_1111_1111_1111 → 0xFFFFFFFF

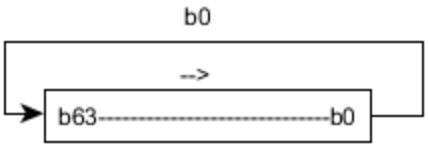
R8 ← 0xFFFFFFFF

C	N	O	P	Z
1	1	X	0	X

ROTATE RIGHT

Description:
Shifts all bits of src1_reg one place to the right. Copies the content in the LSB before the shift and places it in the MSB after the shift.

Operation:



Instruction Format:
RR Src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1100	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	⇔	⇔	⇔

N: Set if, before the rotate, the LSB of src1_reg was set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

Hex	Binary
R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Instruction
RR R5

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1100	000	00101	000	00000	000	00101

1010_0101_1010_0101_1010_0101_1010_0101 ← R5

→ → → → → → → →
1101_1010_0101_1010_0101_1010_0101_1010 1→ 0xDA5A5A5A
↑ ↓
← ← ← ← ← ← ← ←

R5 ← 0xDA5A5A5A

C	N	O	P	Z
0	0	X	0	X

EXAMPLE 2

Contents

Hex	Binary
R8 = 0x12A5011F	0001_0010_1010_0101_0000_0001_0001_1111

Instruction
RR R8

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1100	000	01000	000	00000	000	01000

0001_0010_1010_0101_0000_0001_0001_1111 ← R8

→ → → → → → → →
1000_1001_0101_0010_1000_0000_1000_1111 1 → 0x8952808F
↑ ↓
← ← ← ← ← ← ← ←

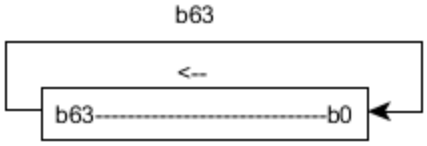
R8 ← 0x8952808F

C	N	O	P	Z
1	1	X	0	X

ROTATE LEFT

Description:
Shifts all bits of register src1_reg one place to the left. Copies the content in the MSB before the shift and places it in the LSB after the shift.

Operation:



Instruction Format:
RL src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1101	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	⇔	⇔	⇔

N: Set if, before the rotate, the MSB of Src1_reg was set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Instruction
RL R5

Contents

Hex	Binary
R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0001_1101	000	00101	000	00000	000	00101

1010_0101_1010_0101_1010_0101_1010_0101 ← R5

← ← ← ← ← ← ← ←

1 0101_1010_0101_1010_0101_1010_0101_1011 → 0x5A5A5A5B

↓ ↓

→ → → → → → → →

R5 ← 0x5A5A5A5B

C	N	O	P	Z
1	0	X	0	X

EXAMPLE 2

Instruction
RL R8

Contents

Hex	Binary
R8 = 0x12A5011F	0001_0010_1010_0101_0000_0001_0001_1111

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0000	000	00010	000	00101	000	00010

0001_0010_1010_0101_0000_0001_0001_1111 ← R8

← ← ← ← ← ← ← ←

0 0010_0101_0100_1010_0000_0010_0011_1110 → 0x254A023E

↓ ↑

→ → → → → → → →

R8 ← 0x254A023E

C	N	O	P	Z
0	0	X	0	X

JUMP IF CARRY

Description:
Jump to an address within the Program Memory if the Carry Flag of the STATUS REGISTER was set from the previous instruction.

Operation:
PC ← K

Instruction Format:
JC K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_0000	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

		Contents Hex	Binary
		R3 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
		R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
Instruction			
ADD R0, R3, R5	→	R0 = 0x25A5A5A5	0010_0101_1010_0101_1010_0101_1010_0101
JC HERE		Carry set to 1 after ADD	
.			
.			
.			
HERE	←	PC after Instructions	

C	N	O	P	Z
1	0	X	0	X

EXAMPLE 2

		Contents Hex	Binary
		R4 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
Instruction			
INC R4	→	R4 = 0x80000001	1000_0000_0000_0000_0000_0000_0000_0001
JC HERE		Carry not set after INC	
.	←	PC after Instructions	
.			
.			
HERE			

C	N	O	P	Z
0	1	X	0	X

JUMP IF NOT CARRY

Description:
Jump to an address within the Program Memory if the Carry Flag of the STATUS REGISTER was not set from the previous instruction.

Operation:
PC ← K

Instruction Format:
JNC LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_0001	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

		Contents Hex	Binary
		R3 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
		R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
Instruction			
ADD R0, R3, R5	→	R0 = 0x25A5A5A5	0010_0101_1010_0101_1010_0101_1010_0101
JNC HERE		Carry set to 1 after ADD	
.	←	PC after Instructions	
.			
.			
HERE			

C	N	O	P	Z
1	0	X	0	X

EXAMPLE 2

		Contents Hex	Binary
		R4 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
Instruction			
INC R4	→	R4 = 0x80000001	1000_0000_0000_0000_0000_0000_0000_0001
JNC HERE		Carry not set after INC	
.			
.			
.			
HERE	←	PC after Instructions	

C	N	O	P	Z
0	1	X	0	X

JUMP IF ZERO

Description:
Jump to an address within the Program Memory if the Zero Flag of the STATUS REGISTER was set from the previous instruction.

Operation:
PC ← K

Instruction Format:
JZ LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_0010	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents		
	Hex	Binary
	R3 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
	R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
Instruction		
ADD R0, R3, R5 →	R0 = 0x25A5A5A5	0010_0101_1010_0101_1010_0101_1010_0101
JZ HERE	Zero Flag set to 0 after ADD	
. ←	PC after Instructions	
.		
.		
HERE		

C	N	O	P	Z
1	0	X	0	0

EXAMPLE 2

Contents		
	Hex	Binary
	R5 = 0x00000001	1000_0000_0000_0000_0000_0000_0000_0001
Instruction		
DEC R5 →	R5 = 0x00000000	0000_0000_0000_0000_0000_0000_0000_0000
JZ HERE	Zero Flag set to 1 after DEC	
.		
.		
.		
HERE ←	PC after Instructions	

C	N	O	P	Z
0	0	X	0	1

JUMP IF NOT ZERO

Description:
Jump to an address within the Program Memory if the Zero Flag of the STATUS REGISTER was not set from the previous instruction.

Operation:
PC ← K

Instruction Format:
JNZ LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_0011	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

		Contents Hex	Binary
		R3 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
		R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
Instruction			
ADD R0, R3, R5	→	R0 = 0x25A5A5A5	0010_0101_1010_0101_1010_0101_1010_0101
JNZ HERE		Zero Flag set to 0 after ADD	
.			
.			
.			
HERE	←	PC after Instructions	

C	N	O	P	Z
1	0	X	0	0

EXAMPLE 2

		Contents Hex	Binary
		R5 = 0x00000001	1000_0000_0000_0000_0000_0000_0000_0001
Instruction			
DEC R5	→	R5 = 0x00000000	0000_0000_0000_0000_0000_0000_0000_0000
JNZ HERE		Zero Flag set to 1 after DEC	
.	←	PC after Instructions	
.			
.			
HERE			

C	N	O	P	Z
0	0	X	0	1

JUMP IF NEGATIVE

Description:
Jump to an address within the Program Memory if the Negative Flag of the STATUS REGISTER was set from the previous instruction.

Operation:
PC ← K

Instruction Format:
JN LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_0100	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Instruction		Contents Hex	Binary
		R3 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
		R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
ADD R0, R3, R5	→	R0 = 0x25A5A5A5	0010_0101_1010_0101_1010_0101_1010_0101
JN HERE		Negative Flag set to 0 after ADD	
.	←	PC after Instructions	
.			
.			
HERE			

C	N	O	P	Z
1	0	1	0	0

EXAMPLE 2

Instruction		Contents Hex	Binary
		R7 = 0x7F000000	0111_1111_0000_0000_0000_0000_0000_0000
SLL R7	→	R7 = 0xFE000000	1111_1110_0000_0000_0000_0000_0000_0000
JN HERE		Negative Flag set to 1 after SLL	
.			
.			
.			
HERE	←	PC after Instructions	

C	N	O	P	Z
0	1	0	0	0

JUMP IF PLUS

Description:
Jump to an address within the Program Memory if the Negative Flag of the STATUS REGISTER was not set from the previous instruction.

Operation:
PC ← K

Instruction Format:
JP LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_0101	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

		Contents	
		Hex	Binary
		R3 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
		R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
Instruction			
ADD R0, R3, R5	→	R0 = 0x25A5A5A5	0010_0101_1010_0101_1010_0101_1010_0101
JP HERE		Zero Flag set to 0 after ADD	
.			
.			
.			
HERE	←	PC after Instructions	

C	N	O	P	Z
1	0	1	0	0

EXAMPLE 2

		Contents	
		Hex	Binary
		R5 = 0x00000001	1000_0000_0000_0000_0000_0000_0000_0001
Instruction			
DEC R5	→	R5 = 0x00000000	0000_0000_0000_0000_0000_0000_0000_0000
JP HERE		Zero Flag set to 1 after DEC	
.	←	PC after Instructions	
.			
.			
HERE			

C	N	O	P	Z
0	0	0	0	1

JUMP IF OVERFLOW

Description:
Jump to an address within the Program Memory if the Overflow Flag of the STATUS REGISTER was set from the previous instruction.

Operation:
PC ← K

Instruction Format:
JO LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_0110	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

		Contents Hex	Binary
		R3 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
		R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
Instruction			
ADD R0, R3, R5	→	R0 = 0x25A5A5A5	0010_0101_1010_0101_1010_0101_1010_0101
JO HERE		Overflow Flag set to 1 after ADD	
.			
.			
.			
HERE	←	PC after Instructions	

C	N	O	P	Z
1	0	1	0	0

EXAMPLE 2

		Contents Hex	Binary
		R5 = 0x00000001	1000_0000_0000_0000_0000_0000_0000_0001
Instruction			
DEC R5	→	R5 = 0x00000000	0000_0000_0000_0000_0000_0000_0000_0000
JO HERE		Overflow Flag set to 0 after DEC	
.	←	PC after Instructions	
.			
.			
HERE			

C	N	O	P	Z
0	0	0	0	1

JUMP IF NO OVERFLOW

Description:
Jump to an address within the Program Memory if the O Flag of the STATUS REGISTER was not set from the previous instruction.

Operation:
PC ← K

Instruction Format:
JNO LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_0111	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents		
	Hex	Binary
	R3 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
	R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101
Instruction		
ADD R0, R3, R5 →	R0 = 0x25A5A5A5	0010_0101_1010_0101_1010_0101_1010_0101
JNO HERE	Overflow Flag set to 1 after ADD	
. ←	PC after Instructions	
.		
.		
HERE		

C	N	O	P	Z
1	0	X	0	0

EXAMPLE 2

Contents		
	Hex	Binary
	R5 = 0x00000001	1000_0000_0000_0000_0000_0000_0000_0001
Instruction		
DEC R5 →	R5 = 0x00000000	0000_0000_0000_0000_0000_0000_0000_0000
JNO HERE	Overflow Flag set to 0 after DEC	
.		
.		
.		
HERE ←	PC after Instructions	

C	N	O	P	Z
0	0	X	0	1

JUMP IF LESS THAN

Description:
Jump to an address within the Program Memory if the content in src_reg1 is less than the content in src_reg2. This instruction may also execute if the content in src_reg1 is less than an immediate value. Subtracts src1_reg from src2_reg or an immediate value and updates the Program Status Register. Executes the jump only if Negative Flag is not equal to the Overflow flag.

Operation:
If A < B, then PC ← K

Instruction Format:
JL LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_1000	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents	
Hex	Binary
R3 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
R5 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Instruction

CMP R3, R5		Overflow Flag set to 1
JL HERE		Negative Flag set to 0
.		
.		
.		
HERE	←	PC after Instructions

C	N	O	P	Z
X	0	1	X	X

EXAMPLE 2

Contents	
Hex	Binary
R1 = 0x7F000001	0111_1111_0000_0000_0000_0000_0000_0001
R7 = 0x7F000000	0111_1111_0000_0000_0000_0000_0000_0000

Instruction

CMP R1, R7		Overflow Flag set to 0
JL HERE		Negative Flag set to 0
.	←	PC after Instructions
.		
.		
HERE		

C	N	O	P	Z
X	0	0	X	X

JUMP IF GREATER THAN OR EQUAL

Description:
Jump to an address within the Program Memory if the content in src_reg1 is greater than or equivalent to the content in src_reg2. This instruction may also execute if the content in src_reg1 is greater than or equivalent to an immediate value. Subtracts src1_reg from src2_reg or an immediate value and updates the Program Status Register. Executes the jump only if Negative Flag is equal to the Overflow flag.

Operation:
If src1_reg >= src2_reg, then PC ← K

Instruction Format:
JGE LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_1001	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents	
Hex	Binary
R3 = 0x7F198456	0111_1111_0001_1001_1000_0100_0101_0110
R5 = 0x1205A310	0001_0010_0000_0101_1010_0011_0001_0000

Instruction		
CMP R3, R5		Overflow Flag set to 0
JGE HERE		Negative Flag set to 0
.		
.		
.		
HERE	←	PC after Instructions

C	N	O	P	Z
X	0	0	X	X

EXAMPLE 2

Contents	
Hex	Binary
R4 = 0x80000000	1000_0000_0000_0000_0000_0000_0000_0000
R7 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Instruction		
CMP R4, R7		Overflow Flag set to 1
JGE HERE		Negative Flag set to 0
.	←	PC after Instructions
.		
.		
HERE		

C	N	O	P	Z
X	0	1	X	X

JUMP IF GREATER THAN

Description:
Jump to an address within the Program Memory if the content in src_reg1 is greater than the content in src_reg2. This instruction may also execute if the content in src_reg1 is greater than an immediate value. Subtracts src1_reg from src2_reg or an immediate value and updates the Program Status Register. Executes the jump only if Negative Flag is equal to the Overflow flag and the Zero Flag is 0.

Operation:
If src1_reg > src2_reg, then PC ← K

Instruction Format:
JG LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_1010	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents	
Hex	Binary
R3 = 0x00053114	0000_0000_0000_0101_0011_0001_0001_0100
R5 = 0x00053113	0000_0000_0000_0101_0011_0001_0001_0011

Instruction		
CMP R3, R5		Overflow Flag set to 0
JG HERE		Negative Flag set to 0
.		Zero Flag set to 0
.		
.		
HERE	←	PC after Instructions

C	N	O	P	Z
X	0	0	X	X

EXAMPLE 2

Contents	
Hex	Binary
R3 = 0x00053114	0000_0000_0000_0101_0011_0001_0001_0100
R5 = 0x00053114	0000_0000_0000_0101_0011_0001_0001_0011

Instruction		
CMP R4, R7		Overflow Flag set to 0
JG HERE		Negative Flag set to 0
		Zero Flag set to 1
.	←	PC after Instructions
.		
.		
HERE		

C	N	O	P	Z
X	0	1	X	X

JUMP IF LESS THAN OR EQUAL

Description:
Jump to an address within the Program Memory if the content in src_reg1 is less than or equivalent to the content in src_reg2. This instruction may also execute if the content in src_reg1 is less than or equivalent to an immediate value. Subtracts src1_reg from src2_reg or an immediate value and updates the Program Status Register. Executes the jump only if the Sign Flag is not equal to the Overflow Flag or the Zero Flag is set to 1.

Operation:
If src1_reg <= src2_reg, then PC ← K

Instruction Format:
JLE LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_1011	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents

Hex	Binary
R0 = 0x00053114	0000_0000_0000_0101_0011_0001_0001_0100
R1 = 0x90053113	1001_0000_0000_0101_0011_0001_0001_0011

Instruction

CMP R0, R1 — Overflow Flag set to 1
JLE HERE — Negative Flag set to 0
. — Zero Flag set to 0
.
.
HERE ← PC after Instructions

C	N	O	P	Z
X	0	0	X	0

EXAMPLE 2

Contents

Hex	Binary
R6 = 0xFF05931A	1111_1111_0000_0101_1001_0011_0001_1010
R9 = 0xFF05931A	1111_1111_0000_0101_1001_0011_0001_1010

Instruction

CMP R6, R9 — Overflow Flag set to 0
JLE HERE — Negative Flag set to 0
. — Zero Flag set to 0
.
.
HERE ← PC after Instructions

C	N	O	P	Z
X	0	1	X	X

JUMP IF BELOW

Description:
Jump to the next instruction if the content in src_reg1 is greater less than the content in src_reg2. This instruction may also execute if the content in src_reg1 less than an immediate value. This instruction will execute the jump only if the Carry Flag is set to 1 immediate after the previous instruction is executed.

Operation:
If src1_reg < src2_reg, then PC ← K

Instruction Format:
JB LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_1100	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents

Hex	Binary
R1 = 0xF1054A41	1111_0001_0000_0101_0100_1010_0100_0001
R2 = 0x10010A74	0001_0000_0000_0001_0000_1010_0111_0100

Instruction

ADD R3, R1, R2 — Carry Flag set to 1

JB HERE

.

.

.

HERE ← PC after Instructions

C	N	O	P	Z
1	X	X	X	X

EXAMPLE 2

Contents

Hex	Binary
R6 = 0x1F05531A	0001_1111_0000_0101_0101_0011_0001_1010

Instruction — Carry Flag set to 0

SRL R6

JB HERE

. ← PC after Instructions

.

.

HERE

C	N	O	P	Z
0	X	X	X	X

JUMP IF ABOVE OR EQUAL

Description:
Jump to an address within the Program Memory if the content in src_reg1 is greater than or equivalent to the content in src_reg2. This instruction may also execute if the content in src_reg1 is greater than or equivalent to an immediate value. This instruction will execute the jump only if the Carry Flag is set to 0 immediate after the previous instruction is executed.

Operation:
If src1_reg >= src2_reg, then PC ← K

Instruction Format:
JAE LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_1101	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents

Hex	Binary
R1 = 0xF1054A41	1111_0001_0000_0101_0100_1010_0100_0001
R2 = 0x10010A74	0001_0000_0000_0001_0000_1010_0111_0100

Instruction

ADD R3, R1, R2 — Carry Flag set to 1

JAE HERE

. ← PC after Instructions

.

.

HERE

C	N	O	P	Z
1	X	X	X	X

EXAMPLE 2

Contents

Hex	Binary
R6 = 0x1F05531A	0001_1111_0000_0101_0101_0011_0001_1010

Instruction — Carry Flag set to 0

SRL R6

JAE HERE

.

.

.

HERE ← PC after Instructions

C	N	O	P	Z
0	X	X	X	X

JUMP IF ABOVE

Description:
Jump to an address within the Program Memory if the content in src_reg1 is greater the content in src_reg2. This instruction may also execute if the content in src_reg1 is greater than an immediate value. This instruction will execute the jump only if the Carry Flag is set to 0 and the Zero Flag is set to 0 immediate after the previous instruction is executed.

Operation:
If src1_reg > src2_reg, then PC ← K

Instruction Format:
JA LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_1110	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents

Hex	Binary
R3 = 0x00000001	0000_0000_0000_0000_0000_0000_0000_0001

Instruction

DEC R3		Carry Flag set to 0
JA HERE		Zero Flag set to 1
.	←	PC after Instructions
.		
.		
HERE		

C	N	O	P	Z
0	X	X	X	1

EXAMPLE 2

Contents

Hex	Binary
R6 = 0x1F05531A	0001_1111_0000_0101_0101_0011_0001_1010

Instruction

SRL R6		Carry Flag set to 0
JA HERE		Overflow Flag Set to 0
.		
.		
.		
HERE	←	PC after Instructions

C	N	O	P	Z
0	X	X	X	X

JUMP IF BELOW OR EQUAL

Description:
Jump to an address within the Program Memory if the content in src1_reg is less than or equivalent to the content in src2_reg. This instruction may also execute if the content in src_reg1 is less than or equivalent to an immediate value. This instruction will execute the jump only if the Carry Flag is set to 1 or the Zero Flag is set to 1 immediate after the previous instruction is executed.

Operation:
If src1_reg <= src2_reg, then PC ← K

Instruction Format:
JBE LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0010_1111	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE 1

Contents

Hex	Binary
R4 = 0xF1111111	1111_0001_0001_0001_0001_0001_0001_0001
R6 = 0x10000000	0001_0000_0000_0000_0000_0000_0000_0000

Instruction

ADD R5, R4, R6 — Carry Flag set to 1

JBE HERE — Zero Flag set to 0

.

.

.

HERE ← PC after Instructions

C	N	O	P	Z
1	X	X	X	0

EXAMPLE 2

Contents

Hex	Binary
R6 = 0x00000010	0000_0000_0000_0000_0000_0000_0001_0000

Instruction

SRL R6 — Carry Flag set to 0

JBE HERE — Zero Flag Set to 0

. ← PC after Instructions

.

.

HERE

C	N	O	P	Z
0	X	x	X	0

JUMP (relative)

Description:
Jump to an address within the Program Memory. Then program counter gets an immediate value.

Operation:
PC ← K

Instruction Format:
JMP LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0011_0000	24 – bit Signed displacement

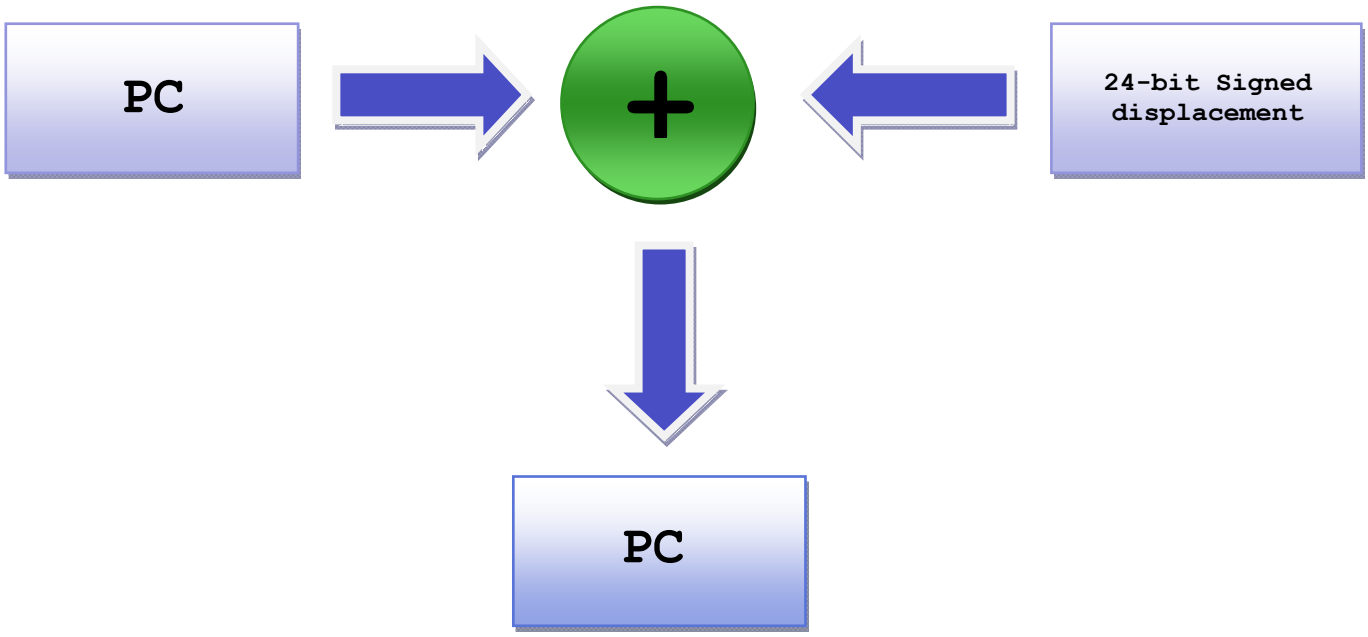
Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Instruction
JMP HERE
ADD R0, R1, R2
.
.
.
SUB R5, R4, R3
HERE ← PC after Instructions

ILLUSTRATION



JUMP (register)

Description:
Jump to an address within the Program Memory. The Program Counter gets the value from the src1_reg.

Operation:
PC ← src1_reg

Instruction Format:
JMP src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0011_0001	0 0 0	src1_reg	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Status Register:

C	N	O	P	Z
-	-	-	-	-

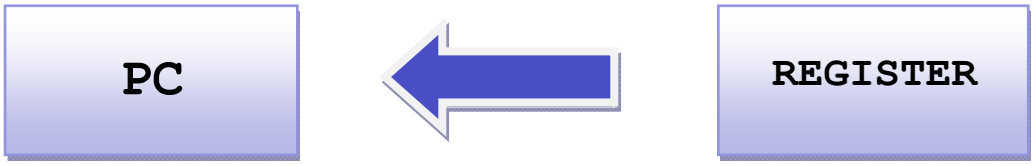
EXAMPLE

Contents
R1 = 0x0000000F

Location	Instruction
0x00000002	JMP R1
0x00000003	ADD R0, R1, R2
.	
.	
.	
SUB R5, R4, R3	
0x0000000F	HERE

← PC after Instructions

ILLUSTRATION



CALL (relative)

Description:
Jump to an address within the Program Memory. The Stack Pointer gets the value of the Program Counter + 1. After then instruction is executed, the content at the top of the Stack gets popped into the PC. This returns the Program Counter back to the next instruction before the call was made.

Operation:
SP ← PC + 1, PC ← K

Instruction Format:
LCALL LABEL

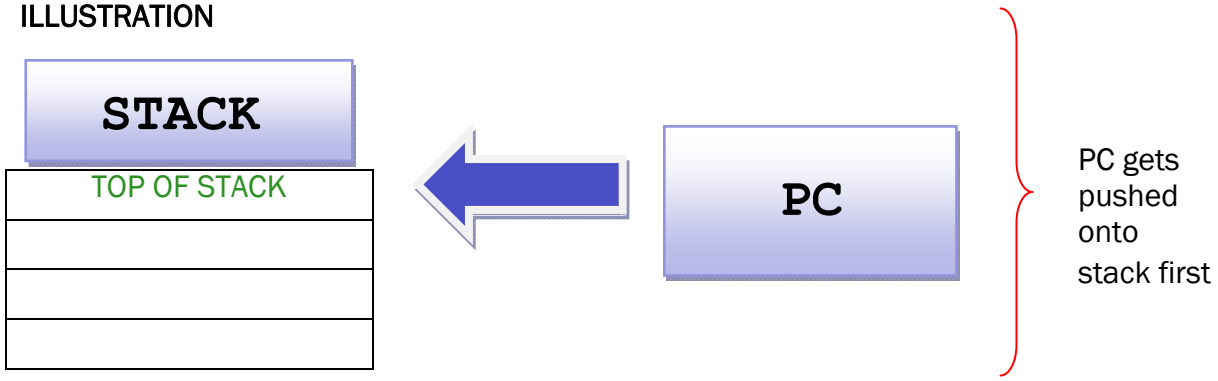
32-bit Opcode:

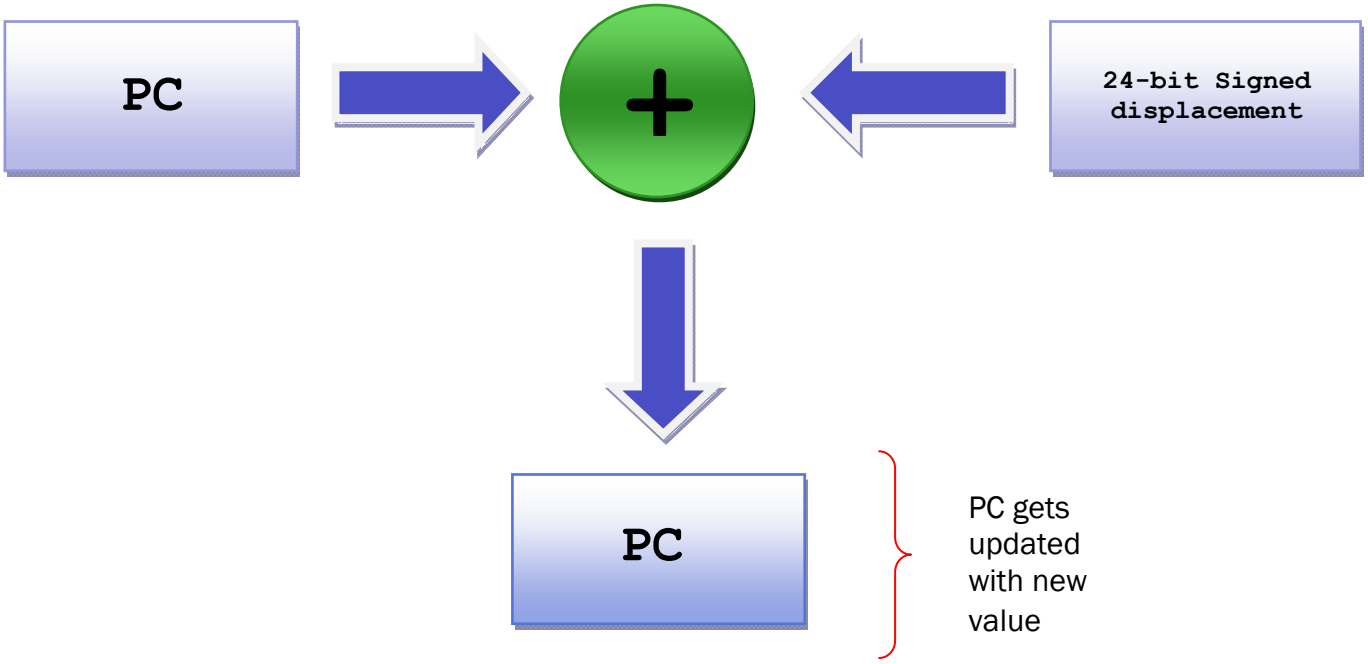
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0011_0010	24 – bit Signed displacement

Status Register:

C	N	O	P	Z
-	-	-	-	-

ILLUSTRATION





CALL (register)

Description:
Jump to an address within the Program Memory. The Stack Pointer gets the value of the Program Counter. The Program Counter gets the value from src1_reg. After then instruction is executed, the content at the top of the Stack gets popped into the PC. This returns the Program Counter back to the next instruction before the call was made.

Operation:
SP ← PC + 1, PC ← src1_reg

Instruction Format:
LCALL src1_reg

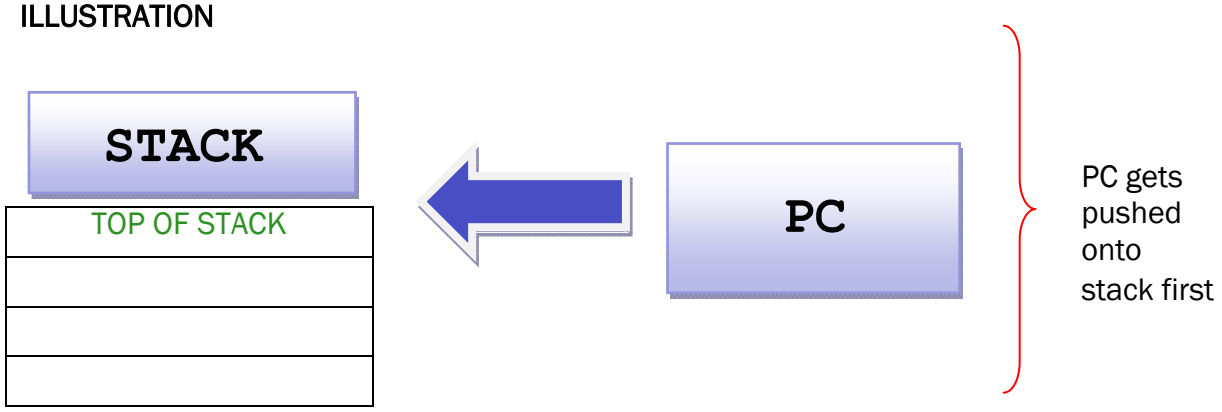
32-bit Opcode:

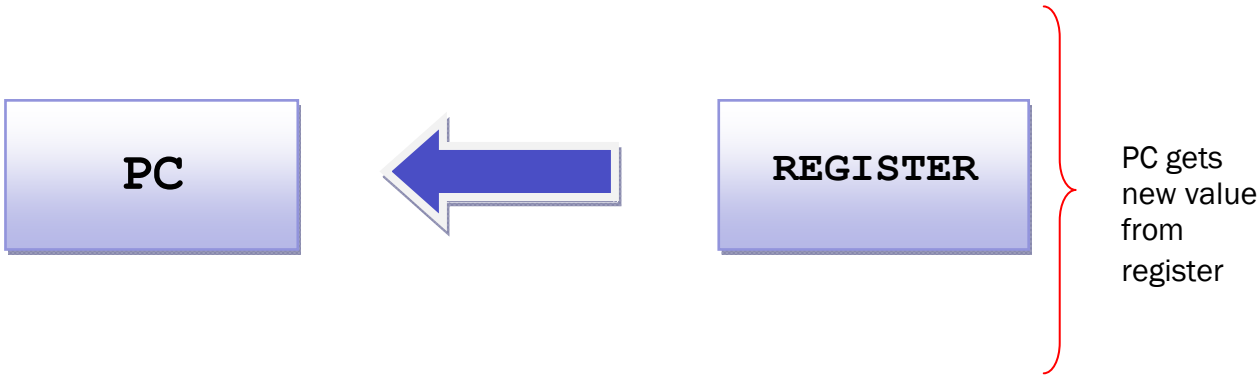
31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0011_0011	0 0 0	src1_reg	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Status Register:

C	N	O	P	Z
-	-	-	-	-

ILLUSTRATION





RETURN

Description:
Jumps to the address stored in the Stack Pointer. Before a call is executed, the address of the next instruction gets pushed onto the Stack. After the call is executed, the Stack is popped into the PC, therefore returning the Program Counter to the next instruction.

Operation:
PC ← SP

Instruction Format:
RET

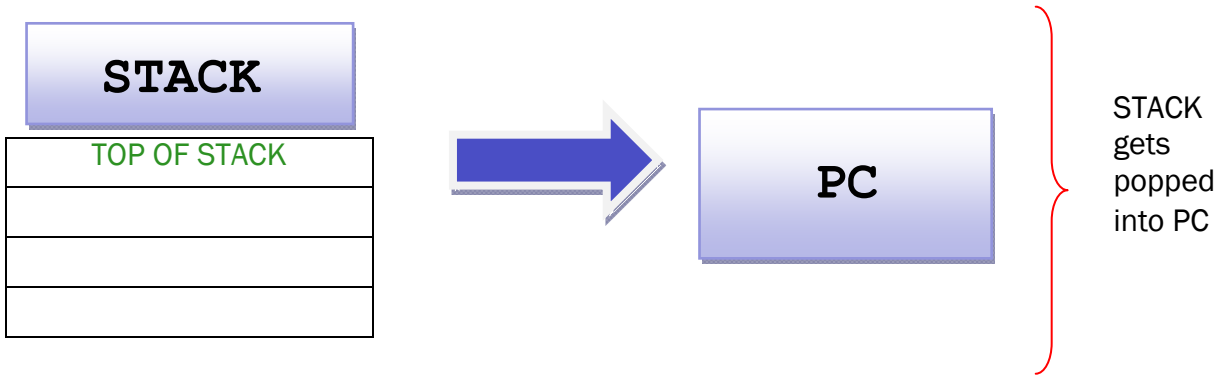
32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0011_0100	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

ILLUSTRATION



RETURN FROM INTERRUPT

Description:
Jumps to the address stored in the Stack Pointer. When an interrupt is invoked, the address of the next instruction gets pushed onto the Stack. After the interrupt is executed, the Stack is popped into the PC, therefore returning the Program Counter to the next instruction.

Operation:
PC ← SP

Instruction Format:
RETI

32-bit Opcode:

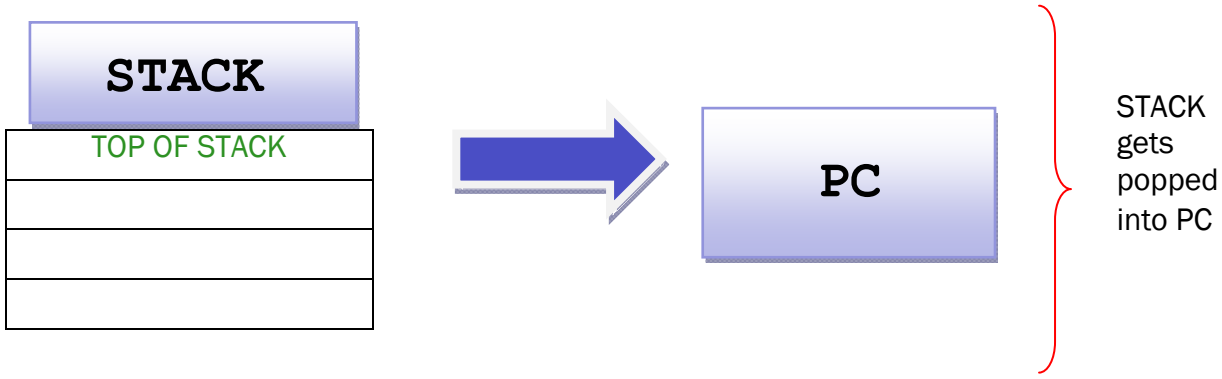
31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0011_0101	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

```
        ...
extint: PUSH R0 ; Save R0 on the Stack
        ...
        POP R0 ; Restore R0
        RETI ; Return and enable interrupts
```



CLEAR CARRY

Description:
Set the content of the Carry Flag to 0.

Operation:
Carry Flag ← 0

Instruction Format:
CLC

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0000	000000000000000000000000

Status Register:

C	N	O	P	Z
0	-	-	-	-

C: Set to 0.

EXAMPLE 1

Contents	
Hex	Binary
R4 = 0xF1111111	1111_0001_0001_0001_0001_0001_0001_0001
R5 = 0x10000000	0001_0000_0000_0000_0000_0000_0000_0000

Instruction

ADD R3, R4, R5 — Carry Flag set to 1

CLC — Carry Flag set to 0

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0000	000000000000000000000000

C	N	O	P	Z
0	X	X	X	X

EXAMPLE 2

Contents	
Hex	Binary
R4 = 0xF1111111	1111_0001_0001_0001_0001_0001_0001_0001

Instruction

SLL R4 — Carry Flag set to 1

CLC — Carry Flag set to 0

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0000	000000000000000000000000

C	N	O	P	Z
0	X	X	X	X

SET CARRY

Description:
Set the content in the Carry Flag to 1.

Operation:
Carry Flag ← 1

Instruction Format:
STC

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0001	000000000000000000000000

Status Register:

C	N	O	P	Z
1	-	-	-	-

C: Set to 1.

EXAMPLE 1

	Contents	
	Hex	Binary
	R1 = 0x01046458	0000_0001_0000_0100_0110_0100_0101_1000
	R2 = 0x10000F51	0001_0000_0000_0000_0000_1111_0101_0001
Instruction		
ADD R7, R1, R2	Carry Flag set to 0	
STC	Carry Flag set to 1	

Binary Instruction Format

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0100_0001	000000000000000000000000

C	N	O	P	Z
1	X	X	X	X

EXAMPLE 2

	Contents	
	Hex	Binary
	R4 = 0x00001111	0000_0000_0000_0000_0001_0001_0001_0001
Instruction		
SRL	Carry Flag set to 1	
STC	Carry Flag set to 1	

Binary Instruction Format

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0100_0001	000000000000000000000000

C	N	O	P	Z
1	X	X	X	X

COMPLEMENT CARRY

Description:
Set the Carry Flag to 1 if the previous content was 0. Set the Carry Flag to 0 if the previous content was 1.

Operation:
If Carry Flag = 1, then Carry Flag ← 0. If Carry Flag = 0, then Carry Flag ← 1.

Instruction Format:
CMC

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0010	000000000000000000000000

Status Register:

C	N	O	P	Z
⇔	-	-	-	-

C: Complement the content in the flag.

EXAMPLE 1

	Contents	
	Hex	Binary
	R1 = 0x01046458	0000_0001_0000_0100_0110_0100_0101_1000
	R2 = 0x10000F51	0001_0000_0000_0000_0000_1111_0101_0001
Instruction		
ADD R7, R1, R2	Carry Flag set to 0	
CMC	Carry Flag set to 1	

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0010	000000000000000000000000

C	N	O	P	Z
1	X	X	X	X

EXAMPLE 2

	Contents	
	Hex	Binary
	R4 = 0x00001111	0000_0000_0000_0000_0001_0001_0001_0001
Instruction		
SRL	Carry Flag set to 1	
CMC	Carry Flag set to 0	

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0010	000000000000000000000000

C	N	O	P	Z
0	X	X	X	X

CLEAR INTR. ENABLE

Description:
The interrupt enable allows the processor to receive external interrupts. This instruction clears the interrupt enable, disabling the interrupt.

Operation:
Interrupt Enable ← 0

Instruction Format:
CLI

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0011	000000000000000000000000

Status Register:

C	N	O	P	Z
-	-	-	-	-

SET INTR. ENABLE

Description:
The interrupt enable allows the processor to receive external interrupts. This instruction sets the interrupt enable, enabling the interrupt.

Operation:
Interrupt Enable ← 1

Instruction Format:
STI

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0100	000000000000000000000000

Status Register:

C	N	O	P	Z
-	-	-	-	-

HALT

Description:
The processor enters an idle state when there is no immediate instruction to be done. The CPU halts until the next external interrupt if executed.

Operation:
PC ← PC

Instruction Format:
HALT

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0100_0101	000000000000000000000000

Status Register:

C	N	O	P	Z
-	-	-	-	-

ADDI

Description:
Performs an arithmetic addition between the content of src1_reg and an immediate value and places the result into the dest_reg

Operation:
dest_reg ← src1_reg + K

Instruction Format:
ADDI dest_reg, src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_0000	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

Hex	Binary
R2 = 0x02F11058	0000_0010_1111_0001_0001_0000_0101_1000
Immediate Value	0000_0000_0000_0000_0000_0000_1001_0111

Instruction

ADDI R0, R2, 0x00000097

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_0000	000	00010	0x00000097	000	00000

```
  0000_0010_1111_0001_0001_0000_0101_1000
+ 0000_0000_0000_0000_0000_0000_1001_0111
  0000_0010_1111_0001_0001_0000_1110_1111   →   0x02F110EF
```

R0 ← 0x0x02F110EF

C	N	O	P	Z
X	X	X	1	X

EXAMPLE 2

Contents

Hex	Binary
R3 = 0xC2D5945A	1100_0010_1101_0101_1001_0100_0101_1010
Immediate Value	0000_0000_0000_0000_0000_0000_0001_0001

Instruction

ADDI R1, R3, 0x00000011

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_0000	000	00011	0x00000011	000	00001

```
  1100_0010_1101_0101_1001_0100_0101_1010
+ 0000_0000_0000_0000_0000_0000_0001_0001
  1100_0010_1101_0101_1001_0100_0110_1011   →   0xC2D5946B
```

R1 ← 0xC2D5946B

C	N	O	P	Z
X	1	X	0	X

SUBI

Description:
Performs an arithmetic subtraction between the content of src1_reg and an immediate value and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg - K

Instruction Format:
SUBI dest_reg, src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_0001	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents	
Hex	Binary
R2 = 0x02F11058	0000_0010_1111_0001_0001_0000_0101_1000
Immediate Value	0000_0000_0000_0000_0000_0000_1001_0111

Instruction
SUBI R0, R2, 0x00000097

```
  0000_0010_1111_0001_0001_0000_0101_1000
- 0000_0000_0000_0000_0000_0000_1001_0111
  0000_0010_1111_0001_0000_1111_1100_0001   →   0x02F10FC1
```

R0 ← 0x0x02F10FC1

C	N	O	P	Z
X	X	X	1	X

EXAMPLE 2

Contents	
Hex	Binary
R3 = 0xC2D5945A	1100_0010_1101_0101_1001_0100_0101_1010
Immediate Value	0000_0000_0000_0000_0001_0001_0001_0001

Instruction
SUBI R1, R3, 0x00001111

```
  1100_0010_1101_0101_1001_0100_0101_1010
- 0000_0000_0000_0000_0001_0001_0001_0001
  1100_0010_1101_0101_1000_0011_0100_1001   →   0xC2D58349
```

R1 ← 0xC2D5A56B

C	N	O	P	Z
X	1	X	0	X

MULI

Description:
Performs an arithmetic multiplication between the content of src1_reg and an immediate value and places the lower 64-bit of the result into the dest_reg.

Operation:
Product[127:64]:dest_reg ← src1_reg * K

Instruction Format:
MULI dest_reg, src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_0010	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

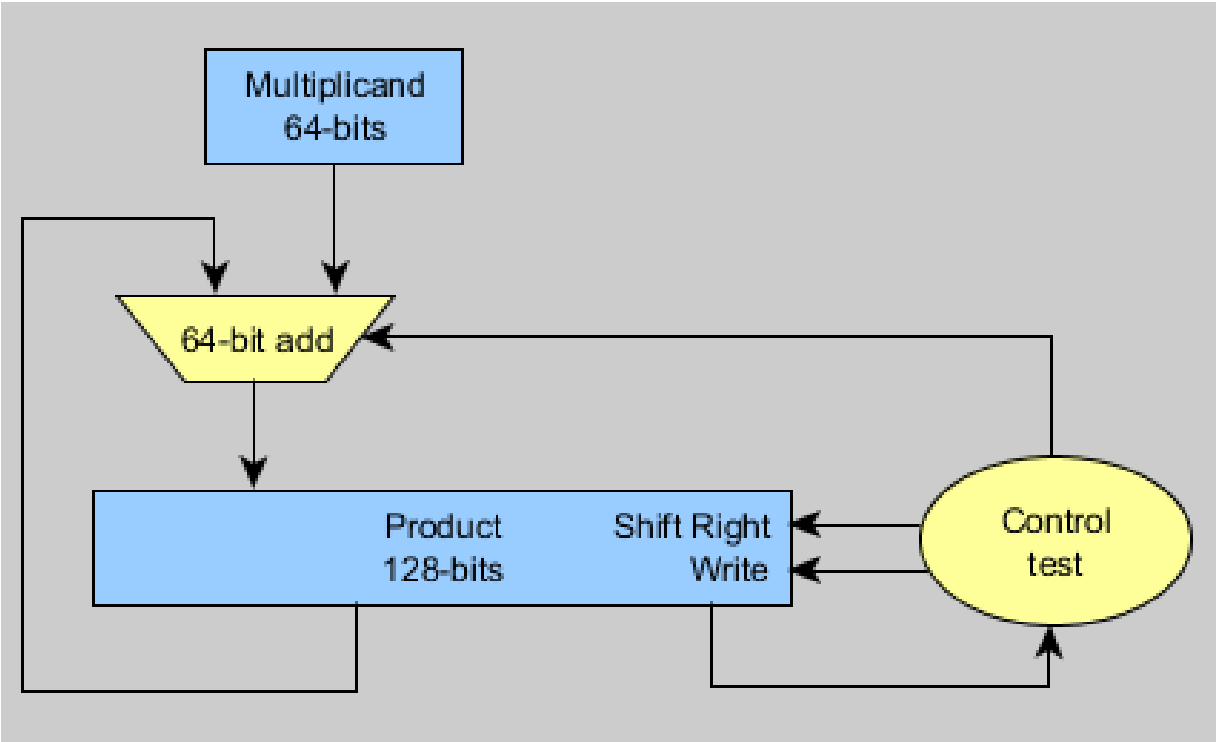
C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

ILLUSTRATION



DIVI

Description:
Performs an arithmetic division between the content of src1_reg and an immediate value and places the lower 64-bit of the result into the dest_reg.

Operation:
Quotient[63:0]:dest_reg ← src1_reg + K

Instruction Format:
DIVI dest_reg, src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_0011	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

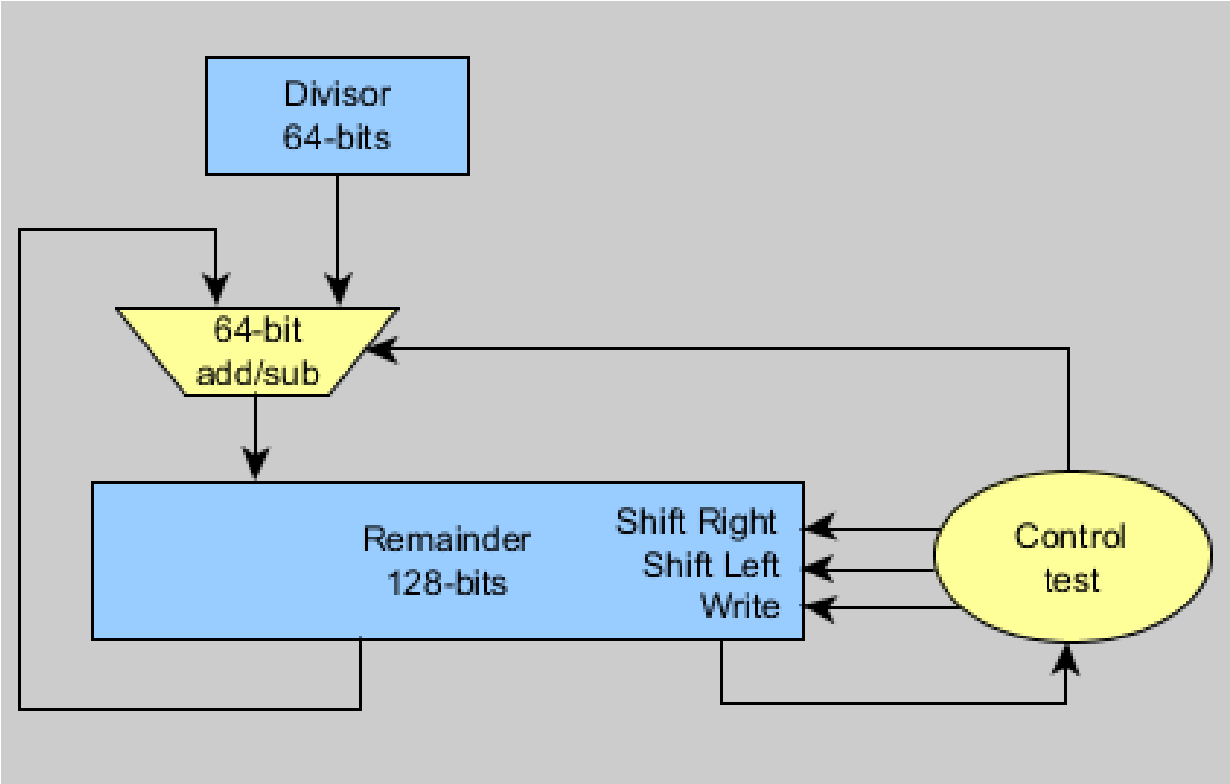
C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

ILLUSTRATION



ANDI

Description:
Performs a logical AND between the contents src1_reg and an immediate value and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg & K

Instruction Format:
ANDI dest_reg, src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1000	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

Hex	Binary
R2 = 0x654A3C51	0110_0101_0100_1010_0011_1100_0101_0001

Instruction
ANDI R0, R2, 0x00000013

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1000	000	00010	0x00000013	000	00000

```
  0110_0101_0100_1010_0011_1100_0101_0001
& 0000_0000_0000_0100_0000_0000_0001_0011
  0000_0000_0000_0000_0000_0000_0001_0001  →  0x00000011
```

R0 ← 0x00000011

C	N	O	P	Z
X	X	X	0	X

EXAMPLE 2

Contents

Hex	Binary
R2 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Instruction
ANDI R4, R2, 0x00000055

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1000	000	00010	0x00000055	000	00100

```
  1010_0101_1010_0101_1010_0101_1010_0101
& 0000_0000_0000_0000_0000_0000_0101_0101
  0000_0000_0000_0000_0000_0000_0000_0101  →  0x00000005
```

R4 ← 0x00000005

C	N	O	P	Z
X	X	X	0	X

ORI

Description:
Performs a logical OR between the content of src1_reg and an immediate value and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg | K

Instruction Format:
ORI dest_reg, src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1001	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

Hex	Binary
R2 = 0x654A3C51	0110_0101_0100_1010_0011_1100_0101_0001

Instruction
ORI R0, R2, 0x00000013

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1001	000	00010	0x00000013	000	00000

```
  0110_0101_0100_1010_0011_1100_0101_0001
| 0000_0000_0000_0000_0000_0000_0001_0011
  0110_0101_0100_1010_0011_1100_0101_0011  →  0x654A3C53
```

R0 ← 0x654A3C53

C	N	O	P	Z
X	X	X	0	X

EXAMPLE 2

Contents

Hex	Binary
R2 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Instruction
ORI R4, R2, 0x00000055

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1000	000	00010	0x00000055	000	00100

```
  1010_0101_1010_0101_1010_0101_1010_0101
| 0000_0000_0000_0000_0000_0000_0101_0101
  1010_1010_1010_1010_1010_1010_1111_1111  →  0xAAAAAAFF
```

R4 ← 0xFAAAAAAFF

C	N	O	P	Z
X	X	X	0	X

XORI

Description:
Performs a logical XOR between the content src1_reg and an immediate value and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg ^ K

Instruction Format:
XORI dest_reg, src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1010	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE 1

Contents

Hex	Binary
R5 = 0x5A36752C	0101_1010_0011_0110_0111_0101_0010_1100

Instruction
XORI R3, R5, 0x00000063

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1010	000	00101	0x00000063	000	00011

```
  0101_1010_0011_0110_0111_0101_0010_1100
  0000_0000_0000_0000_0000_0000_0110_0011
  0101_1010_0011_0110_0111_0101_0100_1111  →  0x5A36754F
```

R3 ← 0x5A36754F

C	N	O	P	Z
X	X	X	0	X

EXAMPLE 2

Contents

Hex	Binary
R9 = 0xA5A5A5A5	1010_0101_1010_0101_1010_0101_1010_0101

Instruction
XORI R5, R9, 0x0000005A

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1000	000	01000	0x0000005A	000	00101

```
  1010_0101_1010_0101_1010_0101_1010_0101
  0000_0000_0000_0000_0000_0000_0101_1010
  1010_0101_1010_0101_1010_0101_1111_1111  →  0xA5A5A5FF
```

R4 ← 0xA5A5A5FF

C	N	O	P	Z
X	X	X	0	1

CMPI

Description:
This instruction will compare the content of src1_reg and an immediate value. This operation subtracts the immediate value from the contents in src1_reg. Updates the Program Status Register.

Operation:
src1_reg - K

Instruction Format:
CMPI src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1011	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if the absolute value of the contents of src1_reg is larger than the absolute value of dest_reg; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if two's complement overflow resulted from the operation; cleared otherwise.

P: Set if the number of 0's is equivalent to the number of 1's in the result.

Z: Set if the result is 0; cleared otherwise.

TESTI

Description:
Tests if an immediate value is zero or negative. Performs a logical AND between the immediate value and itself.

Operation:
dest_reg ← K & K

Instruction Format:
TSTI K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0101_1100	000	src1_reg	8 – bit immediate value	000	dest_reg

Status Register:

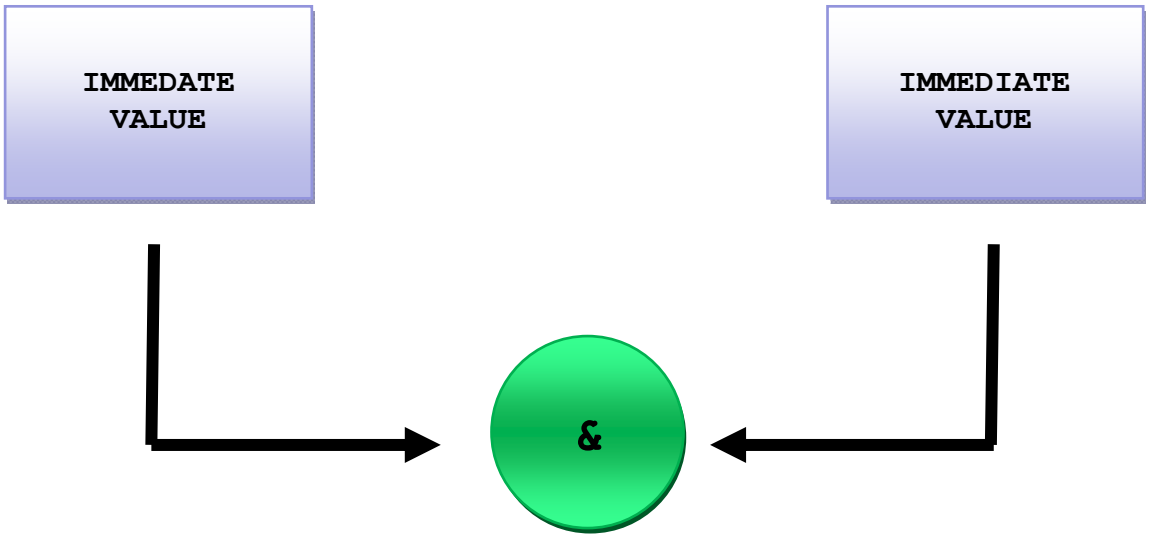
C	N	O	P	Z
-	⇔	0	-	⇔

N: Set if MSB of the result is set; cleared otherwise.

V: 0; Cleared

Z: Set if the result is 0; cleared otherwise.

EXAMPLE



C	N	O	P	Z
-	⇔	0	⇔	⇔

UPDATED STATUS REGISTERS

F_ADD

Description:
Adds two floating point value and places the result in the dest_reg.

Operation:
dest_reg ← src1_reg + src2_reg

Instruction Format:
FADD dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0000	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Instruction
FADD R0, R2, R5

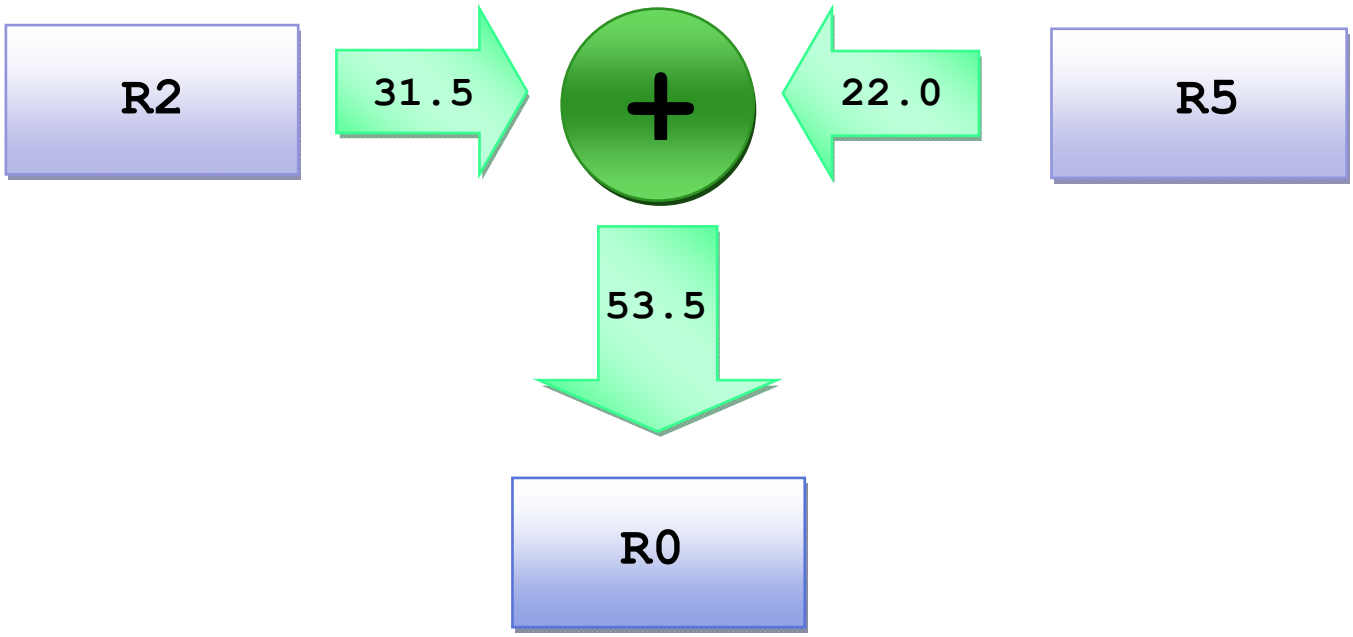
Contents

Real
R2 = 31.5
R5 = 22.0

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0000	000	00100	000	00101	000	00000

```
   31.5
+  22.0
   53.5   →   R0
```



F_SUB

Description:
Subtracts two floating point value and places the result in the dest_reg.

Operation:
dest_reg ← src1_reg – src2_reg

Instruction Format:
FSUB dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Instruction
FSUB R1, R3, R4

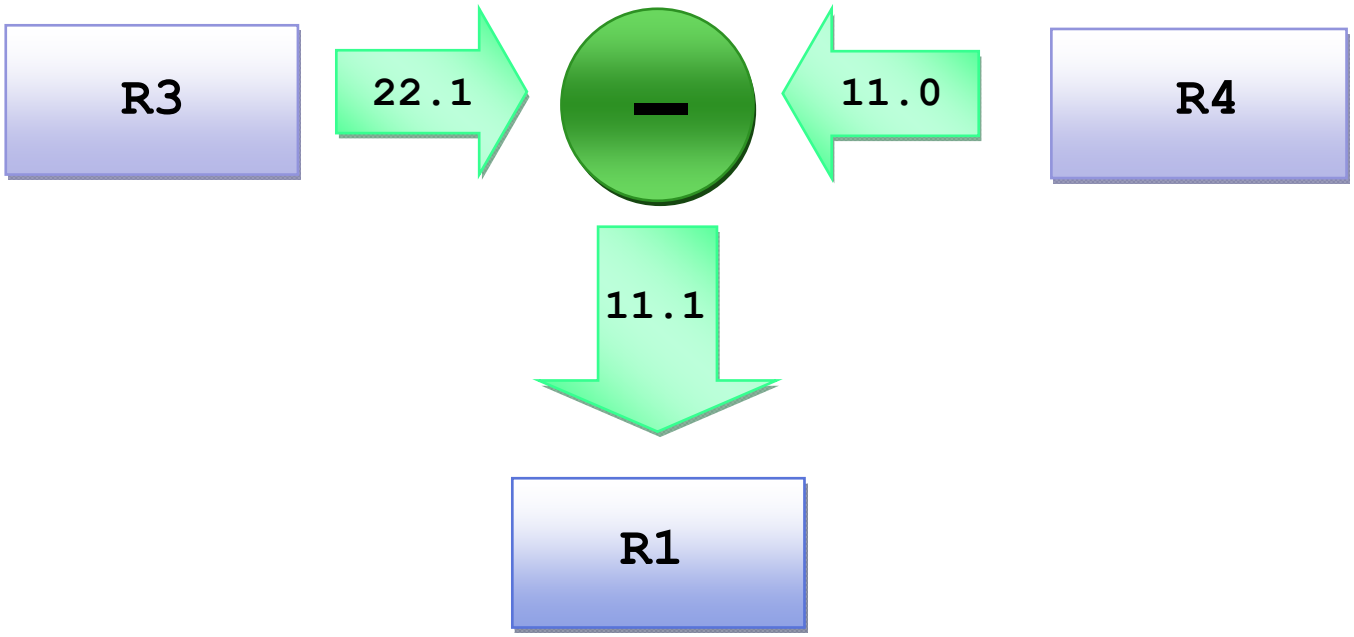
Contents

Real
R3 = 22.1
R4 = 11.0

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0001	000	00100	000	00101	000	00000

```
   22.1
-  11.0
   11.1    →    R1
```



F_MUL

Description:
Performs a 64-bit multiplication between the contents of two floating point registers and places the result in the dest_reg.

Operation:
dest_reg ← src1_reg * src2_reg

Instruction Format:
FMUL dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Contents

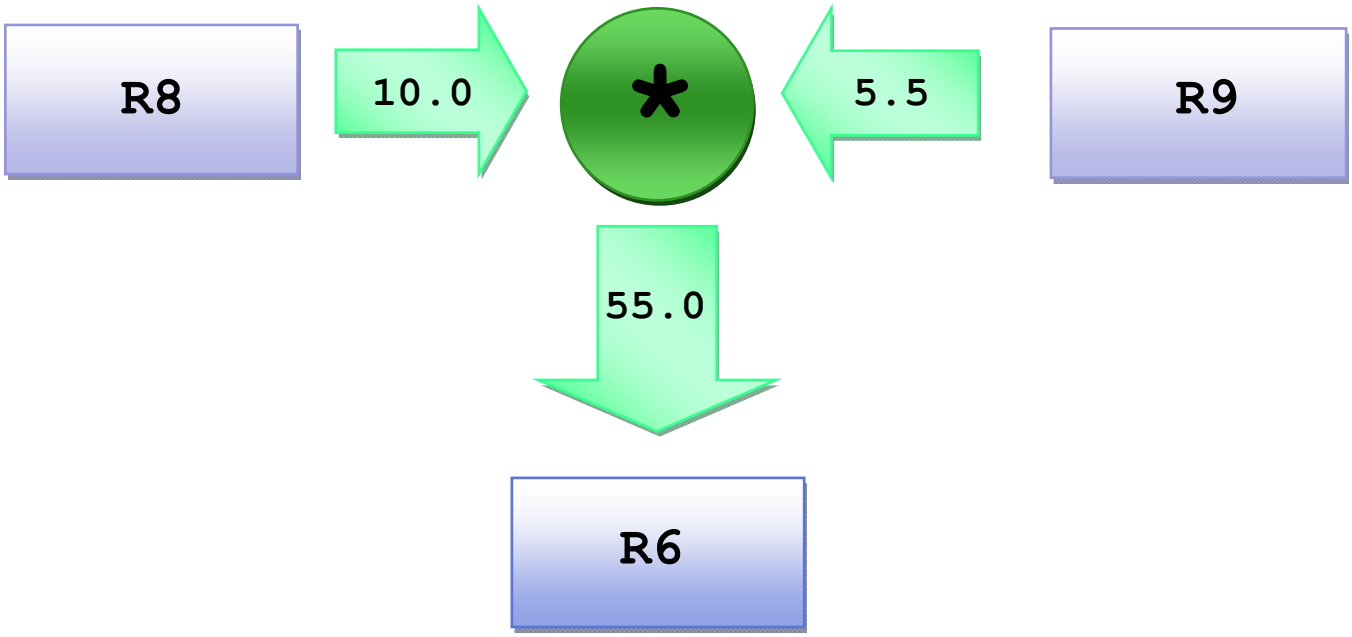
Real
R8 = 5.5
R9 = 10.0

Instruction
FMUL R6, R8, R9

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0010	000	01000	000	01001	000	00110

```
    10.0
*    5.5
    55.0  →   R6
```



F_DIV

Description:
Performs an arithmetic division between the contents of two floating point registers and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg / src2_reg

Instruction Format:
FDIV dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0011	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Instruction
FDIV R2, R5, R6

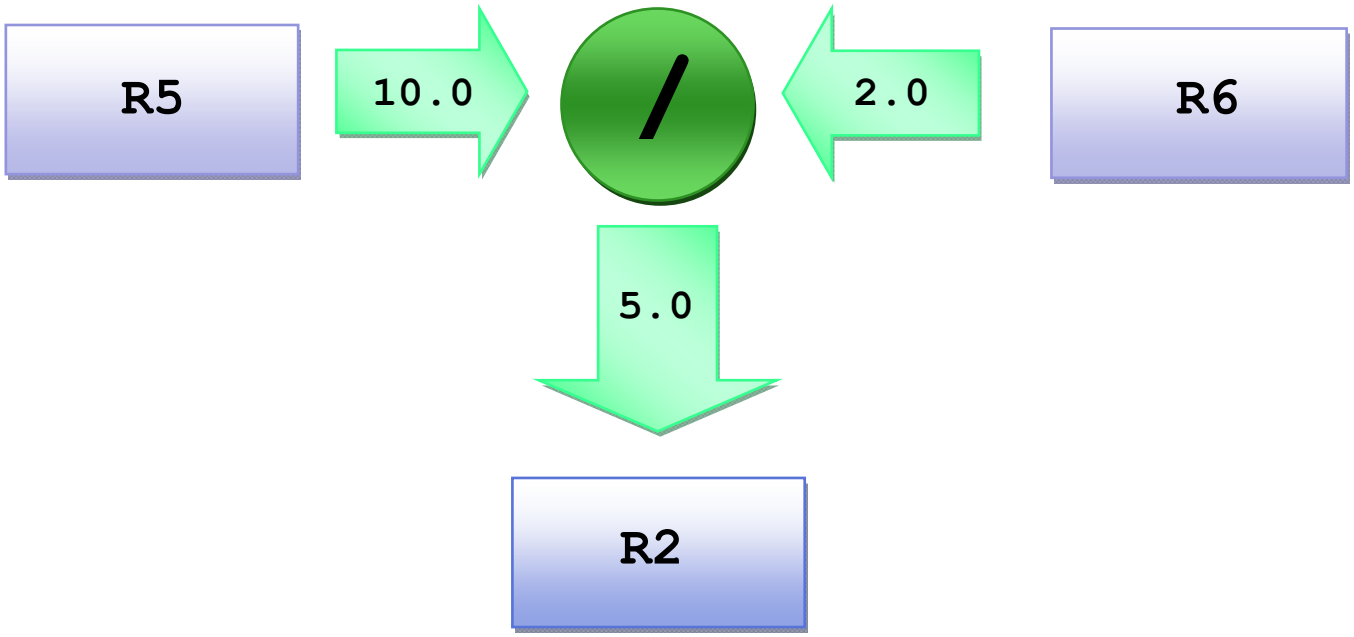
Contents

Real
R5 = 10.0
R6 = 2.0

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0011	000	00100	000	00101	000	00000

```
   10.0
÷   2.0
   5.0  →   R2
```



F_INC

Description:
Adds -1.0- one to the content of a floating point register and places the result int0 the dest_reg.

Operation:
dest_reg ← src1_reg + 1

Instruction Format:
FINC dest_reg, src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0100	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Contents

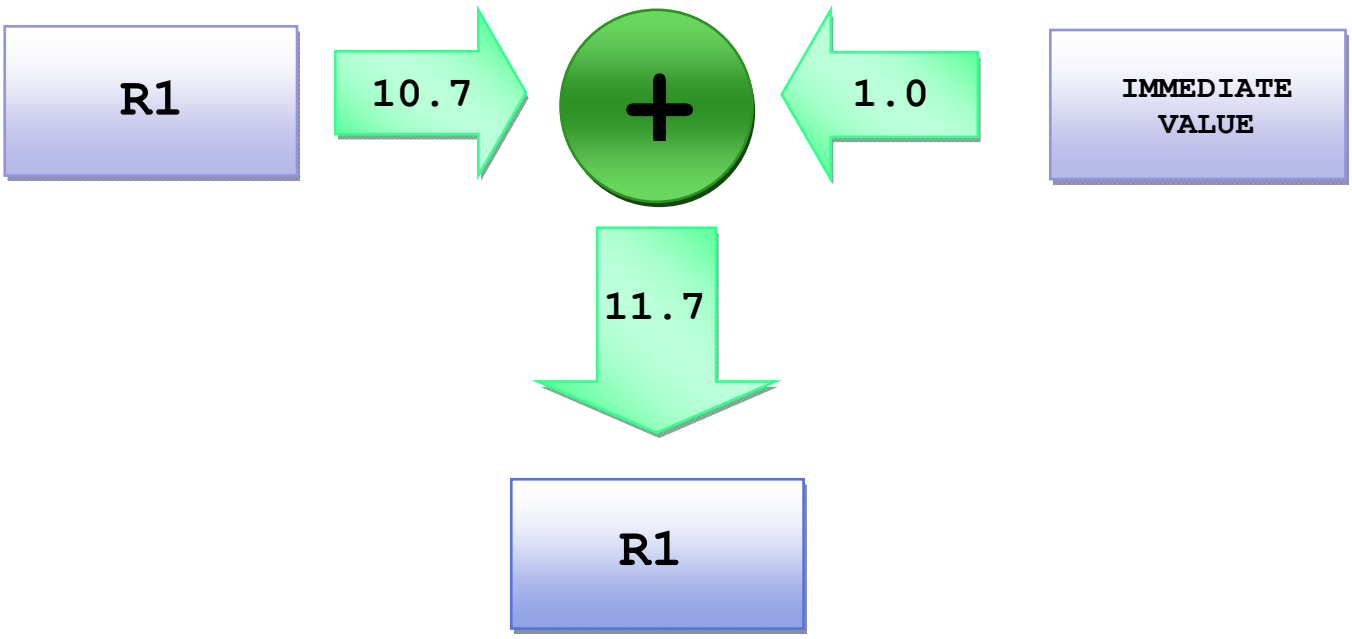
Real
R1 = 10.7

Instruction
FINC R1

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0100	000	00001	000	00000	000	00001

```
   10.7
+   1.0
   11.7  →   R1
```



F_DEC

Description:
Subtract -1.0- one from the content of a floating point and places the result in the dest_reg.

Operation:
dest_reg ← src1_reg - 1

Instruction Format:
FDEC dest_reg, src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0101	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Contents

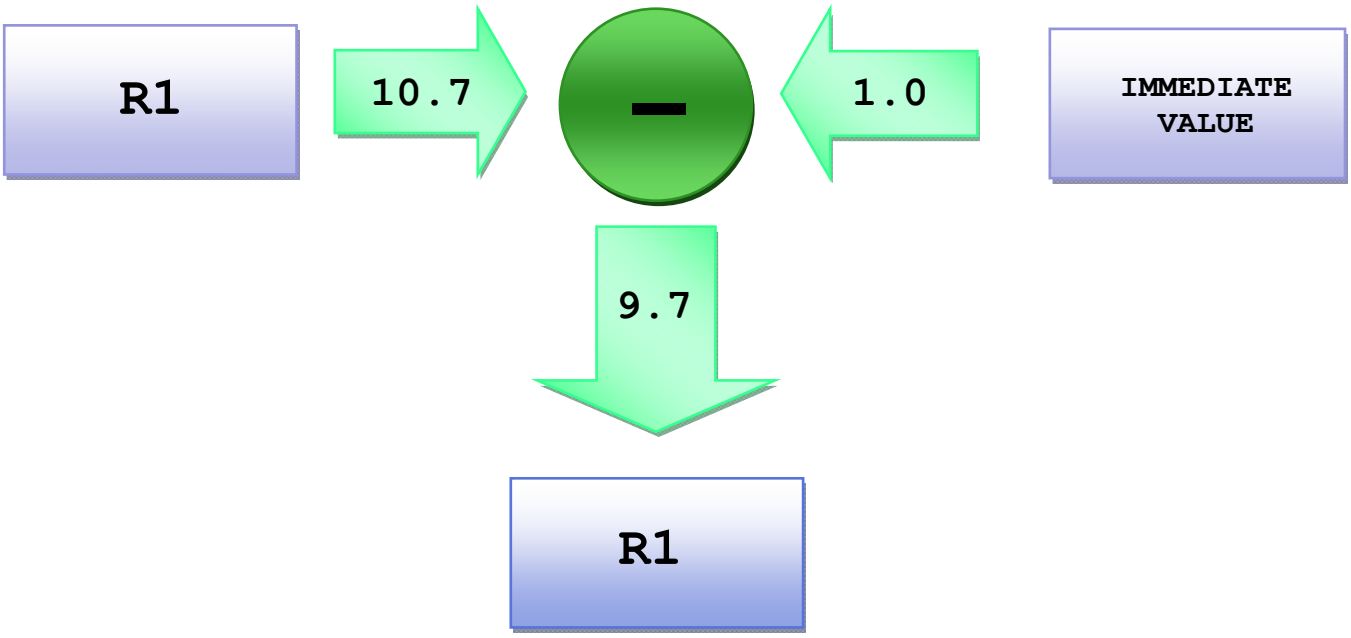
Real
R1 = 10.7

Instruction
FINC R1

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0101	000	00001	000	00000	000	00001

```
   10.7
 -  1.0
   9.7  →   R1
```



F_ZERO

Description:
Clears the floating point register.

Operation:
dest_reg ← 0x00000000

Instruction Format:
FZERO src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0110	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Contents

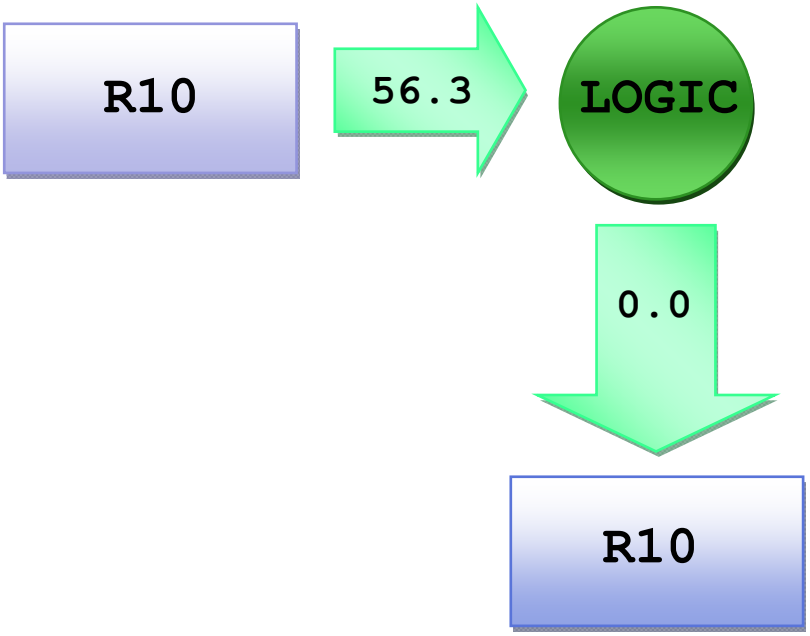
Real
R10 = 56.3

Instruction
FZERO R10

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0110	000	01010	000	00000	000	01010

0.0 → R10



F_ONE

Description:
Writes 1.0 into the floating point register.

Operation:
dest_reg ← 0xFFFFFFFF

Instruction Format:
ONE src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0111	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Contents

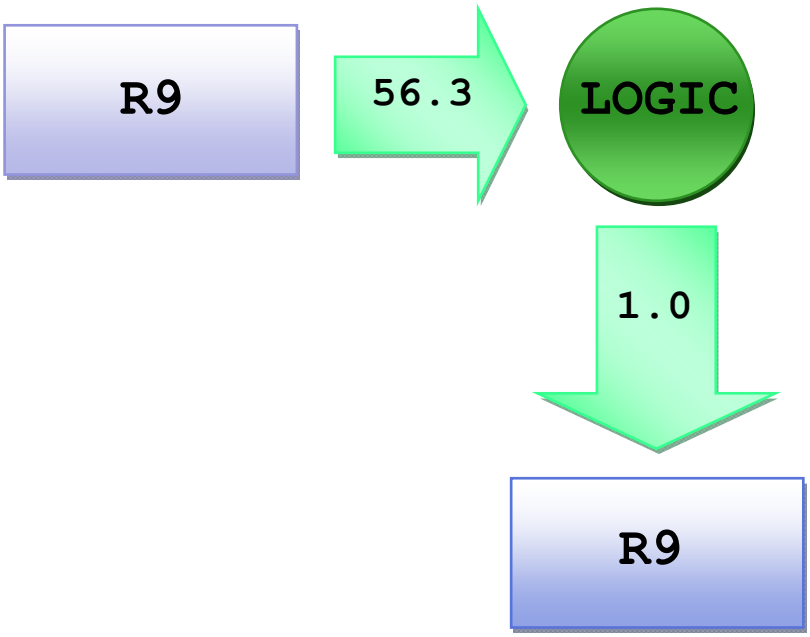
Real
R9 = 56.3

Instruction
FONE R9

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_0111	000	01001	000	00000	000	01001

1.0 → R10



F_LDI

Description:
Loads an immediate value a floating point register.

Operation:
dest_reg ← K

Instruction Format:
FLDI dest_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0110_1000	000	src1_reg	Immediate value	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Contents

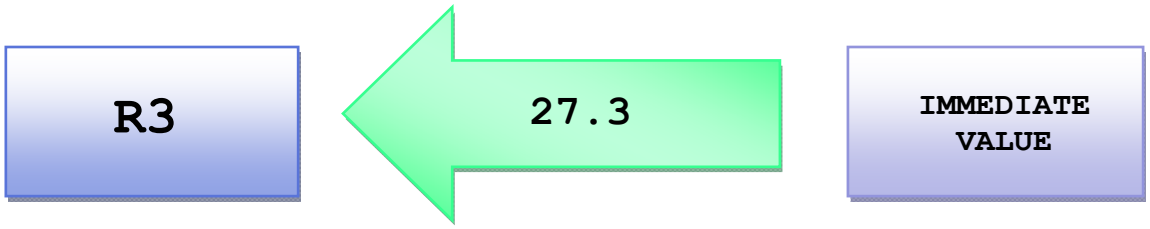
Real
R3 = 56.3

Instruction
FLDI R3, 27.3

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
0000_0111	000	00011	27.3	000	00011

27.3 → R10



F_LOAD

Description:
Loads the content of a memory location into the dest_reg.

Operation:
dest_reg ← [K]

Instruction Format:
FLD dest_reg ← [K]
32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_1001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

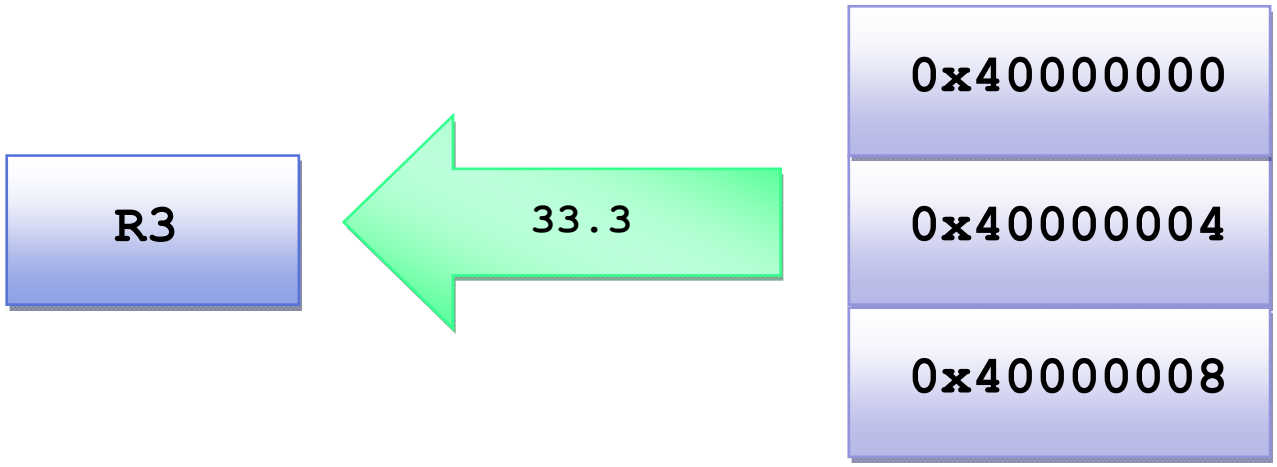
EXAMPLE

	Hex	Binary
	R5 = 0x40000004	0100_0000_0000_0000_0000_0000_0000_0100
Instruction	Memory Location	Contents
FLD R4, [R5]	0x40000000	27.0
	0x40000004	33.3
	0x40000008	05.2

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_1001	000	00101	000	33.3	000	00100

R4 ← 33.3



F_STORE

Description:
Stores the content of a floating point register into a memory location.

Operation:
[src2_reg] ← src1_reg

Instruction Format:
FST [src2_reg], src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_1010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
FST [R4], R2

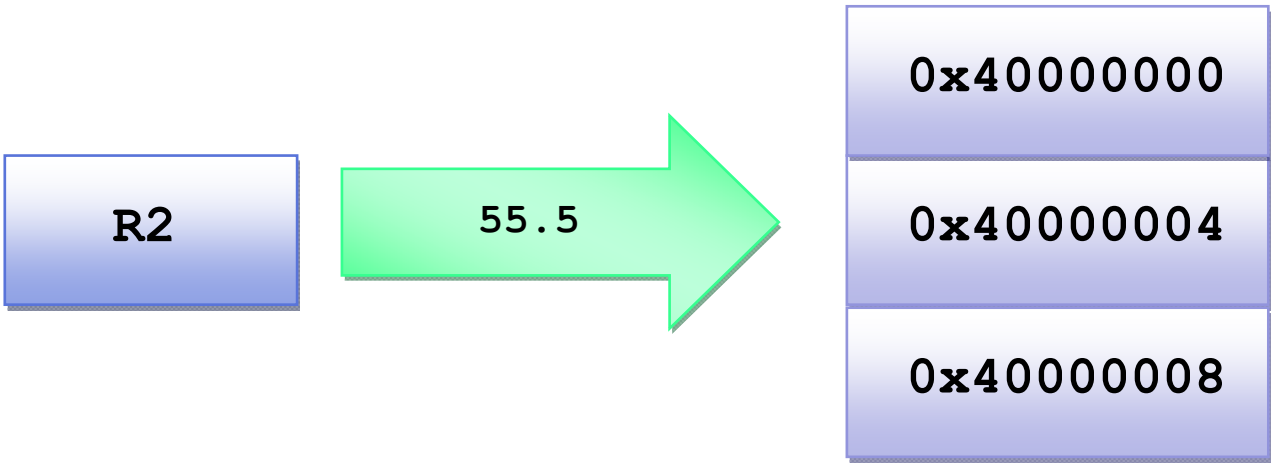
Real
R2 = 55.5
R4 = 22.0

Memory Location	Contents
0x40000000	11.1
0x40000004	25.87
0x40000008	88.1

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0110_1010	000	00010	000	00000	000	00100

0x40000004 ← 55.5



Enhancements

The BIU has been extended to include a Vector mux, which will allow the data input/output bus to be set to the 32 most significant bits or the 32 least significant bits of the Vector buffer depending on the input from the vector output enable. A vector load mux has been added in order to set the Vector buffer to either the 32 most significant, the 32 least significant bits, or to keep the current values in the buffer..

The integer datapath ALU has been extended to include the new Barrel shifter, Rotate shifter, and the swap instructions. The write clock also gets passed into the ALU so that the R register gets stored into the temporary register before any the first swap instruction is executed.

A Vector datapath module has been added to the Execution Unit to format the register data and perform the vector algorithms. The datapath first packages the 64 bit register data into 16 bit registers for both the R and S registers. Once the registers and flags have been packaged, the registers are paired up according to their respective bits and put into a vector ALU. The ALU then performs the specified algorithm on the vector registers and sets the flags. Each ALU outputs a Carry, Negative, Overflow, Zero, and Saturation flag that will be stored in the flags register and the algorithm results will be stored into their respective locations of the vector output register.

V_ADDS

Description:
Performs an arithmetic addition between the contents of two vector registers and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg + src2_reg

Instruction Format:
VADDS dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0000	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VADDS R0, R2, R5

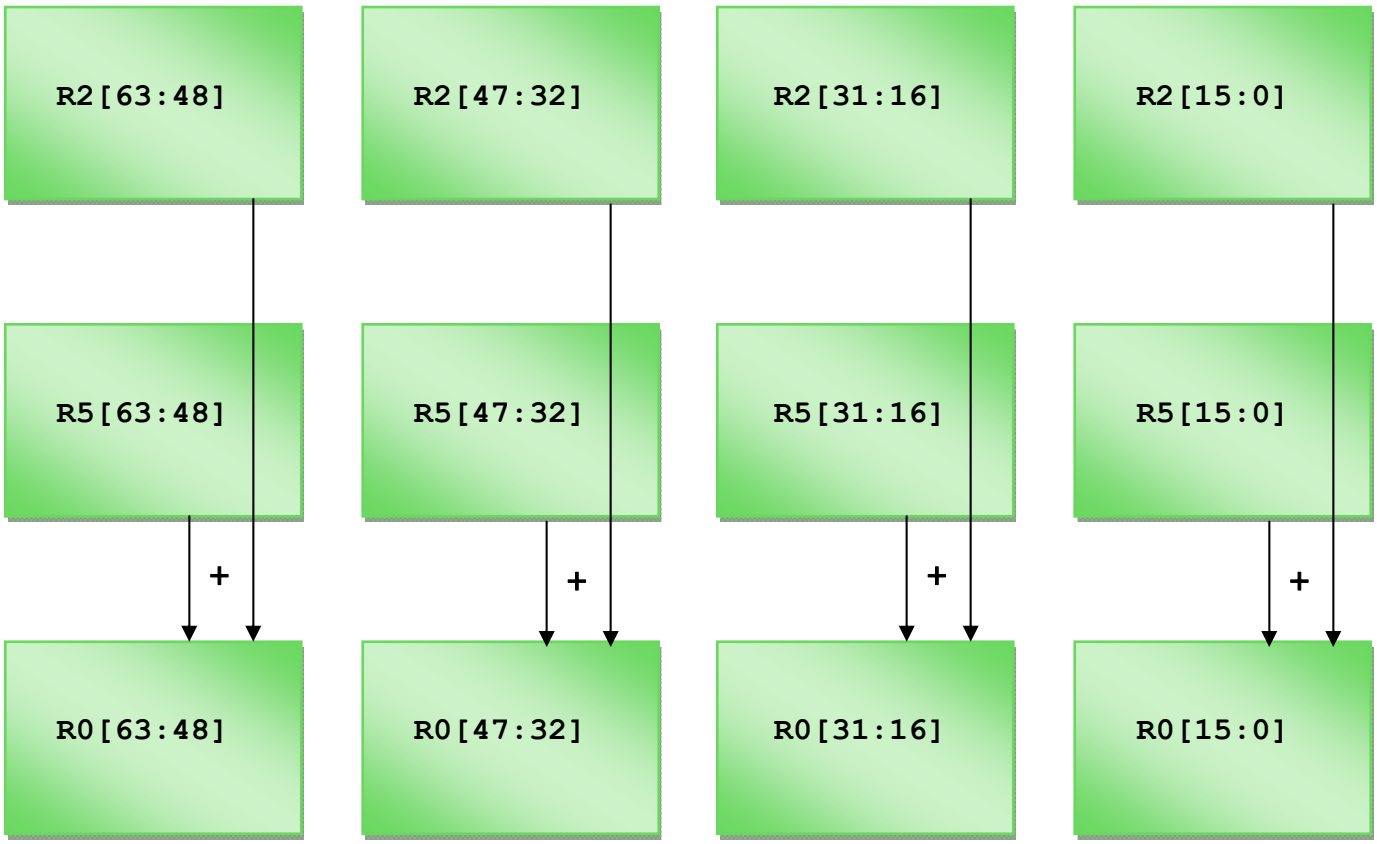
Contents
Hex
R2 = 005F_0010_0005_000Eh
R5 = 0013_0003_0003_0001h

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0000	000	00010	000	00101	000	00000

```
   005F        0010        0005        000E
 + 0013      + 0003      + 0003      + 0001
   0072        0013        0008        000F
```

R0 ← 0072_0013_0008_000F



V_SUBS

Description:
Performs an arithmetic subtration between the contents of two vector registers and places the result into the dest_reg. The result cannot exceed +32767 and -32767.

Operation:
dest_reg ← src1_reg - src2_reg

Instruction Format:
VSUBS dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VSUBS R0, R2, R5

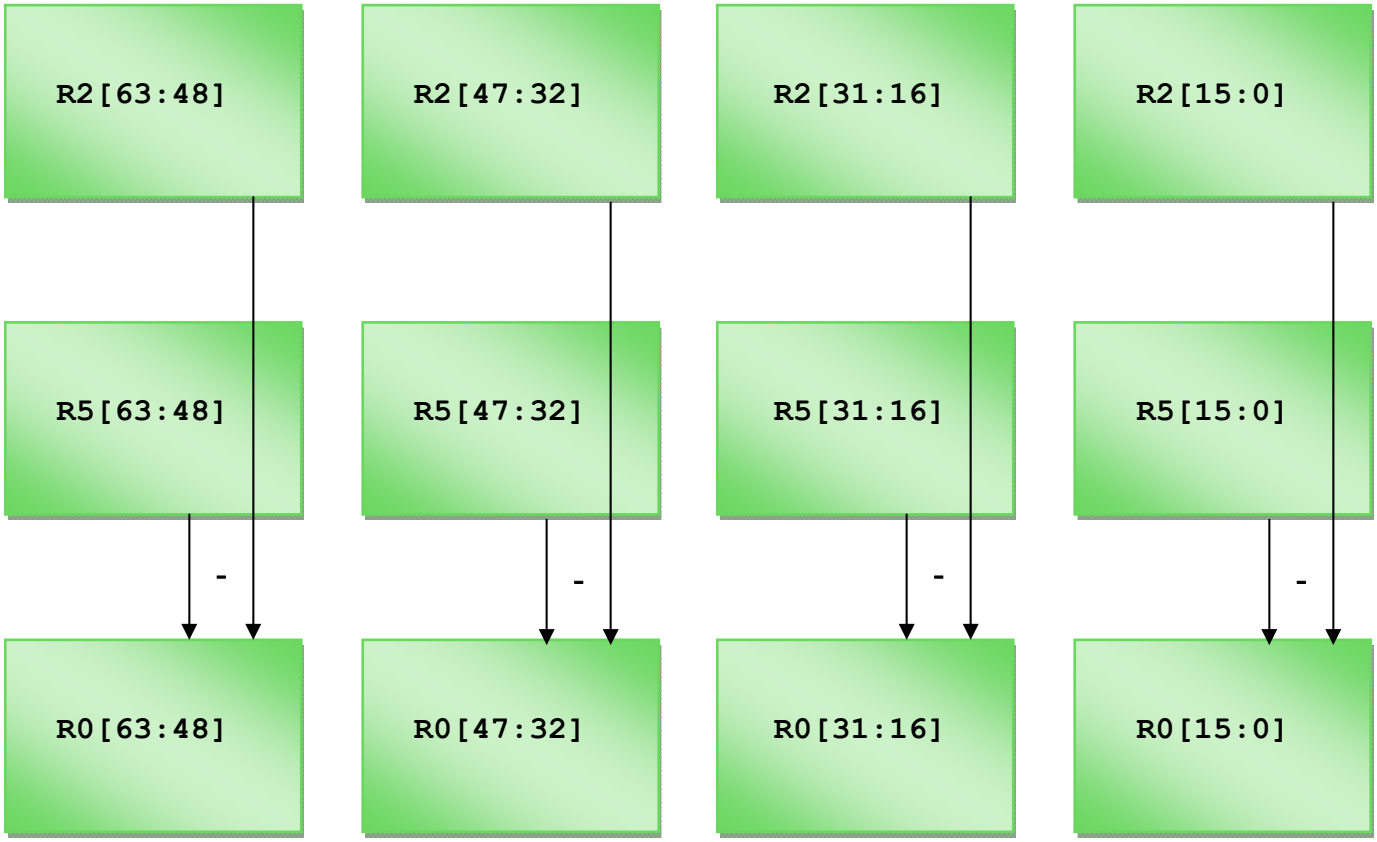
Contents
Hex
R2 = 005F_0010_0005_000Eh
R5 = 0013_0003_0003_0001h

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0001	000	00010	000	00101	000	00000

```
   005F        0010        0005        000E
 - 0013      - 0003      - 0003      - 0001
   00FB        0007        0002        000D
```

R0 ← 00FB_0007_0002_000D



V_SUBW

Description:
Performs an arithmetic subtraction between the contents of two vector registers and places the result into the dest_reg. The result may exceed +32767 and -32767.

Operation:
dest_reg ← src1_reg - src2_reg

Instruction Format:
VSUBW dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VSUBS R0, R2, R5

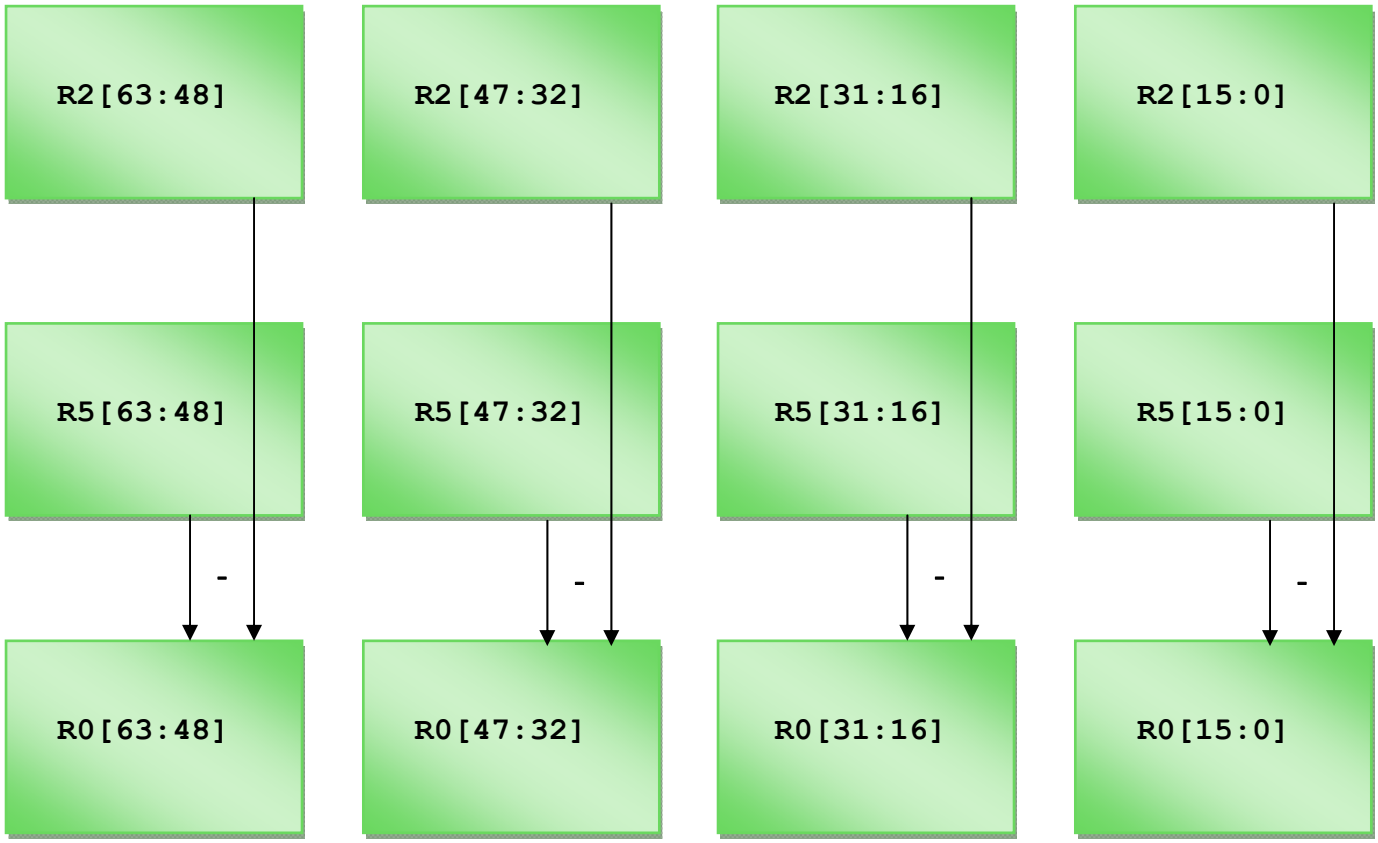
Contents
Hex
R2 = FFFF_FFFF_FFFF_FFFFh
R5 = 0000_0000_0000_0002h

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0010	000	00010	000	00101	000	00000

```
   FFFF        FFFF        FFFF        FFFF
 -  0000     -  0000     - 0000      - 0002
 1  0000        0000        0000        0001
```

R0 ← 0000_0000_0000_0000



V_LSHL

Description:
Performs a logical shift left on the contents of src1_reg and places the result into the dest_reg.

Operation:

Instruction Format:
VLSHL src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0011	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

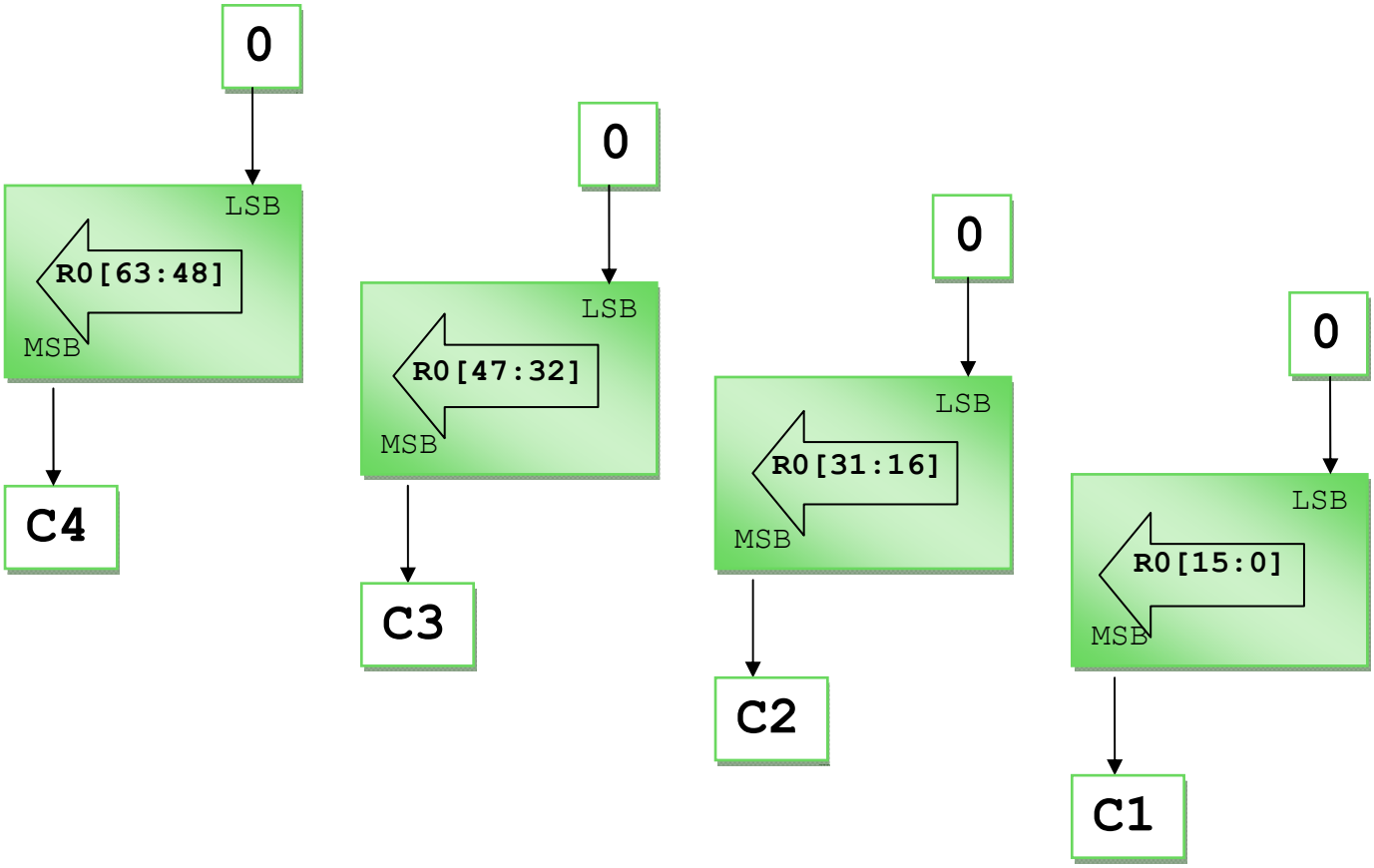
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VLSL R0

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0011	000	00000	000	00000	000	00000



V_LSHR

Description:
Performs a logical shift right on the content of src1_reg and places the result into the dest_reg.

Operation:

Instruction Format:
VLSHR src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0100	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

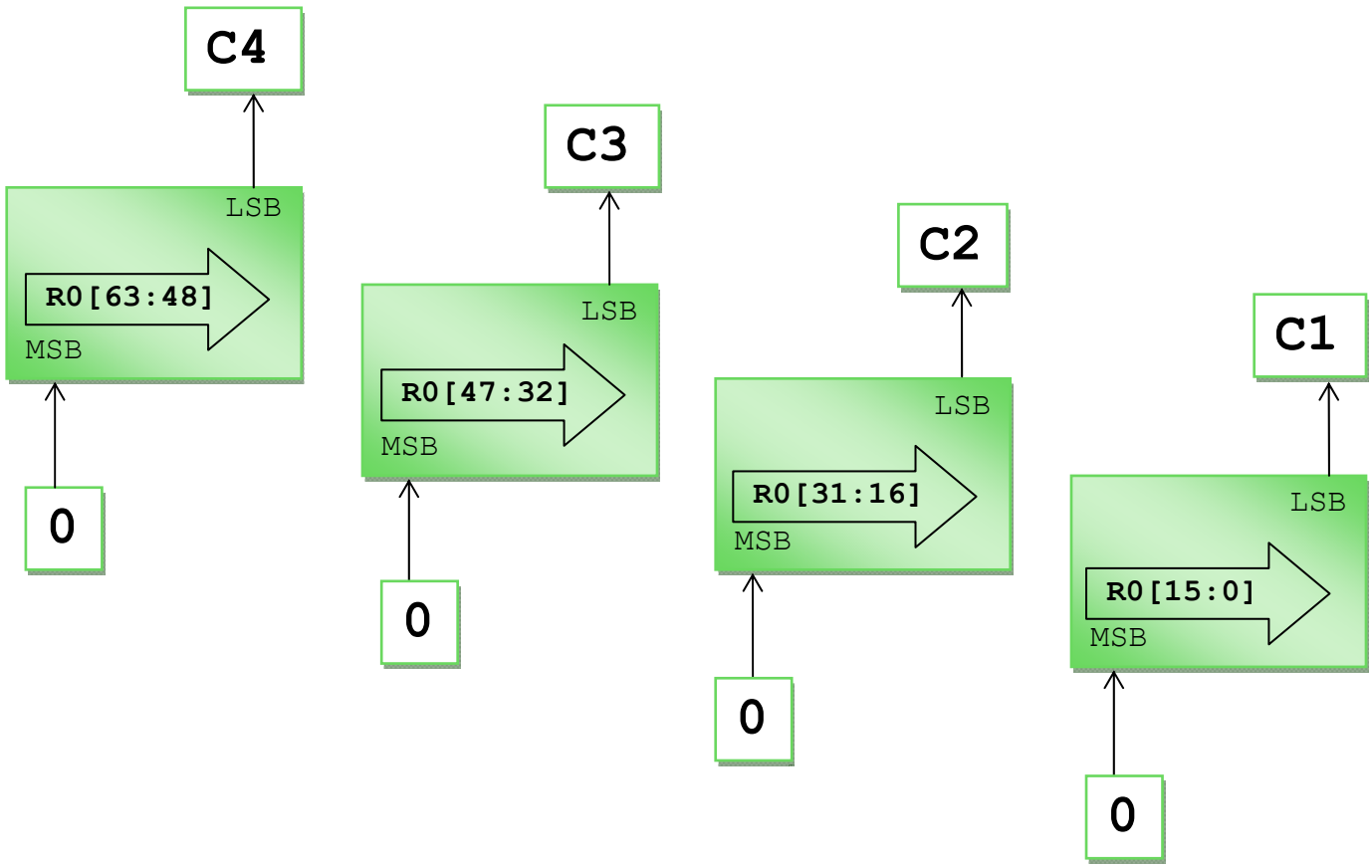
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VLSR R0

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0100	000	00000	000	00000	000	00000



V_AND

Description:
Performs a logical AND between the contents of two vector registers and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg & src2_reg

Instruction Format:
VAND dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0101	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

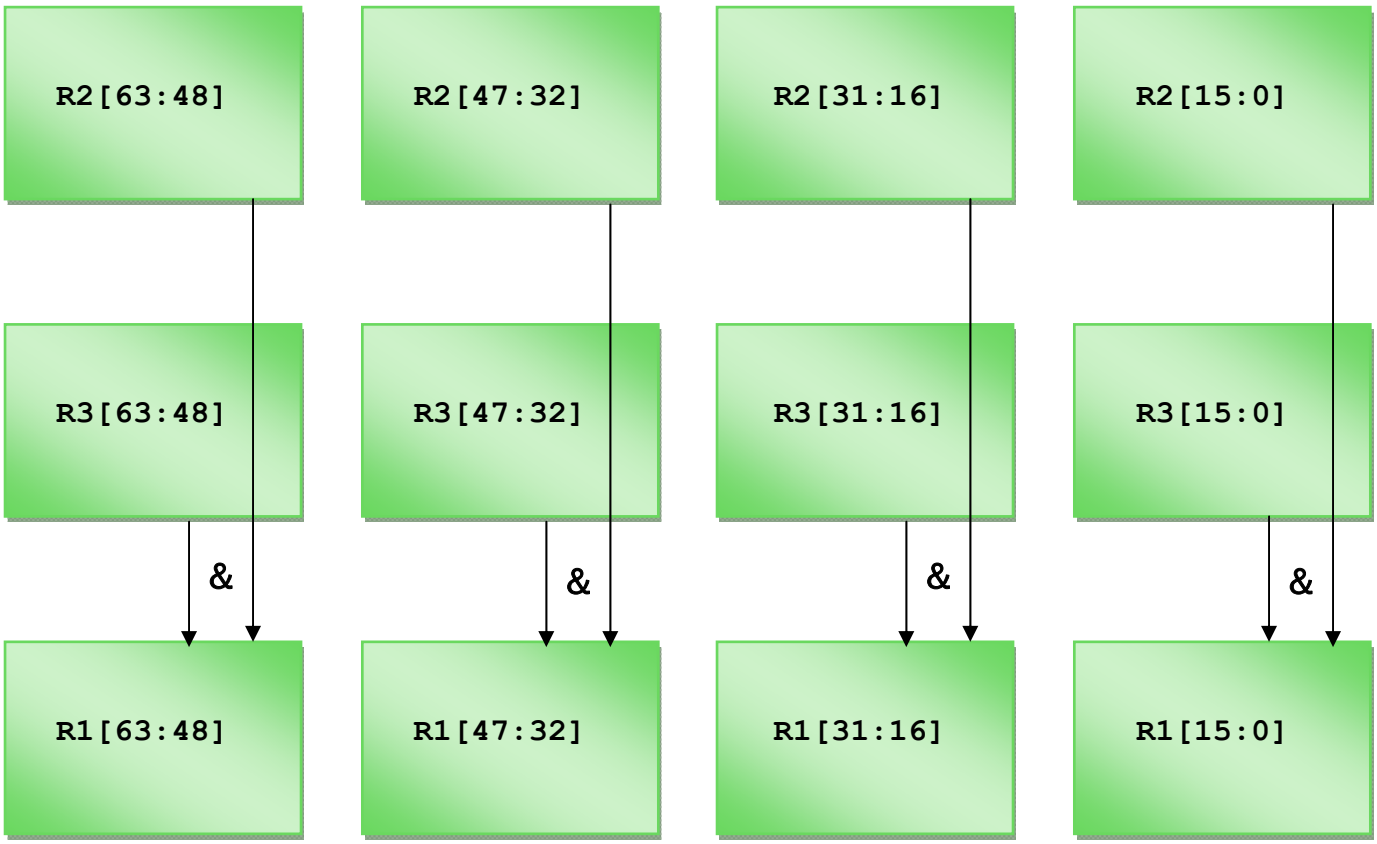
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VAND R1, R2, R3

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0101	000	00010	000	00011	000	00001



V_OR

Description:
Performs a logical OR between the contents of two vector registers and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg | src2_reg

Instruction Format:
VOR dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0110	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

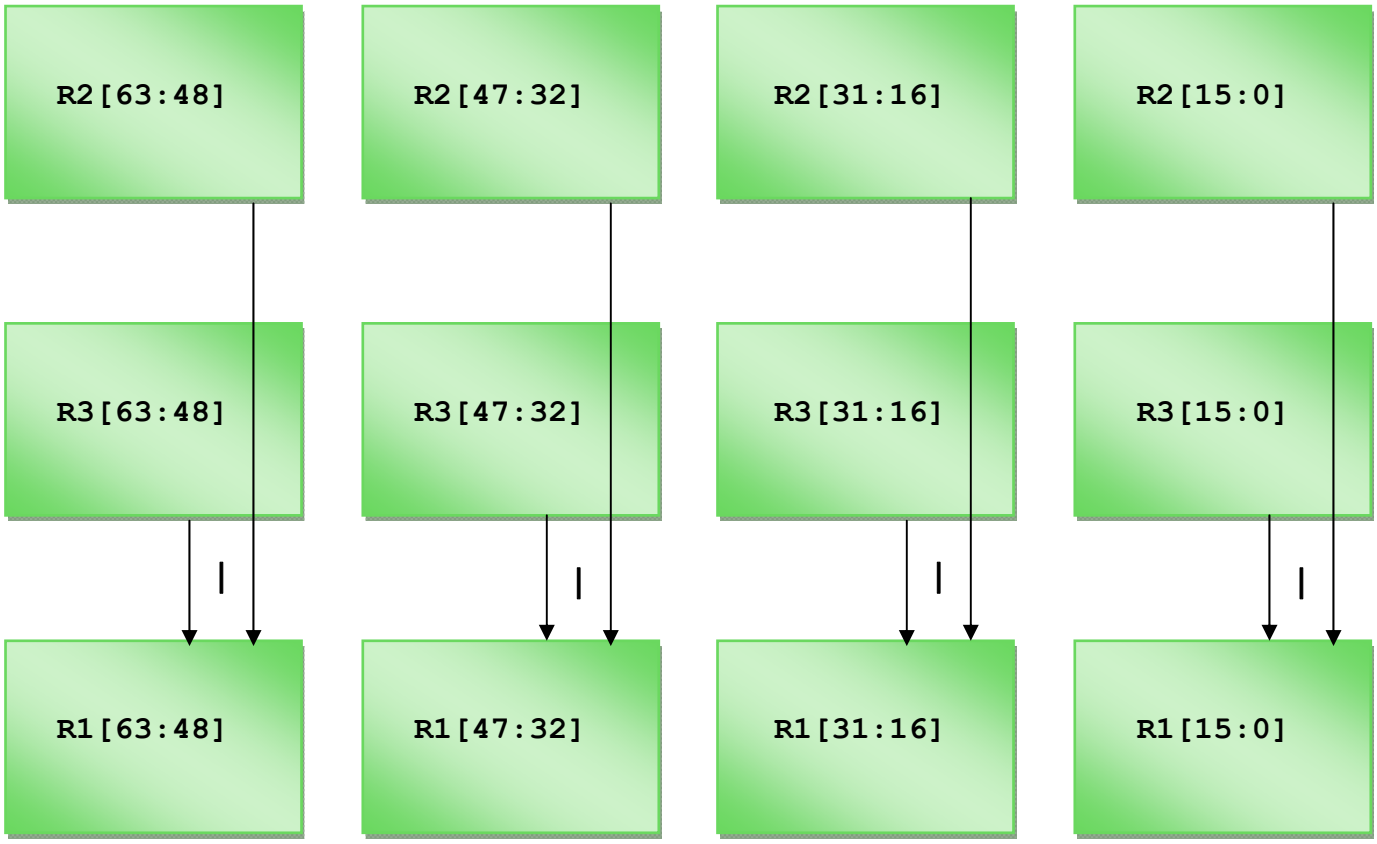
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VOR R1, R2, R3

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0110	000	00010	000	00011	000	00001



V_XOR

Description:
Performs a logical XOR between the contents of two vector registers and places the result into the dest_reg.

Operation:
dest_reg ← src1_reg ^ src2_reg

Instruction Format:
VXOR dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0111	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

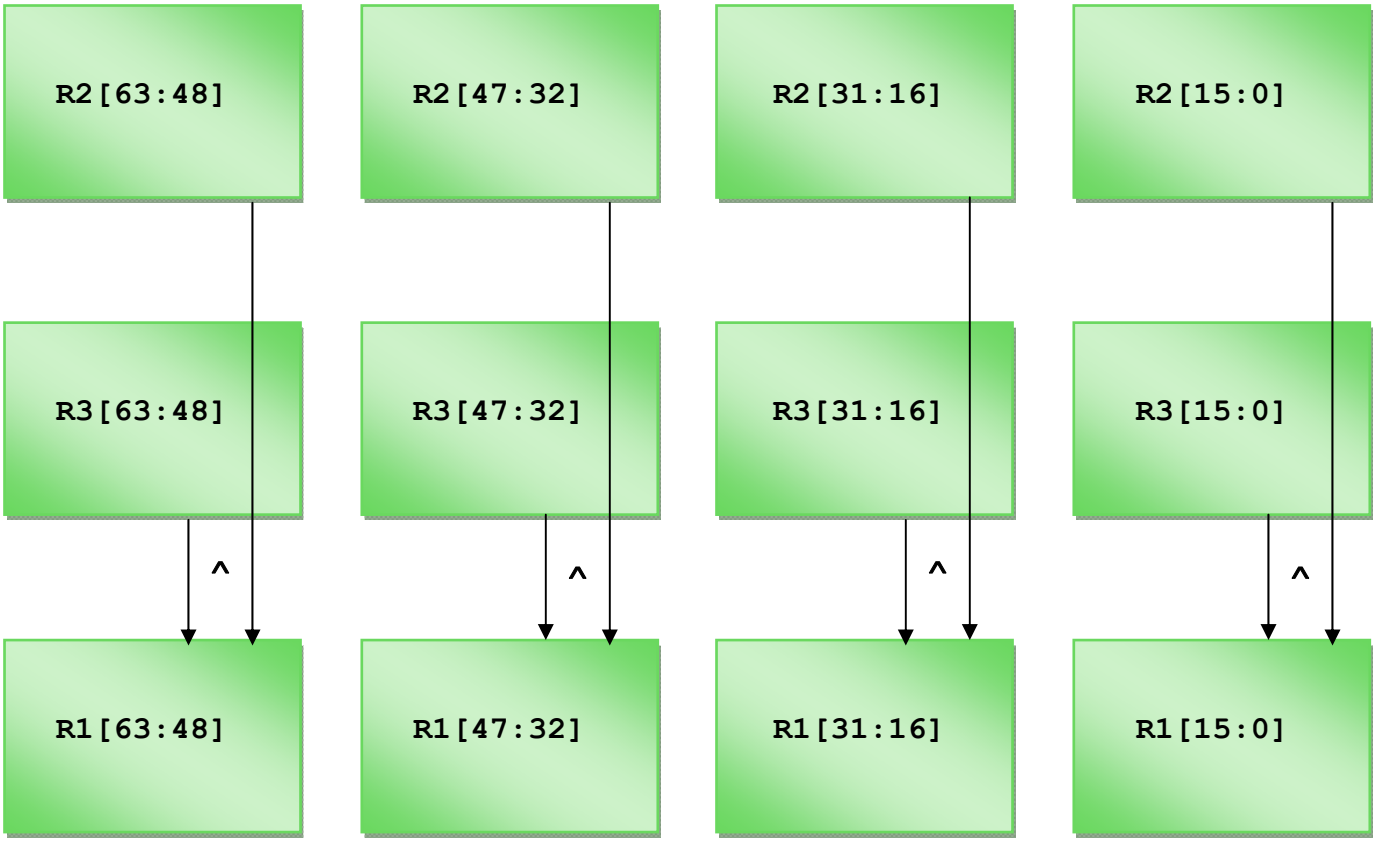
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VXOR R1, R2, R3

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_0111	000	00010	000	00011	000	00001



V_ANDN

Description:
Performs a logical NAND between the contents of two vector registers and places the result into the dest_reg.

Operation:
dest_reg ← ~(src1_reg & src2_reg)

Instruction Format:
VANDN dest_reg, src1_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1000	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	-	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

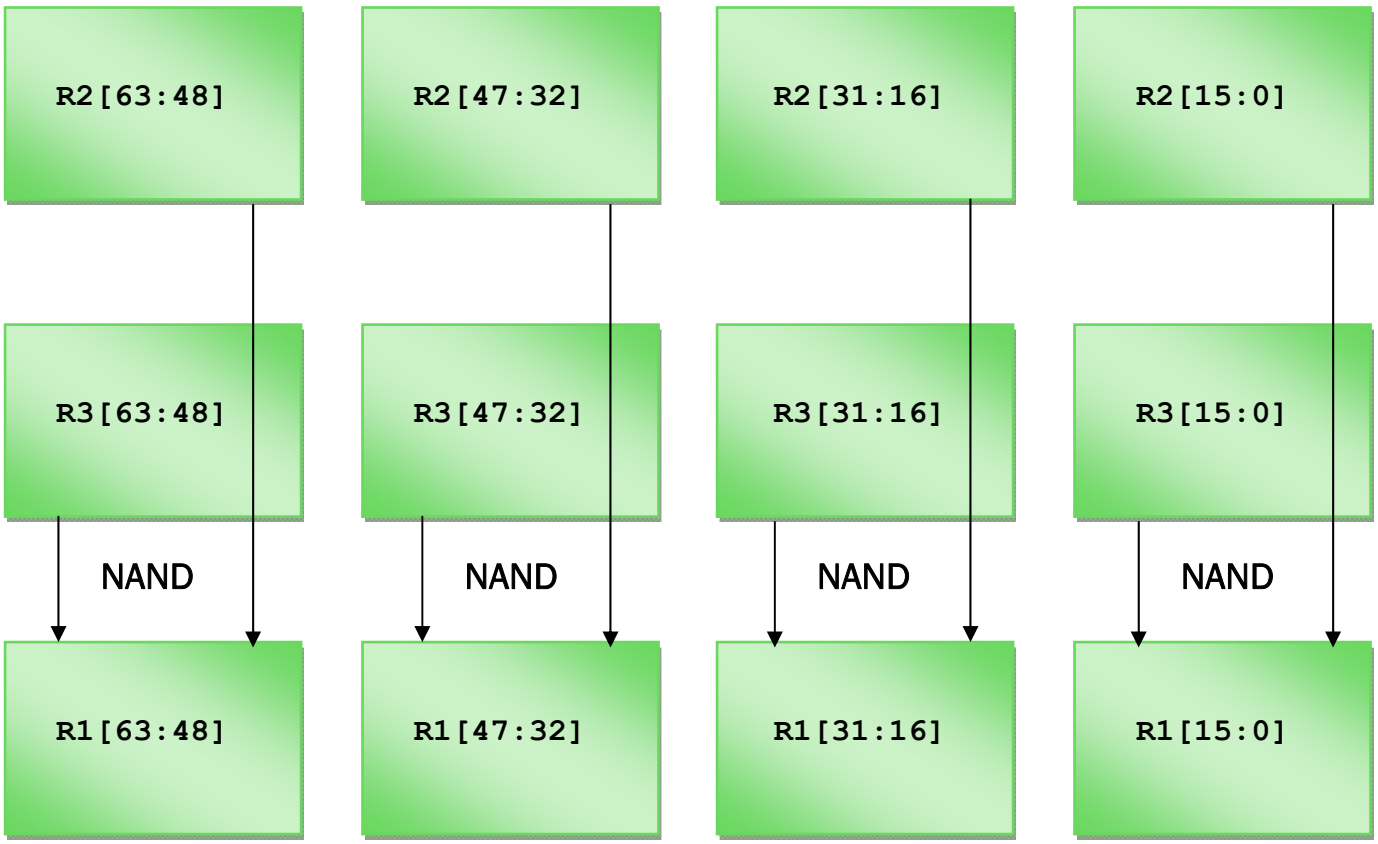
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VANDN R1, R2, R3

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1000	000	00010	000	00011	000	00001



V_NOT

Description:
Performs a logical NOT on the contents src1_reg and places the result into the dest_reg.

Operation:
dest_reg ← ~src1_reg

Instruction Format:
VNOT src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	⇔	⇔	⇔	⇔

N: Set if MSB of the result is set; cleared otherwise.

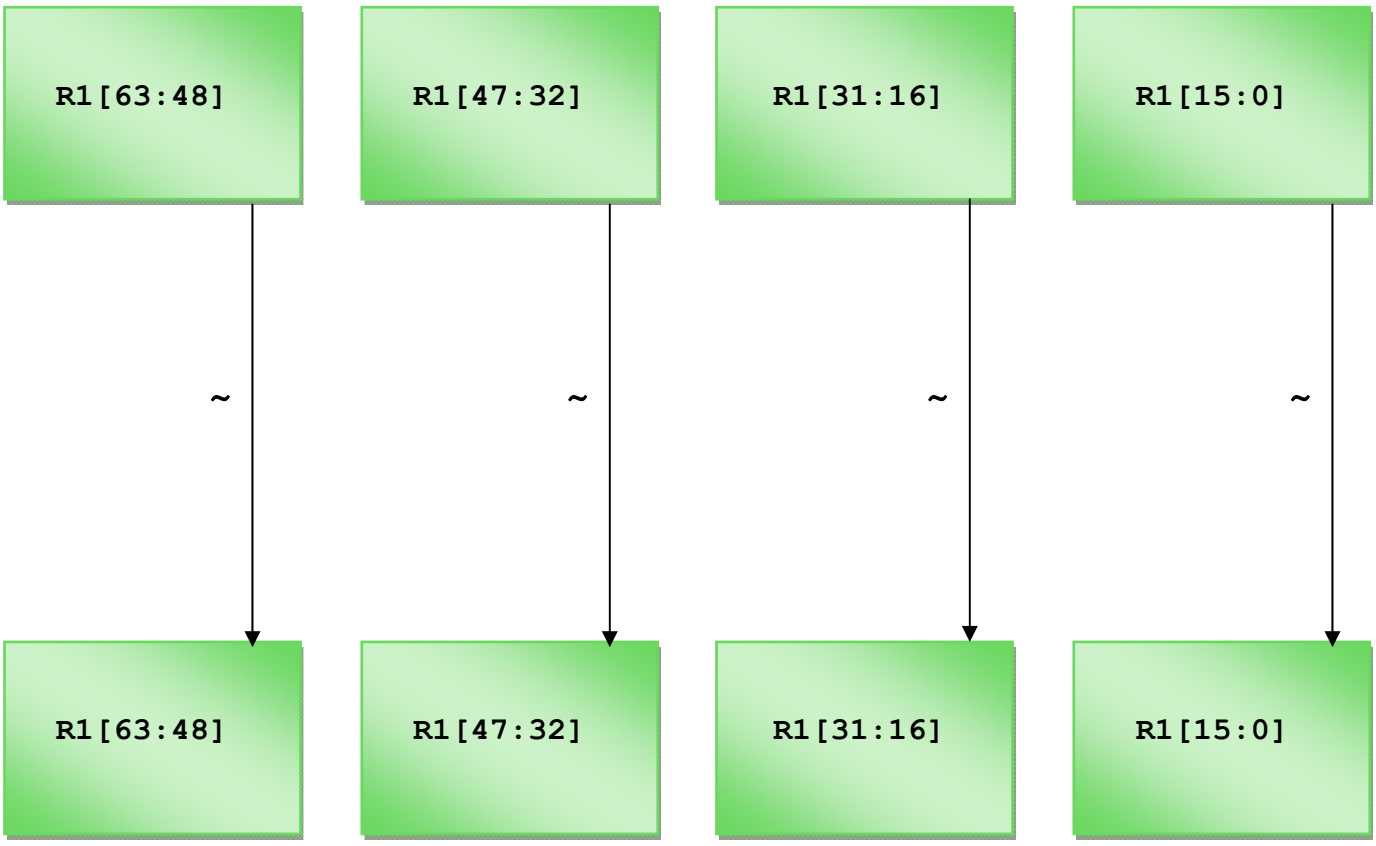
P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VNOT R1
Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1001	000	00001	000	00000	000	00001



V_NEGATE

Description:
Performs a logical NOT on the content of src1_reg and then adds 1 to it and places the result into the dest_reg.

Operation:
dest_reg ← ~src1_reg + 1

Instruction Format:
VNEG dest_reg, src1_reg, src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

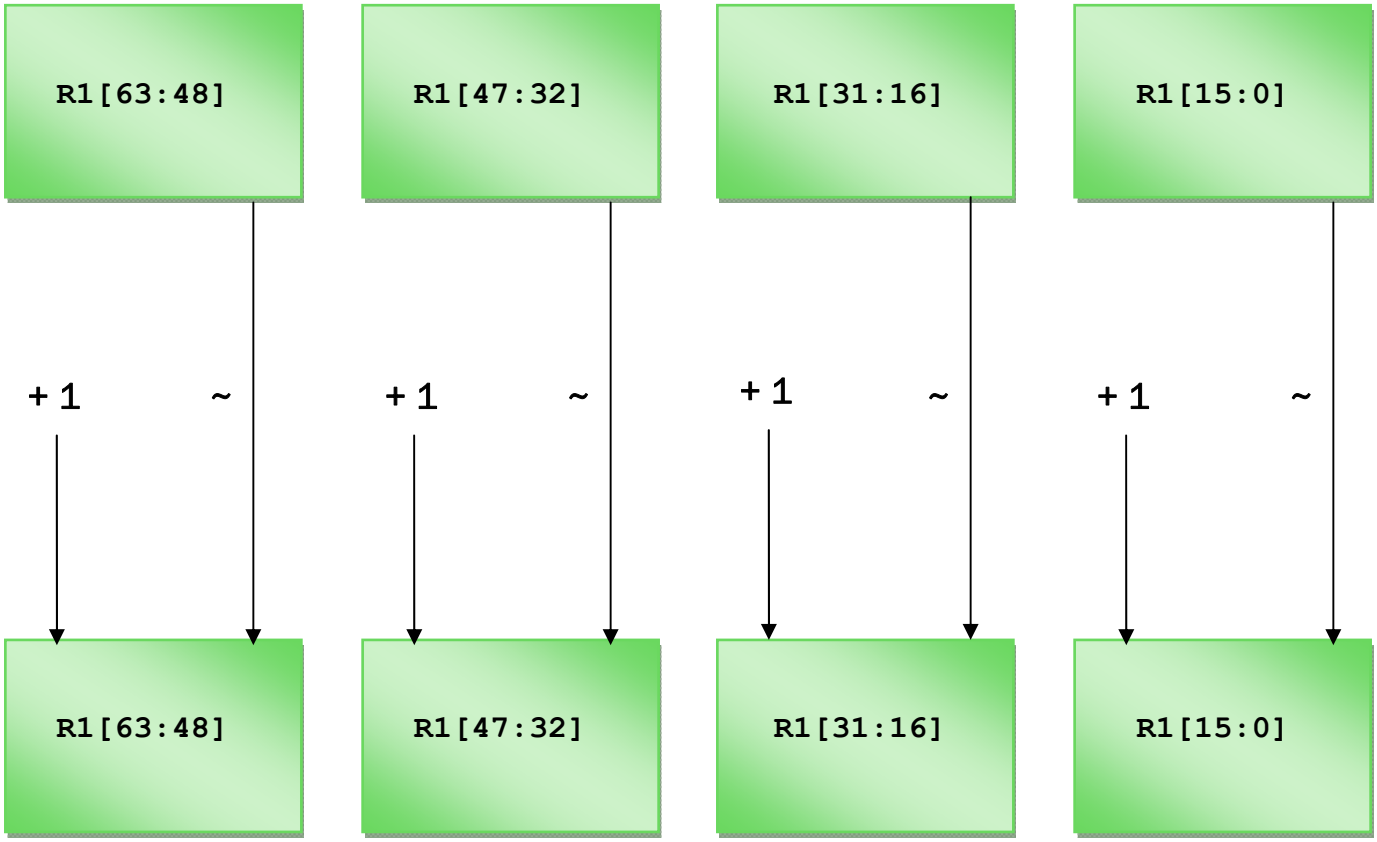
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VNEG R1

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1010	000	00001	000	00000	000	00001



V_PASSR

Description:
Places the content of the src1_reg into the dest_reg.

Operation:
dest_reg ← src1_reg

Instruction Format:
VPASSR src1_reg, dest_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1011	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

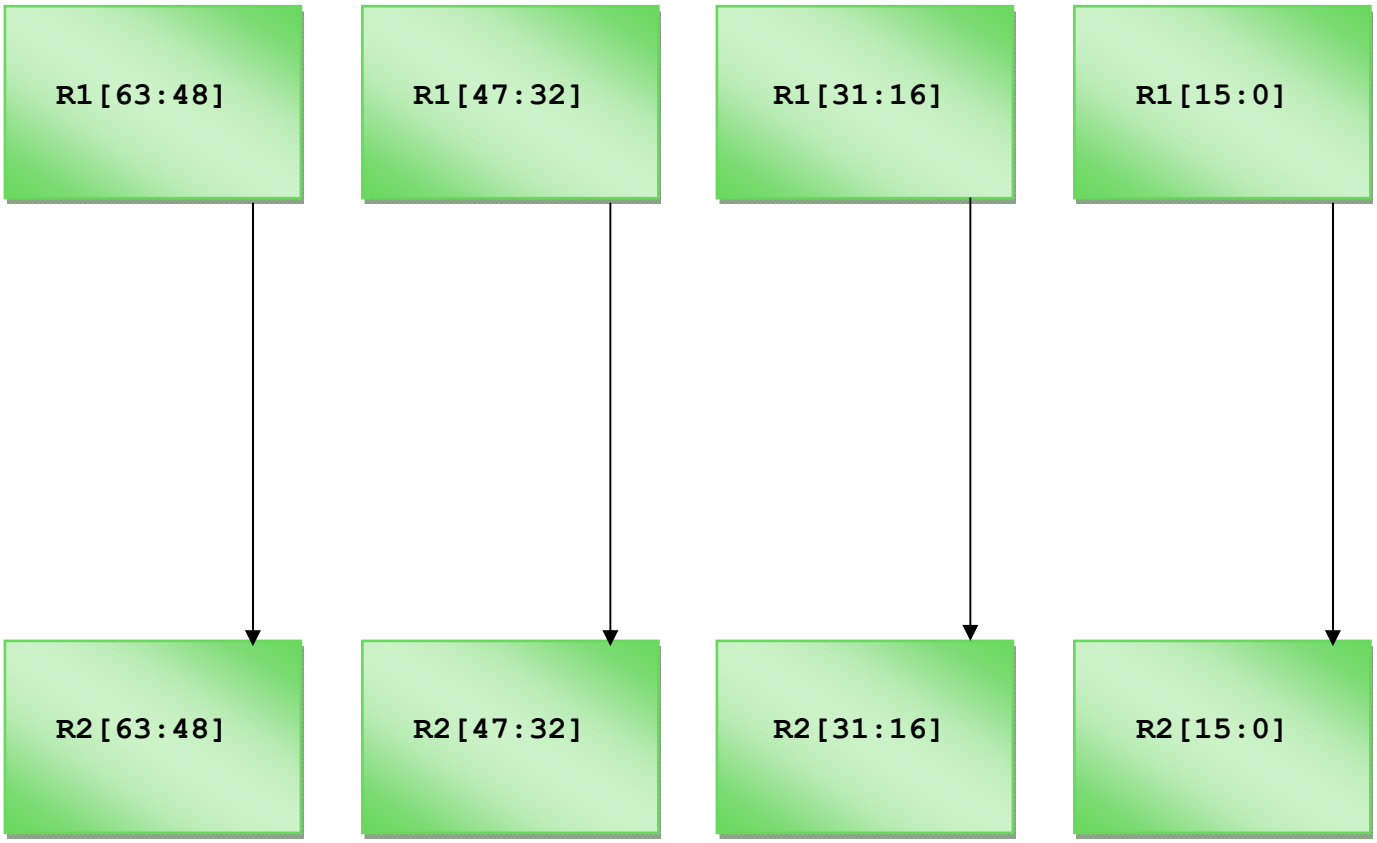
C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Instruction
VPASSR R2, R1

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1011	000	00001	000	00000	000	00010



V_PASSS

Description:
Places the content of the src2_reg into the dest_reg.

Operation:
dest_reg ← src2_reg

Instruction Format:
VPASSS src2_reg, src2_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1100	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

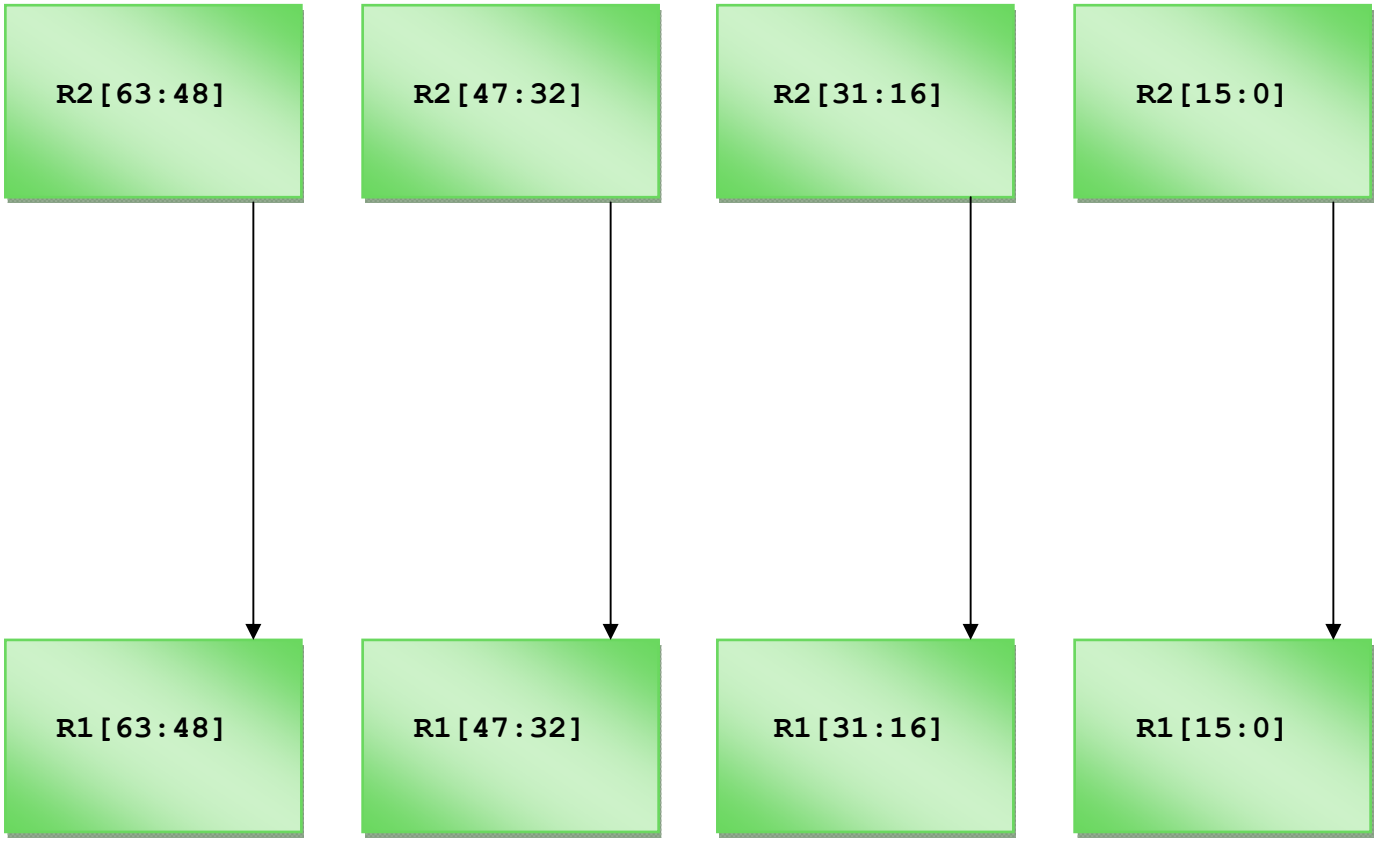
C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Instruction
VPASSS R1, R2

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1100	000	00010	000	00000	000	00001



V_COPY

Description:
Copy the contents of the src1_reg and places it into the desst_reg.

Operation:
dest_reg ← src1_reg

Instruction Format:
VCPY dest_reg, src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1101	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

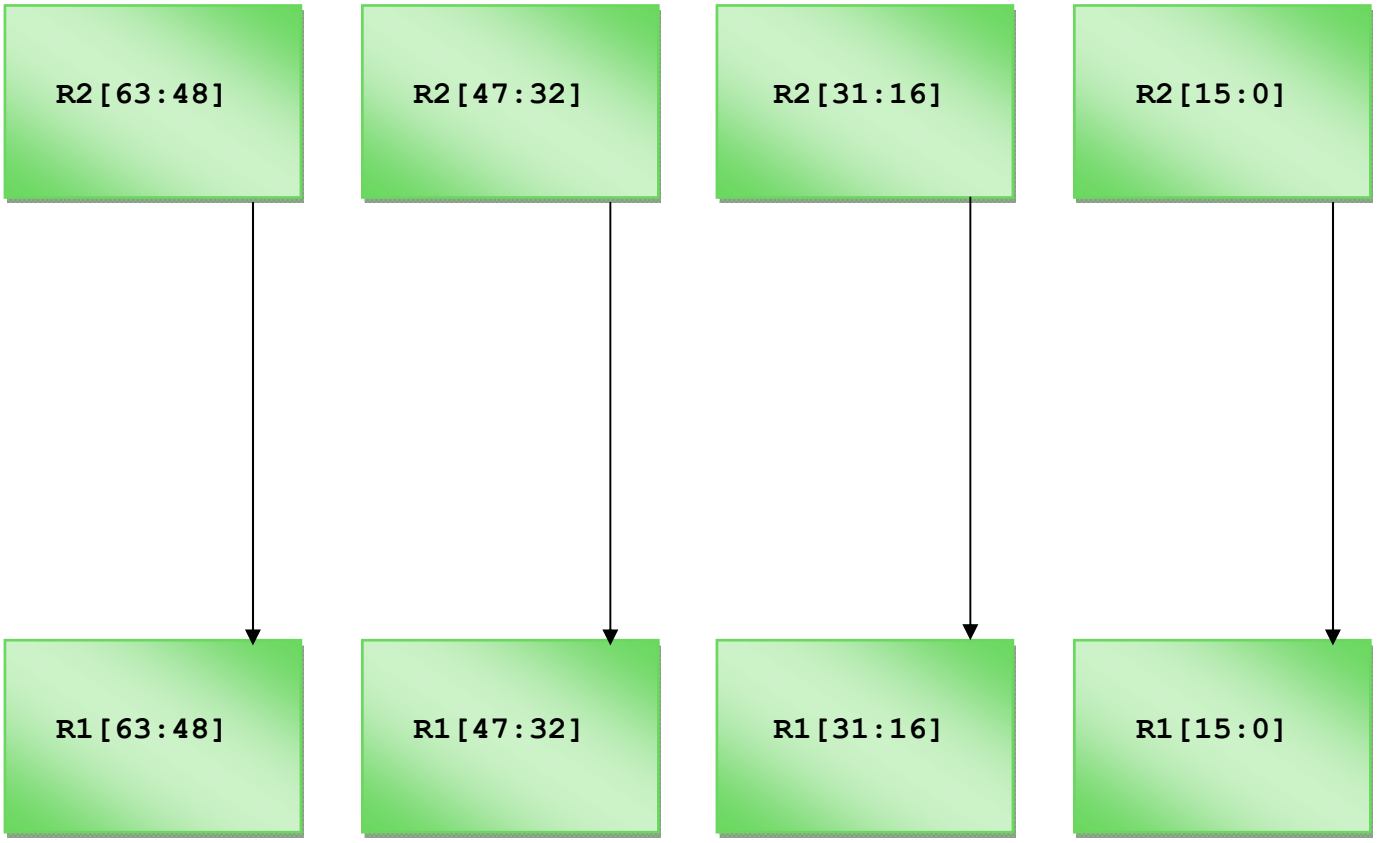
C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Instruction
VCPY R1, R2

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1101	000	00010	000	00000	000	00001



V_COMPARE

Description:
This instruction performs a compare between the contents of two vector registers src2_reg and src1_reg. Subtracts src1_reg from src2_reg and compares the result to zero, but does not store the result. This instruction updates the Program Status Register to use for jump statements.

Operation:
src1_reg – src2_reg == 0

Instruction Format:
VCMP src2_reg, src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1110	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of MSB of the operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

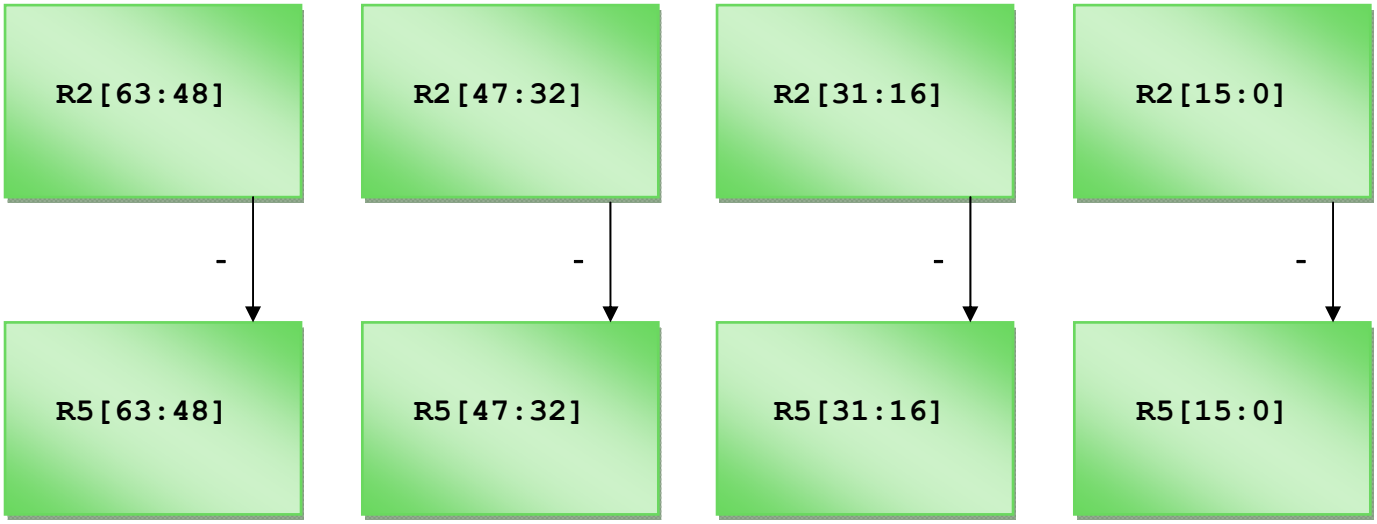
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VCMP R2, R5

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1110	000	00010	000	00101	000	00000



V_GT

Description:
Performs an subtraction addition between the contents of two vector registers and places the result into the dest_reg. Updates the flags register.

Operation:
dest_reg ← src1_reg - src2_reg

Instruction Format:
VGT dest_reg, src1_reg, src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1111	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
⇔	⇔	⇔	⇔	⇔

C: Set if there was carry from MSB of the result; cleared otherwise.

N: Set if MSB of the result is set; cleared otherwise.

O: Set if MSB of the result is opposite of both MSB of operand; cleared otherwise.

P: Set if number of set bits in result is equivalent to numbers of unset bits; cleared otherwise.

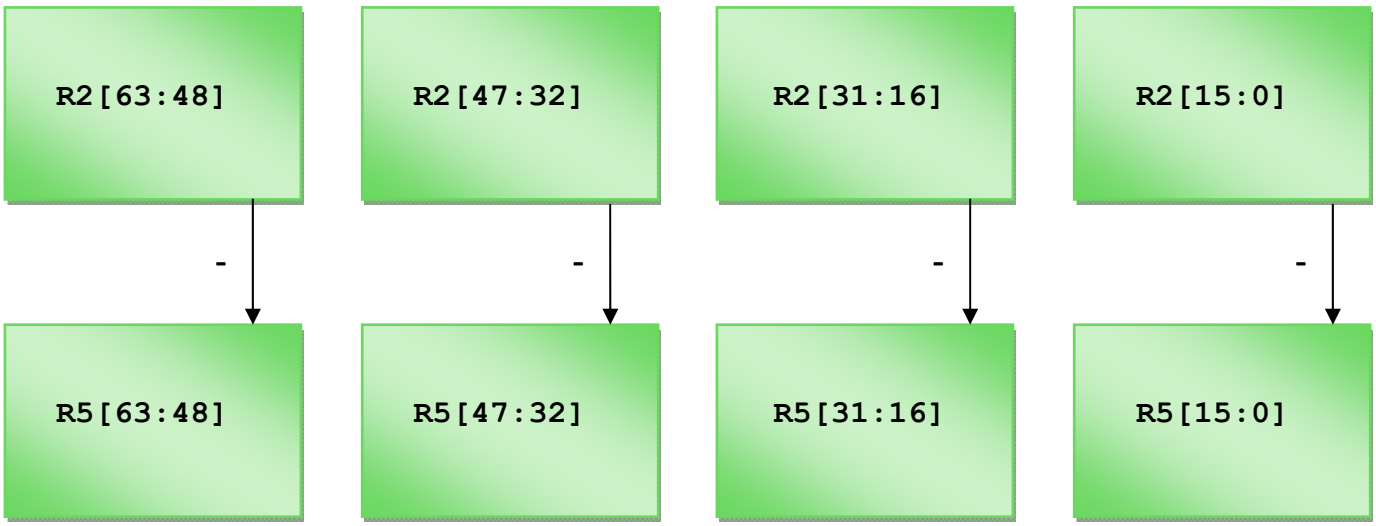
Z: Set if result is 0; cleared otherwise.

EXAMPLE

Instruction
VGT R2, R5

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
0100_1111	000	00010	000	00000	000	00001



V_SWAP

Description:
Performs a swap between the contents of two vector registers.

Operation:
src1_reg ← dest_reg, dest_reg ← src1_reg

Instruction Format:
VSWAP src1_reg, dest_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0000	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

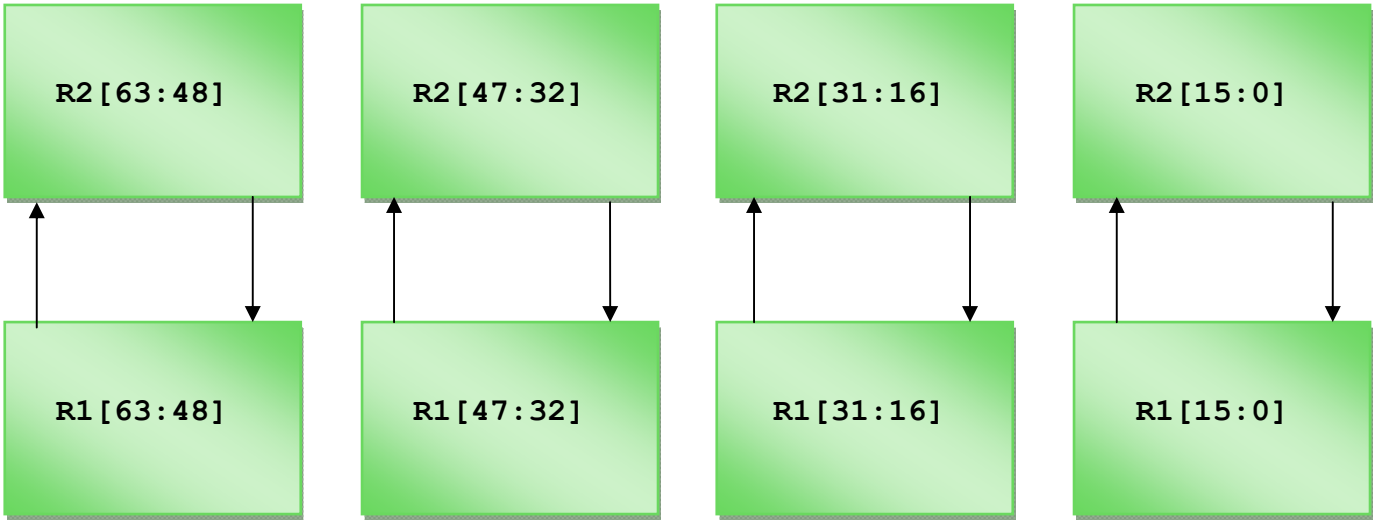
C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Instruction
VSWAP R1, R2

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0000	000	00010	000	00000	000	00001



V_LOAD IMMEDIATE

Description:
Places the content of an immediate value into the dest_reg.

Operation:
dest_reg ← K

Instruction Format:
VLDI dest_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0001	000	src1_reg	Immediate value	000	dest_reg

Status Register:

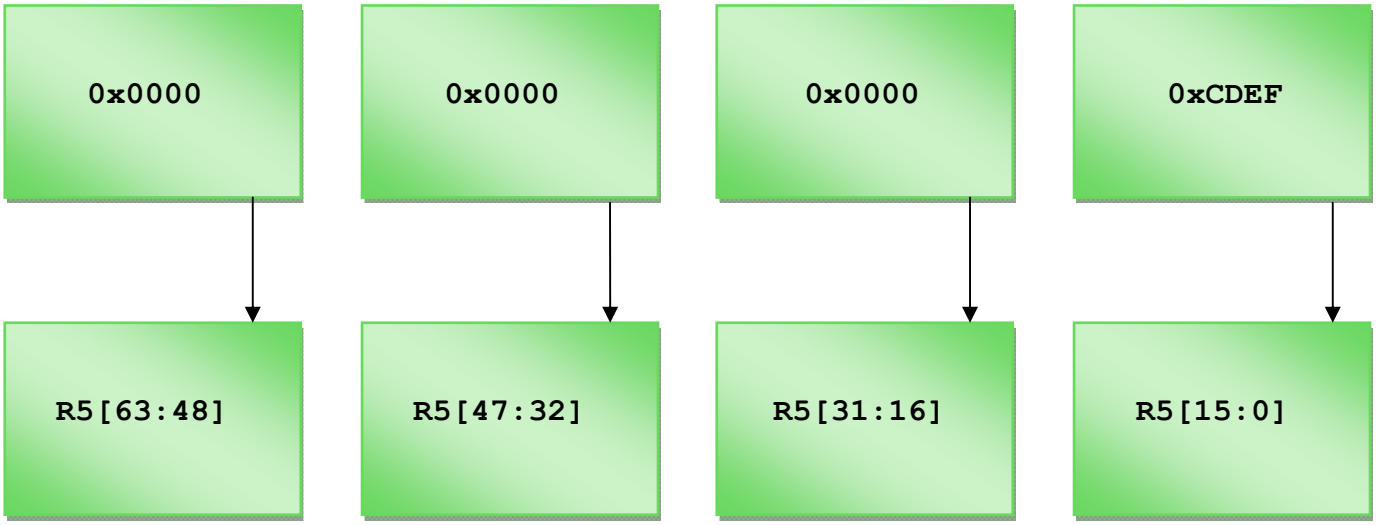
C	N	O	P	Z
-	-	-	-	-

EXAMPLE

Instruction
VLDI R1, 0x0000_0000_0000_CDEF

Binary Instruction Format

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0001	000	00000	CDEF	000	00001



V_STORE_PC

Description:
Places the content of the Program Counter into a memory location.

Operation:
MAR[src1_reg] ← PC

Instruction Format:
VSTPC [src1_reg]

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0010	000	src1_reg	000	src2_reg	000	dest_reg

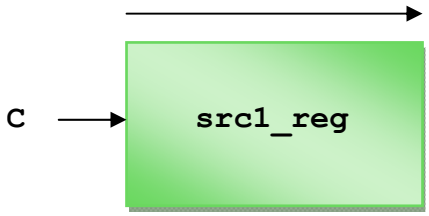
Status Register:

C	N	O	P	Z
-	-	-	-	-

SHIFT RIGHT W/ CARRY

Description:
Performs a logical shift right and shifts in the content of the Carry Flag into the MSB.

Operation:



Instruction Format:
SRC src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0011	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

SHIFT LEFT W/ CARRY

Description:
Performs a logical shift left and shifts in the content of the Carry Flag into the LSB.

Operation:

Instruction Format:
SLC src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0100	000	src1_reg	000	src2_reg	000	dest_reg

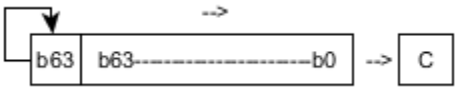
Status Register:

C	N	O	P	Z
-	-	-	-	-

ARITHMETIC ROTATE RIGHT

Description:
Shifts all bits of register src1_reg one place to the right. Copies the content in the MSB before the shift and places it in the MSB after the shift. Bit 0 is loaded into the C Flag of the Status Register.

Operation:



Instruction Format:
ARR src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0101	000	src1_reg	000	src2_reg	000	dest_reg

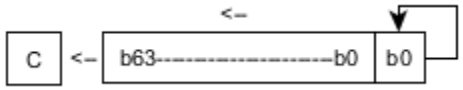
Status Register:

C	N	O	P	Z
-	-	-	-	-

ARITHMETIC ROTATE LEFT

Description:
Shifts all bits of register src1_reg one place to the left. Copies the content in the LSB before the shift and places it in the LSB after the shift. Bit 63 is loaded into the C Flag of the Status Register.

Operation:



Instruction Format:
ARR src1_reg

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0110	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

BARREL ROTATE RIGHT

Description:
Performs a rotate right K times on the content of src1_reg where K is a constant value and places the result into the dest_reg.

Operation:

Instruction Format:
BRR src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_0111	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

BARREL ROTATE LEFT

Description:
Performs a rotate left K times on the content of src1_reg where K is a constant value and places the result into the dest_reg.

Operation:

Instruction Format:
BRL src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_1000	000	src1_reg	000	src2_reg	000	dest_reg

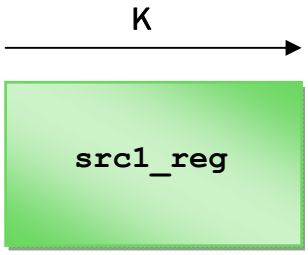
Status Register:

C	N	O	P	Z
-	-	-	-	-

BARREL LOGICAL SHIFT RIGHT

Description:
Performs a logical shift right K times on the content of src1_reg where K is a constant value and places the result into the dest_reg.

Operation:



Instruction Format:
BLSR src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_1001	000	src1_reg	000	src2_reg	000	dest_reg

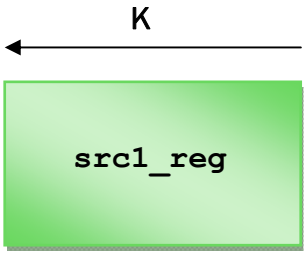
Status Register:

C	N	O	P	Z
-	-	-	-	-

BARREL SHIFT LEFT

Description:
Performs a logical shift left K times on the content of src1_reg where K is a constant value and places the result into the dest_reg.

Operation:



Instruction Format:
BLSL src1_reg, K

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_1010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

JUMP IF EQUAL

Description:
Execute a jump instruction if the Zero Flag is 1.

Operation:
If Zero Flag = 1, then PC ← K

Instruction Format:
JE

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1000_1111	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

JUMP IF NOT EQUAL

Description:
Execute a jump instruction if the Zero Flag is 0.

Operation:
If Zero Flag = 0, then PC ← K

Instruction Format:
JNE

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1001_0000	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

NO OPERATION

Description:
This instruction executes without any effect taking place.

Operation:

Instruction Format:
NOP

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1001_0001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

SET ZERO FLAG

Description:
Set the content in the Zero Flag to 1.

Operation:
Zero Flag ← 1

Instruction Format:
STZ

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1001_0010	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

CLEAR ZERO FLAG

Description:
Set the content of the Zero Flag to 0.

Operation:
Carry Flag ← 0

Instruction Format:
CLZ

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1001_0011	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

DECREMENT JUMP NOT ZERO

Description:
Subtracts to content of the src1_reg by -1- one. Executes a jump instruction is the zero flag is not set.

Operation:
src1_reg ← src1_reg – 1
If Zero Flag = 0, then PC ← K

Instruction Format:
DJNZ src1_reg, LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1001_0100	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

DECREMENT JUMP IF ZERO

Description:
Subtracts to content of the src1_reg by -1- one. Executes a jump instruction is the zero flag is set.

Operation:
src1_reg ← src1_reg – 1
If Zero Flag = 1, then PC ← K

Instruction Format:
DJZ src1_reg, LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1001_0101	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

COMPARE JUMP NOT EQUAL

Description:
Performs an arithmetic subtraction between the contents of the src1_reg and the src2_reg. Executes a jump instruction if the Zero Flag was not set after the subtraction.

Operation:
src1_reg – src2_reg
If Zero Flag = 0, then PC ←K

Instruction Format:
CJNE src1_reg, src2_reg, LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1001_0110	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

COMPARE JUMP IF EQUAL

Description:
Performs an arithmetic subtraction between the contents of the src1_reg and the src2_reg. Executes a jump instruction if the Zero Flag was set after the subtraction.

Operation:
src1_reg – src2_reg
If Zero Flag = 0, then PC ←K

Instruction Format:
CJE src1_reg, src2_reg, LABEL

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1001_0111	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

CLEAR

Description:
Resets all values inside all registers to -0- zero. This includes all registers except for the Program Counter.

Operation:
R0 ← R1 ← R2 ... R32 ← 0

Instruction Format:
CLR

32-bit Opcode:

31 30 29 28 27 26 25 24	23 22 21	20 19 18 17 16	15 14 13	12 11 10 09 08	07 06 05	04 03 02 01 00
1001_1001	000	src1_reg	000	src2_reg	000	dest_reg

Status Register:

C	N	O	P	Z
-	-	-	-	-

Future Enhancements

A 2x2 Matrix datapath module will be added to the Execution Unit that will format the R and S registers into four sections. The four sections of the R registers will be the rows, and the S registers will be the columns. The R registers will then be input into separate ALUs where the matrix algorithms will take place. Each ALU will take in two R registers and two S registers that will be used to perform the selected algorithm. Each instruction will take at least three clock cycles to execute for each row and have the results from the first two clock cycles will be stored into two temporary registers to be used in the last clock cycle, which will perform the algorithm once again but with the temporary registers. The ALU will output 2 bits for each Carry, Negative, Overflow, and Zero flags that will be stored in the flags register and the algorithm results will be stored in the matrix output register.

Another future enhancement will be to add another MUX to the BIU, which will allow the data input/output bus to be set to the 32 most significant bits or the 32 least significant bits of the Matrix buffer depending on the input of the Matrix output enable. A matrix load MUX will also be added to the BIU in order to set the matrix buffer to the 32 most significant bits, 32 least significant bits, or keep the current values.

Verilog Implementation

Top Level Module

```
`timescale 1ps / 500fs
////////////////////////////////////////////////////////////////////////////////
// Author: Jose Trujillo, Sokhom Mom
// Email: jtrujillo2007@gmail.com, smom562@hotmail.com
// Course:          CECS 440
//
// Create Date:    18:06:30 11/12/2012
// Design Name:    CPU_Test_Module
// Project Name:     Final Project
// Target Devices: Xc3s500e-5fg320
// Description: This module interconnects the CPU, Memory, and IO.
//
// Dependencies: CPU, mem, IO
//
////////////////////////////////////////////////////////////////////////////////
module CPU_Test_Module;

   // Inputs
   reg sys_clk;
   reg reset;

   // Outputs
   wire int_ack;
   wire M_CS;
   wire M_RD;
   wire M_WR;
   wire IO_CS;
   wire IO_RD;
   wire IO_WR;
   wire [63:0] Address;
   wire [31:0] Data;
   wire intr;


   // Instantiate the Unit Under Test (UUT)
   CPU cpu(sys_clk, reset, intr, int_ack, M_CS, M_RD, M_WR,
         IO_CS, IO_RD, IO_WR, Address, Data);

   //Instantiate the mem (Memory)
   mem m0 (Address[9:0], Data, M_CS, M_RD, M_WR, sys_clk);
   io io0 (Address[9:0], Data, IO_CS, IO_RD, IO_WR, sys_clk, intr, int_ack);

   //initial $readmemh("mem01_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem02_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem03_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem04_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem05_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem06_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem07_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem08_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem09_64_Fa12.dat", m0.memarray);
   initial $readmemh("mem10_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem11_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem12_64_Fa12.dat", m0.memarray);
   //initial $readmemh("mem13_64_Fa12.dat", m0.memarray); //Floating points
   //initial $readmemh("mem14_64_Fa12_Verify_Interrupts_and_IO.dat", m0.memarray); //Interrupt_Test_Module
          //initial $readmemh("mem15_64_Fa12.dat", m0.memarray); //Vector
          //initial $readmemh("mem16_64_Fa12.dat", m0.memarray);

   always
      #50 sys_clk = ~sys_clk;

   initial begin
      $timeformat(-9, 1, " ns", 6);
      // Initialize Inputs
      sys_clk = 0;
      reset = 0;

      // Wait 100 ns for global reset to finish
      //#100;

      // Add stimulus here
      @(negedge sys_clk)
         reset = 1;
      @(negedge sys_clk)
         reset = 0;

      //$readmemh("Mem_for_CU64.dat", m0.memarray);

   end

endmodule
```

Memory Modules & Log Files

MEMORY MODULE #1

Starting memory locations

```
@0
07_00_00_0F   //          LDI   R15, 0Fh
00_00_00_0F
00_00_00_00
07_00_00_0E   //          LDI   R14, E0h
00_00_00_E0
00_00_00_00
07_00_00_0D   //          LDI   R13, 88888888_0A0A0Ah
0A_0A_0A_0A
88_88_88_88
09_0D_00_0E   //  c_loop: ST    [R14], R13
1A_0D_00_0D   //          ASR   R13
14_0E_00_0E   //          INC   R14
14_0E_00_0E   //          INC   R14
15_0F_00_0F   //          DEC   R15
23_FF_FF_FA   //          JNZ   c_loop
08_0E_00_00   //          LD    R0, [R14]
00_00_0E_01   //          ADD   R1, R0, R14
0A_01_00_02   //          COPY  R2, R1
45_00_00_00   //          HALT

@FE
89_AB_CD_EF
01_23_45_67
```

LOG FILE #1

Registers unaffected during program

```
t=44.0 ns R[00000000] -- 0123456789abcdef R[00000008] -- xxxxxxxxxxxxxxxx
t=45.0 ns R[00000001] -- 0123456789abceed R[00000009] -- xxxxxxxxxxxxxxxx
t=46.0 ns R[00000002] -- 0123456789abceed R[0000000a] -- xxxxxxxxxxxxxxxx
t=47.0 ns R[00000003] -- xxxxxxxxxxxxxxxx R[0000000b] -- xxxxxxxxxxxxxxxx
t=48.0 ns R[00000004] -- xxxxxxxxxxxxxxxx R[0000000c] -- xxxxxxxxxxxxxxxx
t=49.0 ns R[00000005] -- xxxxxxxxxxxxxxxx R[0000000d] -- ffff111111101414
t=50.0 ns R[00000006] -- xxxxxxxxxxxxxxxx R[0000000e] -- 00000000000000fe
t=51.0 ns R[00000007] -- xxxxxxxxxxxxxxxx R[0000000f] -- 0000000000000000
.
.
.
.
.
.
.
.
```

Floating Point And Vector Registers

E0h + (15 loops x (1 inc + 1 inc)) = FEh

```
t=76.0 ns M[000000e0] -- 0a0a0a0a  M[000000f0] -- 880a0a0a
t=77.0 ns M[000000e1] -- 88888888  M[000000f1] -- ff888888
t=78.0 ns M[000000e2] -- 05050505  M[000000f2] -- 44050505
t=79.0 ns M[000000e3] -- c4444444  M[000000f3] -- ffc44444
t=80.0 ns M[000000e4] -- 02828282  M[000000f4] -- 22028282
t=81.0 ns M[000000e5] -- e2222222  M[000000f5] -- ffe22222
t=82.0 ns M[000000e6] -- 01414141  M[000000f6] -- 11014141
t=83.0 ns M[000000e7] -- f1111111  M[000000f7] -- fff11111
t=84.0 ns M[000000e8] -- 80a0a0a0  M[000000f8] -- 8880a0a0
t=85.0 ns M[000000e9] -- f8888888  M[000000f9] -- fff88888
t=86.0 ns M[000000ea] -- 40505050  M[000000fa] -- 44405050
t=87.0 ns M[000000eb] -- fc444444  M[000000fb] -- fffc4444
t=88.0 ns M[000000ec] -- 20282828  M[000000fc] -- 22202828
t=89.0 ns M[000000ed] -- fe222222  M[000000fd] -- fffe2222
t=90.0 ns M[000000ee] -- 10141414  M[000000fe] -- 89abcdef
t=91.0 ns M[000000ef] -- ff111111  M[000000ff] -- 01234567
```

First Value to be Shifted

Content in E0h gets AHR once and written to the next location. Then that location gets ASR and written to the next. Conitues 15 times.

Final Value after 16 Shifts

Starting memory locations

MEMORY MODULE #2

```
@0
07_00_00_0F  //  LDI  R15, F0h
00_00_00_F0
00_00_00_00
08_0F_00_0E  //  LD   R14, [R15]
08_0E_00_0D  //  LD   R13, [R14]
00_0D_0E_0D  //  ADD  R13, R13, R14
25_00_00_04  //  JP   +4   {shouldn't jump}
07_00_00_0C  //  LDI  R12, 8000000000000000h
00_00_00_00
80_00_00_00
00_0C_0D_0C  //  ADD  R12, R12, R13
20_00_00_01  //  JC   +1
45_00_00_00  //  HLT  {shouldn't execute}
09_0C_00_0F  //  ST   [R15], R12
09_0D_00_0E  //  ST   [R14], R13
14_0E_00_0E  //  INC  R14
14_0E_00_0E  //  INC  R14
09_0E_00_0E  //  ST   [R14], R14
07_00_00_0B  //  LDI  R11, 19h
00_00_00_19
00_00_00_00
31_0B_0B_0B  //  JMP  R11
0A_0B_00_0A  //  Three COPY
0A_0A_00_09  //    instructions that
0A_09_00_08  //      shouldn't be done
45_00_00_00  //  HLT

@EF
45_00_00_00
00_00_00_F2
00_00_00_00
FF_FF_FF_0E
7F_FF_FF_FF
FF_FF_FF_FF
5A_5A_5A_5A
12_34_56_78
AB_CD_EF_10
A5_AA_5A_55
FF_00_FF_00
```

LOG FILE #2

Registers unaffected during program

```
t=10.0 ns R[00000000] -- xxxxxxxxxxxxxxxx R[00000008] -- xxxxxxxxxxxxxxxx
t=11.0 ns R[00000001] -- xxxxxxxxxxxxxxxx R[00000009] -- xxxxxxxxxxxxxxxx
t=12.0 ns R[00000002] -- xxxxxxxxxxxxxxxx R[0000000a] -- xxxxxxxxxxxxxxxx
t=13.0 ns R[00000003] -- xxxxxxxxxxxxxxxx R[0000000b] -- 0000000000000019
t=14.0 ns R[00000004] -- xxxxxxxxxxxxxxxx R[0000000c] -- 0000000000000000
t=15.0 ns R[00000005] -- xxxxxxxxxxxxxxxx R[0000000d] -- 8000000000000000
t=16.0 ns R[00000006] -- xxxxxxxxxxxxxxxx R[0000000e] -- 00000000000000f4
t=17.0 ns R[00000007] -- xxxxxxxxxxxxxxxx R[0000000f] -- 00000000000000f0
.
.
.
.
.
.
.
.
```

Floating Point And Vector Registers

```
R12 + R13 = R12
  8000000000000000
+ 8000000000000000
1 0000000000000000
```

ST [R15], R12

```
t=42.0 ns M[000000e0] -- xxxxxxxx  M[000000f0] -- 00000000
t=43.0 ns M[000000e1] -- xxxxxxxx  M[000000f1] -- 00000000
t=44.0 ns M[000000e2] -- xxxxxxxx  M[000000f2] -- 00000000
t=45.0 ns M[000000e3] -- xxxxxxxx  M[000000f3] -- 80000000
t=46.0 ns M[000000e4] -- xxxxxxxx  M[000000f4] -- 000000f4
t=47.0 ns M[000000e5] -- xxxxxxxx  M[000000f5] -- 00000000
t=48.0 ns M[000000e6] -- xxxxxxxx  M[000000f6] -- 12345678
t=49.0 ns M[000000e7] -- xxxxxxxx  M[000000f7] -- abcdef10
t=50.0 ns M[000000e8] -- xxxxxxxx  M[000000f8] -- a5aa5a55
t=51.0 ns M[000000e9] -- xxxxxxxx  M[000000f9] -- ff00ff00
t=52.0 ns M[000000ea] -- xxxxxxxx  M[000000fa] -- xxxxxxxx
t=53.0 ns M[000000eb] -- xxxxxxxx  M[000000fb] -- xxxxxxxx
t=54.0 ns M[000000ec] -- xxxxxxxx  M[000000fc] -- xxxxxxxx
t=55.0 ns M[000000ed] -- xxxxxxxx  M[000000fd] -- xxxxxxxx
t=56.0 ns M[000000ee] -- xxxxxxxx  M[000000fe] -- xxxxxxxx
t=57.0 ns M[000000ef] -- 45000000  M[000000ff] -- xxxxxxxx
```

```
R13 + R14 = R13
  7FFFFFFFFFFFFF0E
+ 00000000000000F2
  8000000000000000
```

MEMORY MODULE #3

Starting memory locations

64-bit content
Lowed Endian on top
Upper Endian on bottom

```
@0
07_00_00_00  // 00  LDI  R0, 07h
00_00_00_07  // 01
00_00_00_00  // 02
31_00_00_00  // 03  JMP  R0
45_00_00_00  // 04  HLT
0A_00_00_1F  // 05  COPY R31, R0
14_1F_00_1F  // 06  INC  R31
0A_00_00_01  // 07  COPY R1, R0
14_01_00_01  // 08  INC  R1
07_00_00_02  // 09  LDI  R2, FEh
00_00_00_FE  // 0A
00_00_00_00  // 0B
01_02_02_03  // 0C  SUB  R3,R2,R2
19_00_00_00  // 0D  LSL  R0
09_00_00_02  // 0E  ST   [R2], R0
14_03_00_03  // 0F  INC  R3
15_02_00_02  // 10  DEC  R2
15_02_00_02  // 11  DEC  R2
0E_01_03_00  // 12  CMP  R1, R3
23_FF_FF_F9  // 13  JNZ  -7
02_01_03_04  // 14  MUL  R4,R1,R3
45_00_00_00  // 15  HLT
```

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
FF_FF_FF_FE
FF_FF_FF_FF
FF_FF_FF_FD
FF_FF_FF_FF
FF_FF_FF_FC
FF_FF_FF_FF
FF_FF_FF_FB
FF_FF_FF_FF
FF_FF_FF_FA
FF_FF_FF_FF
FF_FF_FF_F9
FF_FF_FF_FF
FF_FF_FF_F8
FF_FF_FF_FF
FF_FF_FF_F7
FF_FF_FF_FF
FF_FF_FF_F6
FF_FF_FF_FF
FF_FF_FF_F5
FF_FF_FF_FF
FF_FF_FF_F4
FF_FF_FF_FF
FF_FF_FF_F3
FF_FF_FF_FF
FF_FF_FF_F2
FF_FF_FF_FF
FF_FF_FF_F1
FF_FF_FF_FF
FF_FF_FF_F0
FF_FF_FF_FF
```

Contents from memory location E0 - FF

LOG FILE #3

Registers unaffected during program

```
t=29.0 ns R[00000000] -- 0000000000000700 R[00000008] -- xxxxxxxxxxxxxxxx
t=30.0 ns R[00000001] -- 0000000000000008 R[00000009] -- xxxxxxxxxxxxxxxx
t=31.0 ns R[00000002] -- 00000000000000ee R[0000000a] -- xxxxxxxxxxxxxxxx
t=32.0 ns R[00000003] -- 0000000000000008 R[0000000b] -- xxxxxxxxxxxxxxxx
t=33.0 ns R[00000004] -- 0000000000000040 R[0000000c] -- xxxxxxxxxxxxxxxx
t=34.0 ns R[00000005] -- 0000000000000000 R[0000000d] -- xxxxxxxxxxxxxxxx
t=35.0 ns R[00000006] -- xxxxxxxxxxxxxxxx R[0000000e] -- xxxxxxxxxxxxxxxx
t=36.0 ns R[00000007] -- xxxxxxxxxxxxxxxx R[0000000f] -- xxxxxxxxxxxxxxxx
.
.
.
.
.
.
.
.
```

Floating Point And Vector Registers

Ending of LSL 07h << 8

```
t=61.0 ns M[000000e0] -- ffffffff  M[000000f0] -- 00000700
t=62.0 ns M[000000e1] -- ffffffff  M[000000f1] -- 00000000
t=63.0 ns M[000000e2] -- fffffffe  M[000000f2] -- 00000380
t=64.0 ns M[000000e3] -- ffffffff  M[000000f3] -- 00000000
t=65.0 ns M[000000e4] -- fffffffd  M[000000f4] -- 000001c0
t=66.0 ns M[000000e5] -- ffffffff  M[000000f5] -- 00000000
t=67.0 ns M[000000e6] -- fffffffc  M[000000f6] -- 000000e0
t=68.0 ns M[000000e7] -- ffffffff  M[000000f7] -- 00000000
t=69.0 ns M[000000e8] -- fffffffb  M[000000f8] -- 00000070
t=70.0 ns M[000000e9] -- ffffffff  M[000000f9] -- 00000000
t=71.0 ns M[000000ea] -- fffffffa  M[000000fa] -- 00000038
t=72.0 ns M[000000eb] -- ffffffff  M[000000fb] -- 00000000
t=73.0 ns M[000000ec] -- fffffff9  M[000000fc] -- 0000001c
t=74.0 ns M[000000ed] -- ffffffff  M[000000fd] -- 00000000
t=75.0 ns M[000000ee] -- fffffff8  M[000000fe] -- 0000000e
t=76.0 ns M[000000ef] -- ffffffff  M[000000ff] -- 00000000
```

Beginning of LSL loop with value 07h

MEMORY MODULE #4

Starting memory locations

64-bit content
Lowed Endian on top
Upper Endian on bottom

```
@0
07_00_00_0F  //00   LDI   R15, E0h
00_00_00_E0  //01
00_00_00_00  //02
07_00_00_0E  //03   LDI   R14, E2h
00_00_00_E2  //04
00_00_00_00  //05
07_00_00_0D  //06   LDI   R13, F0h
00_00_00_F0  //07
00_00_00_00  //08
07_00_00_0C  //09   LDI   R12, F2h
00_00_00_F2  //0A
00_00_00_00  //0B
0A_0D_00_0B  //0C   COPY  R11, R13
07_00_00_0A  //0D   LDI   R10, 4h
00_00_00_04  //0E
00_00_00_00  //0F
08_0F_00_09  //10   LD    R9, [R15]
08_0E_00_08  //11   LD    R8, [R14]
09_09_00_0C  //12   ST    [R12], R9
09_08_00_0D  //13   ST    [R13], R8
00_0F_0A_0F  //14   ADD   R15, R15, R10
00_0E_0A_0E  //15   ADD   R14, R14, R10
00_0D_0A_0D  //16   ADD   R13, R13, R10
00_0C_0A_0C  //17   ADD   R12, R12, R10
0E_0B_0F_00  //18   CMP   R11, R15
22_00_00_01  //19   JZ    +1
30_FF_FF_F5  //1A   JMP   -11
03_0D_09_0A  //1B   DIV   R10, R13, R9
06_09_0A_07  //1C   XOR   R7,  R9,  R10
00_08_09_06  //1D   ADD   R6,  R8,  R9
45_00_00_00  //1E   HLT
```

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
00_00_00_01
00_00_00_00
FF_FF_FF_FE
FF_FF_FF_FF
00_00_00_02
00_00_00_00
FF_FF_FF_FD
FF_FF_FF_FF
00_00_00_03
00_00_00_00
FF_FF_FF_FC
FF_FF_FF_FF
00_00_00_04
00_00_00_00
FF_FF_FF_FB
FF_FF_FF_FF
00_00_00_05
00_00_00_00
FF_FF_FF_FA
FF_FF_FF_FF
00_00_00_06
00_00_00_00
FF_FF_FF_F9
FF_FF_FF_FF
00_00_00_07
00_00_00_00
FF_FF_FF_F8
FF_FF_FF_FF
00_00_00_08
00_00_00_00
```

Contents from memory location E0 - FF

LOG FILE #4

Registers unaffected during program

```
t=28.0 ns R[00000000] -- xxxxxxxxxxxxxxxx R[00000008] -- 0000000000000004
t=29.0 ns R[00000001] -- xxxxxxxxxxxxxxxx R[00000009] -- fffffffffffffffc
t=30.0 ns R[00000002] -- xxxxxxxxxxxxxxxx R[0000000a] -- fffffffffffffffc0
t=31.0 ns R[00000003] -- xxxxxxxxxxxxxxxx R[0000000b] -- 0000000000000000
t=32.0 ns R[00000004] -- xxxxxxxxxxxxxxxx R[0000000c] -- 0000000000000102
t=33.0 ns R[00000005] -- xxxxxxxxxxxxxxxx R[0000000d] -- 0000000000000100
t=34.0 ns R[00000006] -- 0000000000000000 R[0000000e] -- 00000000000000f2
t=35.0 ns R[00000007] -- 000000000000003c R[0000000f] -- 00000000000000f0
.
.
.
.
.
.
.
.
```

Floating Point And Vector Registers

R13 / R9 = R10
100 / FFFFFFFFFFFFFFFC = FFFFFFFFFFFFFFC0
FFFFFFFFFFFFFFC0 xor FFFFFFFFFFFFFFFC = 3Ch
R8 + R9 = R6
FFFFFFFFFFFFFFFC + 4h = 0

```
t=60.0 ns M[000000e0] -- ffffffff  M[000000f0] -- 00000001
t=61.0 ns M[000000e1] -- ffffffff  M[000000f1] -- 00000000
t=62.0 ns M[000000e2] -- 00000001  M[000000f2] -- ffffffff
t=63.0 ns M[000000e3] -- 00000000  M[000000f3] -- ffffffff
t=64.0 ns M[000000e4] -- fffffffe  M[000000f4] -- 00000002
t=65.0 ns M[000000e5] -- ffffffff  M[000000f5] -- 00000000
t=66.0 ns M[000000e6] -- 00000002  M[000000f6] -- fffffffe
t=67.0 ns M[000000e7] -- 00000000  M[000000f7] -- ffffffff
t=68.0 ns M[000000e8] -- fffffffd  M[000000f8] -- 00000003
t=69.0 ns M[000000e9] -- ffffffff  M[000000f9] -- 00000000
t=70.0 ns M[000000ea] -- 00000003  M[000000fa] -- fffffffd
t=71.0 ns M[000000eb] -- 00000000  M[000000fb] -- ffffffff
t=72.0 ns M[000000ec] -- fffffffc  M[000000fc] -- 00000004
t=73.0 ns M[000000ed] -- ffffffff  M[000000fd] -- 00000000
t=74.0 ns M[000000ee] -- 00000004  M[000000fe] -- fffffffc
t=75.0 ns M[000000ef] -- 00000000  M[000000ff] -- ffffffff
```

The contents of
E0h = F2h
E4h = F6h
E8h = FAh
ECh = FEh

MEMORY MODULE #5

Starting memory locations

64-bit content
Lowed Endian on top
Upper Endian on bottom

```
@0
32_00_00_04  //00  CALL +04
03_0D_09_0A  //01  DIV  R10, R13, R9
06_09_0A_07  //02  XOR  R7,  R9,  R10
00_08_09_06  //03  ADD  R6,  R8,  R9
45_00_00_00  //04  HALT
07_00_00_0F  //05  LDI  R15, E0h
00_00_00_E0  //06
00_00_00_00  //07
07_00_00_0E  //08  LDI  R14, E2h
00_00_00_E2  //09
00_00_00_00  //0A
07_00_00_0D  //0B  LDI  R13, F0h
00_00_00_F0  //0C
00_00_00_00  //0D
07_00_00_0C  //0E  LDI  R12, F8h
00_00_00_F8  //0F
00_00_00_00  //10
0A_0C_00_0B  //11  COPY R11, R12
07_00_00_0A  //12  LDI  R10, 4
00_00_00_04  //13
00_00_00_00  //14
1A_0A_00_07  //15  ASHR R7,  R10
08_0F_00_09  //16  LD   R9,  R15
08_0E_00_08  //17  LD   R8,  R14
09_09_00_0C  //18  ST   [R12], R9
09_08_00_0D  //19  ST   [R13], R8
00_0F_0A_0F  //1A  ADD  R15, R15, R10
00_0E_0A_0E  //1B  ADD  R14, R14, R10
00_0D_07_0D  //1C  ADD  R13, R13, R7
00_0C_07_0C  //1D  ADD  R12, R12, R7
0E_0D_0B_00  //1E  CMP  R13, R11
23_FF_FF_F6  //1F  JNZ  -10
34_00_00_00  //20  RET
45_00_00_00  //21  HALT   {Safety Net}
```

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
00_00_00_01
00_00_00_00
FF_FF_FF_FE
FF_FF_FF_FF
00_00_00_02
00_00_00_00
FF_FF_FF_FD
FF_FF_FF_FF
00_00_00_03
00_00_00_00
FF_FF_FF_FC
FF_FF_FF_FF
00_00_00_04
00_00_00_00
FF_FF_FF_FB
FF_FF_FF_FF
00_00_00_05
00_00_00_00
FF_FF_FF_FA
FF_FF_FF_FF
00_00_00_06
00_00_00_00
FF_FF_FF_F9
FF_FF_FF_FF
00_00_00_07
00_00_00_00
FF_FF_FF_F8
FF_FF_FF_FF
00_00_00_08
00_00_00_00
```

Contents from memory location E0 - FF

LOG FILE #5

Registers unaffected during program

R13 / R9 = R10
F8/fffffffffffffffc =
fffffffffffffffc2

```
t=29.0 ns R[00000000] -- xxxxxxxxxxxxxxxx R[00000008] -- 0000000000000004
t=30.0 ns R[00000001] -- xxxxxxxxxxxxxxxx R[00000009] -- fffffffffffffffc
t=31.0 ns R[00000002] -- xxxxxxxxxxxxxxxx R[0000000a] -- fffffffffffffffc2
t=32.0 ns R[00000003] -- xxxxxxxxxxxxxxxx R[0000000b] -- 0000000000000000
t=33.0 ns R[00000004] -- xxxxxxxxxxxxxxxx R[0000000c] -- 0000000000000100
t=34.0 ns R[00000005] -- xxxxxxxxxxxxxxxx R[0000000d] -- 00000000000000f8
t=35.0 ns R[00000006] -- 0000000000000000 R[0000000e] -- 00000000000000f2
t=36.0 ns R[00000007] -- 000000000000003e R[0000000f] -- 00000000000000f0
.
.
.
.
.
.
.
.
```

Floating Point And Vector Registers

R8 + R9 = R6

```
    FFFFFFFFFFFFFFFC
+   0000000000000004
1   0000000000000000
```

Contents of E2 and E3 gets written into F0 and F1 via ST [R13], R8 Repeats to FF and gets contents from location from E0 = E0 +4

```
t=61.0 ns M[000000e0] -- ffffffff  M[000000f0] -- 00000001
t=62.0 ns M[000000e1] -- ffffffff  M[000000f1] -- 00000000
t=63.0 ns M[000000e2] -- 00000001  M[000000f2] -- 00000002
t=64.0 ns M[000000e3] -- 00000000  M[000000f3] -- 00000000
t=65.0 ns M[000000e4] -- fffffffe  M[000000f4] -- 00000003
t=66.0 ns M[000000e5] -- ffffffff  M[000000f5] -- 00000000
t=67.0 ns M[000000e6] -- 00000002  M[000000f6] -- 00000004
t=68.0 ns M[000000e7] -- 00000000  M[000000f7] -- 00000000
t=69.0 ns M[000000e8] -- fffffffd  M[000000f8] -- ffffffff
t=70.0 ns M[000000e9] -- ffffffff  M[000000f9] -- ffffffff
t=71.0 ns M[000000ea] -- 00000003  M[000000fa] -- fffffffe
t=72.0 ns M[000000eb] -- 00000000  M[000000fb] -- ffffffff
t=73.0 ns M[000000ec] -- fffffffc  M[000000fc] -- fffffffd
t=74.0 ns M[000000ed] -- ffffffff  M[000000fd] -- ffffffff
t=75.0 ns M[000000ee] -- 00000004  M[000000fe] -- fffffffc
t=76.0 ns M[000000ef] -- 00000000  M[000000ff] -- ffffffff
```

Contents of E0 and E1 gets written into F8 and F9 via ST [R12], R9 Repeats to FF and gets contents from location from E0 = E0 +4

MEMORY MODULE #6

Starting memory locations

64-bit content
Lowed Endian on top
Upper Endian on bottom

Contents from memory location E0 - FF

```
@0                                          @E0
07_00_00_0E  //00  LDI  R14, 4               FF_FF_FF_FF
00_00_00_04  //01                           FF_FF_FF_FF
00_00_00_00  //02                           00_00_00_01
07_00_00_0F  //03  LDI  R15, E0h            00_00_00_00
00_00_00_E0  //04                           FF_FF_FF_FE
00_00_00_00  //05                           FF_FF_FF_FF
08_0F_00_00  //06  LD   R0,  [R15]          00_00_00_02
00_0F_0E_0F  //07  ADD  R15, R15, R14       00_00_00_00
08_0F_00_01  //08  LD   R1,  [R15]          FF_FF_FF_FD
00_0F_0E_0F  //09  ADD  R15, R15, R14       FF_FF_FF_FF
08_0F_00_02  //0A  LD   R2,  [R15]          00_00_00_03
00_0F_0E_0F  //0B  ADD  R15, R15, R14       00_00_00_00
08_0F_00_03  //0C  LD   R3,  [R15]          FF_FF_FF_FC
00_0F_0E_0F  //0D  ADD  R15, R15, R14       FF_FF_FF_FF
08_0F_00_04  //0E  LD   R4,  [R15]          00_00_00_04
00_0F_0E_0F  //0F  ADD  R15, R15, R14       00_00_00_00
08_0F_00_05  //10  LD   R5,  [R15]
00_0F_0E_0F  //11  ADD  R15, R15, R14       FF_FF_FF_FB
08_0F_00_06  //12  LD   R6,  [R15]          FF_FF_FF_FF
00_0F_0E_0F  //13  ADD  R15, R15, R14       00_00_00_05
08_0F_00_07  //14  LD   R7,  [R15]          00_00_00_00
12_07_00_07  //15  NEG  R7                  FF_FF_FF_FA
19_07_00_07  //16  LSL  R7                  FF_FF_FF_FF
52_07_02_07  //17  MULi R7,  R7, 2          00_00_00_06
33_07_00_00  //18  CALL [R7]                00_00_00_00
45_00_00_00  //19  HALT                     FF_FF_FF_F9
                                            FF_FF_FF_FF
@20                                         00_00_00_07
03_06_00_0F  //20  DIV  R15, R6, R0         00_00_00_00
14_0F_00_0F  //21  INC  R15                 FF_FF_FF_F8
07_00_00_0E  //22  LDI  R14, F0h            FF_FF_FF_FF
00_00_00_F0  //23                           00_00_00_08
00_00_00_00  //24                           00_00_00_00
07_00_00_0D  //25  LDI  R13, E0h
00_00_00_E0  //26
00_00_00_00  //27
08_0D_00_0B  //28  LD   R11, [R13]
02_0B_00_0B  //29  MUL  R11, R11, R0
09_0B_00_0E  //2A  ST   [R14], R11
50_0D_02_0D  //2B  ADDi R13, R13, 2
50_0E_02_0E  //2C  ADDi R14, R14, 2
15_0F_00_0F  //2D  DEC  R15
23_FF_FF_F9  //2E  JNZ  -7
34_00_00_00  //3F  RET
45_00_00_00  //30  HALT  {Safety Net}
```

LOG FILE #6

R0 gets loaded with content of E0h. Memory Location gets incremented by 4. Next register gets loaded by content from the incremented memory location. Repeats up to R6.

```
t=44.0 ns R[00000000] -- ffffffffffffffff     R[00000008] -- 0000000000000000
t=45.0 ns R[00000001] -- fffffffffffffffe     R[00000009] -- xxxxxxxxxxxxxxxx
t=46.0 ns R[00000002] -- fffffffffffffffd     R[0000000a] -- xxxxxxxxxxxxxxxx
t=47.0 ns R[00000003] -- fffffffffffffffc     R[0000000b] -- fffffffffffffffc
t=48.0 ns R[00000004] -- fffffffffffffffb     R[0000000c] -- 0000000000000000
t=49.0 ns R[00000005] -- fffffffffffffffa     R[0000000d] -- 00000000000000f0
t=50.0 ns R[00000006] -- fffffffffffffff9     R[0000000e] -- 0000000000000100
t=51.0 ns R[00000007] -- 0000000000000020     R[0000000f] -- 0000000000000000
.
.
.
.
.
.
.
.
```

Floating Point And Vector Registers

~fffffffffffff8h = 7h + 1 = 8h
LSL 8h = 10h x 2 = 20h

R13 = E0h R14 = F0h
ADD 2 to E0h and F0h
Loop 8 times
E0h (2x8) = F0h
F0h (2x8) = 100h

```
t=76.0 ns M[000000e0] -- ffffffff  M[000000f0] -- 00000001
t=77.0 ns M[000000e1] -- ffffffff  M[000000f1] -- 00000000
t=78.0 ns M[000000e2] -- 00000001  M[000000f2] -- ffffffff
t=79.0 ns M[000000e3] -- 00000000  M[000000f3] -- ffffffff
t=80.0 ns M[000000e4] -- fffffffe  M[000000f4] -- 00000002
t=81.0 ns M[000000e5] -- ffffffff  M[000000f5] -- 00000000
t=82.0 ns M[000000e6] -- 00000002  M[000000f6] -- fffffffe
t=83.0 ns M[000000e7] -- 00000000  M[000000f7] -- ffffffff
t=84.0 ns M[000000e8] -- fffffffd  M[000000f8] -- 00000003
t=85.0 ns M[000000e9] -- ffffffff  M[000000f9] -- 00000000
t=86.0 ns M[000000ea] -- 00000003  M[000000fa] -- fffffffd
t=87.0 ns M[000000eb] -- 00000000  M[000000fb] -- ffffffff
t=88.0 ns M[000000ec] -- fffffffc  M[000000fc] -- 00000004
t=89.0 ns M[000000ed] -- ffffffff  M[000000fd] -- 00000000
t=90.0 ns M[000000ee] -- 00000004  M[000000fe] -- fffffffc
t=91.0 ns M[000000ef] -- 00000000  M[000000ff] -- ffffffff
```

Contents of memory location EEh gets written to FCh via ST [R14], R11 after 8 loops

MEMORY MODULE #7

Starting memory locations

64-bit content
Lowed Endian on top
Upper Endian on bottom

```
@0
07_00_00_0F  //00  LDI  R15,
7FFFFFFF_FFFFFFFF
FF_FF_FF_FF  //01
7F_FF_FF_FF  //02
10_0F_00_00  //03  PUSH R15
07_00_00_0E  //04  LDI  R14, 8
00_00_00_08  //05
00_00_00_00  //06
10_0E_00_00  //07  PUSH R14
32_00_00_17  //08  CALL +17h
07_00_00_0D  //09  LDI  R13, E0h
00_00_00_E0  //0A
00_00_00_00  //0B
07_00_00_0C  //0C  LDI  R12, FEh
00_00_00_FE  //0D
00_00_00_00  //0E
07_00_00_0B  //1F  LDI  R11,
E00000000_00000000
00_00_00_00  //10
E0_00_00_00  //11
08_0C_00_0A  //12  LD   R10, [R12]
09_0A_00_0D  //13  ST   [R13], R10
51_0C_02_0C  //14  SUBi R12, R12, 2
50_0D_02_0D  //15  ADDi R13, R13, 2
0E_0B_0A_00  //16  CMP  R11, R10
23_FF_FF_FA  //17  JNZ  -6
0A_00_00_07  //18  COPY R7,  R0
0A_01_00_08  //19  COPY R8,  R1
0A_0F_00_09  //1A  COPY R9,  R15
13_02_00_06  //1B  NOT  R6,  R2
45_00_00_00  //1C  HALT

@20
11_00_00_00  //20  POP  R0
11_00_00_01  //21  POP  R1
11_00_00_02  //22  POP  R2
14_02_00_02  //23  INC  R2
07_00_00_05  //24  LDI  R5, F0h
00_00_00_F0  //25
00_00_00_00  //26
07_00_00_04  //27  LDI  R4, 08h
00_00_00_08  //28
00_00_00_00  //29
03_02_01_02  //2A  DIV  R2, R2, R1
1B_02_00_02  //2B  ASL  R2
09_02_00_05  //2C  ST   [R5], R2
50_05_02_05  //2D  ADDi R5, R5, 2
15_04_00_04  //2E  DEC  R4
23_FF_FF_FA  //2F  JNZ  -6
10_00_00_00  //30  PUSH R0
34_00_00_00  //31  RET
45_00_00_00  //32  HALT    {Safety Net}
```

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
00_00_00_01
00_00_00_00
FF_FF_FF_FE
FF_FF_FF_FF
00_00_00_02
00_00_00_00
FF_FF_FF_FD
FF_FF_FF_FF
00_00_00_03
00_00_00_00
FF_FF_FF_FC
FF_FF_FF_FF
00_00_00_04
00_00_00_00

FF_FF_FF_FB
FF_FF_FF_FF
00_00_00_05
00_00_00_00
FF_FF_FF_FA
FF_FF_FF_FF
00_00_00_06
00_00_00_00
FF_FF_FF_F9
FF_FF_FF_FF
00_00_00_07
00_00_00_00
FF_FF_FF_F8
FF_FF_FF_FF
00_00_00_08
00_00_00_00
```

Contents from memory location E0 - FF

LOG FILE #7

NOT R6, R2

```
t=59.0 ns R[00000000] -- 0000000000000009 R[00000008] -- 0000000000000008
t=60.0 ns R[00000001] -- 0000000000000008 R[00000009] -- 7fffffffffffffff
t=61.0 ns R[00000002] -- ffff800000000000 R[0000000a] -- e000000000000000
t=62.0 ns R[00000003] -- 0000000000000000 R[0000000b] -- e000000000000000
t=63.0 ns R[00000004] -- 0000000000000000 R[0000000c] -- 00000000000000ee
t=64.0 ns R[00000005] -- 0000000000000100 R[0000000d] -- 00000000000000f0
t=65.0 ns R[00000006] -- 00007fffffffffff R[0000000e] -- 0000000000000008
t=66.0 ns R[00000007] -- 0000000000000009 R[0000000f] -- 7fffffffffffffff
.
.
.
.
.
.
.
.
```

COPY R9, R15

Floating Point And Vector Registers

```
t=91.0 ns M[000000e0] -- 00000000  M[000000f0] -- 00000000
t=92.0 ns M[000000e1] -- ffff8000  M[000000f1] -- e0000000
t=93.0 ns M[000000e2] -- 00000000  M[000000f2] -- 00000000
t=94.0 ns M[000000e3] -- fffe0000  M[000000f3] -- f8000000
t=95.0 ns M[000000e4] -- 00000000  M[000000f4] -- 00000000
t=96.0 ns M[000000e5] -- fff80000  M[000000f5] -- fe000000
t=97.0 ns M[000000e6] -- 00000000  M[000000f6] -- 00000000
t=98.0 ns M[000000e7] -- ffe00000  M[000000f7] -- ff800000
t=99.0 ns M[000000e8] -- 00000000  M[000000f8] -- 00000000
t=100.0 ns M[000000e9] -- ff800000 M[000000f9] -- ffe00000
t=101.0 ns M[000000ea] -- 00000000 M[000000fa] -- 00000000
t=102.0 ns M[000000eb] -- fe000000 M[000000fb] -- fff80000
t=103.0 ns M[000000ec] -- 00000000 M[000000fc] -- 00000000
t=104.0 ns M[000000ed] -- f8000000 M[000000fd] -- fffe0000
t=105.0 ns M[000000ee] -- 00000000 M[000000fe] -- 00000000
t=106.0 ns M[000000ef] -- e0000000 M[000000ff] -- ffff8000
```

LOOP 8 times
Memory Location E0h ← Contents FEh
Increment E0 by 2.
Decrement FE by 2.

MEMORY MODULE #8

Starting memory locations

64-bit content
Lowed Endian on top
Upper Endian on bottom

Contents from memory location E0 – EF

```
@0
07_00_00_00  //00   LDI   R0,
F0F0F0F0_F0F0F0F0
F0_F0_F0_F0  //01
F0_F0_F0_F0  //02
07_00_00_01  //03   LDI   R1,
A5A5A5A5_A5A5A5A5
A5_A5_A5_A5  //04
A5_A5_A5_A5  //05
07_00_00_02  //06   LDI   R2,
0A0A0A0A_0A0A0A0A
0A_0A_0A_0A  //07
0A_0A_0A_0A  //08
07_00_00_03  //09   LDI   R3,
E3E3E3E3_E3E3E3E3
E3_E3_E3_E3  //0A
E3_E3_E3_E3  //0B
07_00_00_04  //0C   LDI   R4, E0h
00_00_00_E0  //0D
00_00_00_00  //0E
07_00_00_05  //0F   LDI   R5, 07h
00_00_00_07  //10
00_00_00_00  //11
03_04_05_07  //12   DIV    R7, R4, R5
14_05_00_05  //13   INC    R5
53_05_02_05  //14   DIVi   R5, R5, 2
33_07_00_00  //15   CALL  [R7]
13_05_00_05  //16   NOT    R5
0A_05_00_06  //17   COPY   R6, R5
45_00_00_00  //18   HALT
```

```
@E0
FF_FF_FF_FF
00_00_00_01
FF_FF_FF_FE
00_00_00_02
FF_FF_FF_FD
00_00_00_03
FF_FF_FF_FC
00_00_00_04
FF_FF_FF_FB
00_00_00_05
FF_FF_FF_FA
00_00_00_06
FF_FF_FF_F9
00_00_00_07
FF_FF_FF_F8
00_00_00_08
```

```
@20
1D_00_00_00  //20   ROL  R0
09_00_00_04  //21   ST  [R4], R0
50_04_02_04  //22   ADDi R4,  R4, 2
1D_01_00_01  //23   ROL  R1
09_01_00_04  //24   ST  [R4], R1
50_04_02_04  //25   ADDi R4,  R4, 2
1C_02_00_02  //26   ROR  R2
09_02_00_04  //27   ST  [R4], R2
50_04_02_04  //28   ADDi R4,  R4, 2
1C_03_00_03  //29   ROR  R3
09_03_00_04  //2A   ST  [R4], R3
50_04_02_04  //2B   ADDi R4,  R4, 2
15_05_00_05  //2C   DEC  R5
22_00_00_01  //2D   JZ   +1
31_07_00_00  //2E   JMP [R7]
34_00_00_00  //2F   RET
45_00_00_00  //30   HALT  {Safety Net}
```

LOG FILE #8

Initial values before being rotated

DEC R5 till R5 = 0, then NOT R5
R5 = FFFFFFFFFFFFFFFFh
COPY R6, R5

```
t=36.0 ns R[00000000] -- 0f0f0f0f0f0f0f0f R[00000008] -- 0000000000000000
t=37.0 ns R[00000001] -- 5a5a5a5a5a5a5a5a R[00000009] -- xxxxxxxxxxxxxxxx
t=38.0 ns R[00000002] -- a0a0a0a0a0a0a0a0 R[0000000a] -- xxxxxxxxxxxxxxxx
t=39.0 ns R[00000003] -- 3e3e3e3e3e3e3e3e R[0000000b] -- xxxxxxxxxxxxxxxx
t=40.0 ns R[00000004] -- 0000000000000100 R[0000000c] -- xxxxxxxxxxxxxxxx
t=41.0 ns R[00000005] -- ffffffffffffffff R[0000000d] -- xxxxxxxxxxxxxxxx
t=42.0 ns R[00000006] -- ffffffffffffffff R[0000000e] -- xxxxxxxxxxxxxxxx
t=43.0 ns R[00000007] -- 0000000000000020 R[0000000f] -- xxxxxxxxxxxxxxxx
.
.
.
.
.
.
.
.
t=68.0 ns M[000000e0] -- e1e1e1e1  M[000000f0] -- 87878787
t=69.0 ns M[000000e1] -- e1e1e1e1  M[000000f1] -- 87878787
t=70.0 ns M[000000e2] -- 4b4b4b4b  M[000000f2] -- 2d2d2d2d
t=71.0 ns M[000000e3] -- 4b4b4b4b  M[000000f3] -- 2d2d2d2d
t=72.0 ns M[000000e4] -- 05050505  M[000000f4] -- 41414141
t=73.0 ns M[000000e5] -- 05050505  M[000000f5] -- 41414141
t=74.0 ns M[000000e6] -- f1f1f1f1  M[000000f6] -- 7c7c7c7c
t=75.0 ns M[000000e7] -- f1f1f1f1  M[000000f7] -- 7c7c7c7c
t=76.0 ns M[000000e8] -- c3c3c3c3  M[000000f8] -- 0f0f0f0f
t=77.0 ns M[000000e9] -- c3c3c3c3  M[000000f9] -- 0f0f0f0f
t=78.0 ns M[000000ea] -- 96969696  M[000000fa] -- 5a5a5a5a
t=79.0 ns M[000000eb] -- 96969696  M[000000fb] -- 5a5a5a5a
t=80.0 ns M[000000ec] -- 82828282  M[000000fc] -- a0a0a0a0
t=81.0 ns M[000000ed] -- 82828282  M[000000fd] -- a0a0a0a0
t=82.0 ns M[000000ee] -- f8f8f8f8  M[000000fe] -- 3e3e3e3e
t=83.0 ns M[000000ef] -- f8f8f8f8  M[000000ff] -- 3e3e3e3e
```

R1 = 5A5A5A5A5A5A5A5Ah
Rotated left 1, 2, 3 and 4 times

R0 = 0F0F0F0F0F0F0F0Fh
Rotated left 1, 2, 3 and 4 times

R2 = A0A0A0A0A0A0A0A0h
Rotated left 1, 2, 3 and 4 times

R2 = A0A0A0A0A0A0A0A0h
Rotated left 1, 2, 3 and 4 times

MEMORY MODULE #9

Starting memory locations

64-bit content
Lowed Endian on top
Upper Endian on bottom

Contents from memory location E0 - FF

```
@0                                          E0
07_00_00_0F  //00  LDI   R15, E0h           C3_C3_C3_C3
00_00_00_E0  //01                          C3_C3_C3_C3
00_00_00_00  //02                          1E_1E_1E_1E
07_00_00_0E  //03  LDI   R14, F0h           1E_1E_1E_1E
00_00_00_F0  //04                          F1_F1_F1_F0
00_00_00_00  //05                          F1_F1_F1_F1
07_00_00_0D  //06  LDI   R13, 8             07_07_07_07
00_00_00_08  //07                          07_07_07_07
00_00_00_00  //08                          E3_E3_E3_E1
19_0D_00_0D  //09  LSL   R13                E3_E3_E3_E3
19_0D_00_0D  //0A  LSL   R13                0E_0E_0E_0E
33_0D_00_00  //0B  CALL [R13]               0E_0E_0E_0E
08_0F_00_03  //0C  LD    R3, [R15]          17_17_17_14
0A_03_00_04  //0D  COPY  R4,  R3            17_17_17_17
0A_04_00_05  //0E  COPY  R5,  R4            74_74_74_74
0A_05_00_06  //0F  COPY  R6,  R5            74_74_74_74
0A_06_00_07  //10  COPY  R7,  R6            27_27_27_23
0A_07_00_08  //11  COPY  R8,  R7            27_27_27_27
0A_08_00_09  //12  COPY  R9,  R8            6C_6C_6C_6C
0A_09_00_0A  //13  COPY  R10, R9            6C_6C_6C_6C
0A_0A_00_0B  //14  COPY  R11, R10           4B_4B_4B_4E
45_00_00_00  //15  HALT                     4B_4B_4B_4b
                                            5A_5A_5A_5A
@20                                         5A_5A_5A_5A
07_00_00_0C  //20  LDI  R12, 100h           81_81_81_87
00_00_01_00  //21                          81_81_81_81
00_00_00_00  //22                          3F_3F_3F_3F
08_0F_00_00  //23  LD   R0,  [R15]          3F_3F_3F_3F
50_0F_02_0F  //24  ADDi R15,  R15, 2        73_73_73_74
08_0F_00_01  //25  LD   R1,  [R15]          73_73_73_73
50_0F_02_0F  //26  ADDi R15,  R15, 2        46_46_46_46
1D_01_00_01  //27  ROL  R1                  46_46_46_46
06_00_01_02  //28  XOR  R2,   R0, R1
09_02_00_0E  //29  ST  [R14], R2
50_0E_02_0E  //2A  ADDi R14,  R14, 2
12_02_00_02  //2B  NEG  R2
09_02_00_0E  //2C  ST  [R14], R2
50_0E_02_0E  //2D  ADDi R14,  R14, 2
0E_0E_0C_00  //2E  CMP  R14,  R12
28_FF_FF_F3  //2F  JLT  -13
34_00_00_00  //30  RET
45_00_00_00  //31  HALT  {Safety Net}
```

LOG FILE #9

R3 copied into R4, R5, R6, R7, R8, R9, R10, and R11

R0 xor R1 = R2
NEGATE R2

```
t=35.0 ns R[00000000] -- 1717171717171714 R[00000008] -- ffffffffffffffff
t=36.0 ns R[00000001] -- e8e8e8e8e8e8e8e8 R[00000009] -- ffffffffffffffff
t=37.0 ns R[00000002] -- 0000000000000004 R[0000000a] -- ffffffffffffffff
t=38.0 ns R[00000003] -- ffffffffffffffff R[0000000b] -- ffffffffffffffff
t=39.0 ns R[00000004] -- ffffffffffffffff R[0000000c] -- 0000000000000100
t=40.0 ns R[00000005] -- ffffffffffffffff R[0000000d] -- 0000000000000020
t=41.0 ns R[00000006] -- ffffffffffffffff R[0000000e] -- 0000000000000100
t=42.0 ns R[00000007] -- ffffffffffffffff R[0000000f] -- 00000000000000f0
.
.
.
.
.
.
.
.
```

ROL 1E1E1E1E1E1E1E1E =
3C3C3C3C3C3C3C3C
xor C3C3C3C3C3C3C3C3
FFFFFFFFFFFFFFFF

```
t=67.0 ns M[000000e0] -- c3c3c3c3  M[000000f0] -- ffffffff
t=68.0 ns M[000000e1] -- c3c3c3c3  M[000000f1] -- ffffffff
t=69.0 ns M[000000e2] -- 1e1e1e1e  M[000000f2] -- 00000001
t=70.0 ns M[000000e3] -- 1e1e1e1e  M[000000f3] -- 00000000
t=71.0 ns M[000000e4] -- f1f1f1f0  M[000000f4] -- fffffffe
t=72.0 ns M[000000e5] -- f1f1f1f1  M[000000f5] -- ffffffff
t=73.0 ns M[000000e6] -- 07070707  M[000000f6] -- 00000002
t=74.0 ns M[000000e7] -- 07070707  M[000000f7] -- 00000000
t=75.0 ns M[000000e8] -- e3e3e3e1  M[000000f8] -- fffffffd
t=76.0 ns M[000000e9] -- e3e3e3e3  M[000000f9] -- ffffffff
t=77.0 ns M[000000ea] -- 0e0e0e0e  M[000000fa] -- 00000003
t=78.0 ns M[000000eb] -- 0e0e0e0e  M[000000fb] -- 00000000
t=79.0 ns M[000000ec] -- 17171714  M[000000fc] -- fffffffc
t=80.0 ns M[000000ed] -- 17171717  M[000000fd] -- ffffffff
t=81.0 ns M[000000ee] -- 74747474  M[000000fe] -- 00000004
t=82.0 ns M[000000ef] -- 74747474  M[000000ff] -- 00000000
```

NEGATE

NEGATE

NEGATE

MEMORY MODULE #10

Starting memory locations

```
@0
07_00_00_0F  //00  LDI  R15, F0h
00_00_00_F0  //01
00_00_00_00  //02
07_00_00_0E  //03  LDI   R14, E0h
00_00_00_E0  //04
00_00_00_00  //05
32_00_00_19  //06  CALL  @20h
20_00_00_08  //07  JC    @10h
02_00_01_02  //08  MUL   R2,   R0, R1
09_02_00_0F  //09  ST   [R15], R2
50_0F_02_0F  //0A  ADDi  R15,  R15, 2
12_00_00_00  //0B  NEG   R0
03_01_00_03  //0C  DIV   R3,   R1,  R0
09_03_00_0F  //0D  ST   [R15], R3
50_0F_02_0F  //0E  ADDi  R15,  R15, 2
30_FF_FF_F6  //0F  JMP   @6
02_00_01_02  //10  MUL   R2,   R0, R1
09_02_00_0F  //11  ST   [R15], R2
50_0F_02_0F  //12  ADDi  R15,  R15, 2
12_01_00_01  //13  NEG   R1
03_00_01_03  //14  DIV   R3,  R0,  R1
09_03_00_0F  //15  ST   [R15], R3
45_00_00_00  //16  HALT

@20
08_0E_00_00  //20  LD   R0, [R14]
50_0E_02_0E  //21  ADDi R14, R14, 2
08_0E_00_01  //22  LD   R1, [R14]
50_0E_02_0E  //23  ADDi R14, R14, 2
0E_00_01_00  //24  CMP  R0,  R1
29_00_00_02  //25  JGE  @28h
41_00_00_00  //26  STC
34_00_00_00  //27  RET
40_00_00_00  //28  CLC
34_00_00_00  //29  RET
55_00_00_00  //2A  HALT    {Safety Net}
```

```
@E0
FF_FF_FF_FF  //E0 -1
FF_FF_FF_FF  //E1
FF_FF_FF_FE  //E2 -2
FF_FF_FF_FF  //E3
FF_FF_FF_FD  //E4 -3
FF_FF_FF_FF  //E5
FF_FF_FF_FC  //E6 -4
FF_FF_FF_FF  //E7
FF_FF_FF_FB  //E8 -5
FF_FF_FF_FF  //E9
FF_FF_FF_FA  //EA -6
FF_FF_FF_FF  //EB
FF_FF_FF_F9  //EC -7
FF_FF_FF_FF  //ED
FF_FF_FF_FF  //EE -1
FF_FF_FF_FF  //EF
FF_FF_FF_F7  //F0 -9
FF_FF_FF_FF  //F1
FF_FF_FF_F6  //F2 -10
FF_FF_FF_FF  //F3
FF_FF_FF_F5  //F4 -11
FF_FF_FF_FF  //F5
FF_FF_FF_F4  //F6 -12
FF_FF_FF_FF  //F7
FF_FF_FF_F3  //F8 -13
FF_FF_FF_FF  //F9
FF_FF_FF_F2  //FA -14
FF_FF_FF_FF  //FB
FF_FF_FF_F1  //FC -15
FF_FF_FF_FF  //FD
FF_FF_FF_F0  //FE -16
FF_FF_FF_FF  //FF
```

LOG FILE #10

Memory Location EEh gets NEGATE put into R1

```
t=38.0 ns R[00000000] -- fffffffffffffff9 R[00000008] -- xxxxxxxxxxxxxxxx
t=39.0 ns R[00000001] -- 0000000000000001 R[00000009] -- xxxxxxxxxxxxxxxx
t=40.0 ns R[00000002] -- 0000000000000007 R[0000000a] -- xxxxxxxxxxxxxxxx
t=41.0 ns R[00000003] -- fffffffffffffff9 R[0000000b] -- xxxxxxxxxxxxxxxx
t=42.0 ns R[00000004] -- 0000000000000000 R[0000000c] -- xxxxxxxxxxxxxxxx
t=43.0 ns R[00000005] -- xxxxxxxxxxxxxxxx R[0000000d] -- xxxxxxxxxxxxxxxx
t=44.0 ns R[00000006] -- xxxxxxxxxxxxxxxx R[0000000e] -- 00000000000000f0
t=45.0 ns R[00000007] -- xxxxxxxxxxxxxxxx R[0000000f] -- 00000000000000fe
.
.
.
.
.
.
.
.
.
t=70.0 ns M[000000e0] -- ffffffff  M[000000f0] -- 00000002
t=71.0 ns M[000000e1] -- ffffffff  M[000000f1] -- 00000000
t=72.0 ns M[000000e2] -- fffffffe  M[000000f2] -- fffffffe
t=73.0 ns M[000000e3] -- ffffffff  M[000000f3] -- ffffffff
t=74.0 ns M[000000e4] -- fffffffd  M[000000f4] -- 0000000c
t=75.0 ns M[000000e5] -- ffffffff  M[000000f5] -- 00000000
t=76.0 ns M[000000e6] -- fffffffc  M[000000f6] -- ffffffff
t=77.0 ns M[000000e7] -- ffffffff  M[000000f7] -- ffffffff
t=78.0 ns M[000000e8] -- fffffffb  M[000000f8] -- 0000001e
t=79.0 ns M[000000e9] -- ffffffff  M[000000f9] -- 00000000
t=80.0 ns M[000000ea] -- fffffffa  M[000000fa] -- ffffffff
t=81.0 ns M[000000eb] -- ffffffff  M[000000fb] -- ffffffff
t=82.0 ns M[000000ec] -- fffffff9  M[000000fc] -- 00000007
t=83.0 ns M[000000ed] -- ffffffff  M[000000fd] -- 00000000
t=84.0 ns M[000000ee] -- ffffffff  M[000000fe] -- fffffff9
t=85.0 ns M[000000ef] -- ffffffff  M[000000ff] -- ffffffff
```

R0 / R1 = R3

Remainder of R0 / R1

MEMORY MODULE #11

Starting memory locations

```
@0
07_00_00_00  //00  LDI  R0,  13h        ;R0 <-- 13h
00_00_00_13  //01
00_00_00_00  //02
51_00_05_01  //03  SUBi R1,  R0,  05h   ;R1 <-- 0Eh   for 13h^^(Eh)
0A_00_00_02  //04  COPY R2,  R0         ;R2 <-- 13h
07_00_00_0F  //05  LDI  R15, E0h        ;intermiediate results buffer
00_00_00_E0  //06
00_00_00_00  //07
50_01_12_0E  //08  ADDi  R14,  R1,  12h ; R14 <-- 20h
33_0E_00_00  //09  CALL [R14]
09_03_00_0F  //0A  STO  [R15], R3
45_00_00_00  //0B  HALT
```

```
@20
5B_01_00_00  //20  CMPi  R1, 0
23_00_00_01  //21  JNE   @23
34_00_00_00  //22  RET

02_02_00_02  //23  MUL    R2,  R2,  R0
15_01_00_01  //24  DEC    R1
09_02_00_0F  //25  STO  [R15], R2
50_0F_02_0F  //26  ADDi  R15, R15,  2
33_0E_00_00  //27  CALL [R14]
30_FF_FF_F7  //28  JMP   @20
```

64-bit content
Lowed Endian on top
Upper Endian on bottom

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
FF_FF_FF_FE
FF_FF_FF_FF
FF_FF_FF_FD
FF_FF_FF_FF
FF_FF_FF_FC
FF_FF_FF_FF
FF_FF_FF_FB
FF_FF_FF_FF
FF_FF_FF_FA
FF_FF_FF_FF
FF_FF_FF_F9
FF_FF_FF_FF
FF_FF_FF_F8
FF_FF_FF_FF
FF_FF_FF_F7
FF_FF_FF_FF
FF_FF_FF_F6
FF_FF_FF_FF
FF_FF_FF_F5
FF_FF_FF_FF
FF_FF_FF_F4
FF_FF_FF_FF
FF_FF_FF_F3
FF_FF_FF_FF
FF_FF_FF_F2
FF_FF_FF_FF
FF_FF_FF_F1
FF_FF_FF_FF
FF_FF_FF_F0
FF_FF_FF_FF
```

Contents from memory location E0 - EF

LOG FILE #11

R15 = FCh
STO [R15], R3

```
t=81.0 ns R[00000000] -- 0000000000000013 R[00000008] -- xxxxxxxxxxxxxxxx
t=82.0 ns R[00000001] -- 0000000000000000 R[00000009] -- xxxxxxxxxxxxxxxx
t=83.0 ns R[00000002] -- d2ae3299c1c4aedb R[0000000a] -- xxxxxxxxxxxxxxxx
t=84.0 ns R[00000003] -- fffffffffffffffc R[0000000b] -- xxxxxxxxxxxxxxxx
t=85.0 ns R[00000004] -- xxxxxxxxxxxxxxxx R[0000000c] -- xxxxxxxxxxxxxxxx
t=86.0 ns R[00000005] -- xxxxxxxxxxxxxxxx R[0000000d] -- xxxxxxxxxxxxxxxx
t=87.0 ns R[00000006] -- xxxxxxxxxxxxxxxx R[0000000e] -- 0000000000000020
t=88.0 ns R[00000007] -- xxxxxxxxxxxxxxxx R[0000000f] -- 00000000000000fc
.
.
.
.
.
.
.
.
```

Registers unaffected during program

13h x 13h = 169h
Beginning of loop

```
t=113.0 ns M[000000e0] -- 00000169 M[000000f0] -- 8006c189
t=114.0 ns M[000000e1] -- 00000000 M[000000f1] -- 00000593
t=115.0 ns M[000000e2] -- 00001acb M[000000f2] -- 80805d2b
t=116.0 ns M[000000e3] -- 00000000 M[000000f3] -- 000069f2
t=117.0 ns M[000000e4] -- 0001fd11 M[000000f4] -- 8986ea31
t=118.0 ns M[000000e5] -- 00000000 M[000000f5] -- 0007dcff
t=119.0 ns M[000000e6] -- 0025c843 M[000000f6] -- 350361a3
t=120.0 ns M[000000e7] -- 00000000 M[000000f7] -- 009566f7
t=121.0 ns M[000000e8] -- 02cddcf9 M[000000f8] -- ef403f19
t=122.0 ns M[000000e9] -- 00000000 M[000000f9] -- 0b16a458
t=123.0 ns M[000000ea] -- 3547667b M[000000fa] -- c1c4aedb
t=124.0 ns M[000000eb] -- 00000000 M[000000fb] -- d2ae3299
t=125.0 ns M[000000ec] -- f44c9b21 M[000000fc] -- fffffffc
t=126.0 ns M[000000ed] -- 00000003 M[000000fd] -- ffffffff
t=127.0 ns M[000000ee] -- 21af8373 M[000000fe] -- fffffff0
t=128.0 ns M[000000ef] -- 0000004b M[000000ff] -- ffffffff
```

x 13h
x 13h
x 13h
x 13h
x 13h
x 13h
x 13h

End of loop

Continues to memory location FAh

MEMORY MODULE #12

```
@0
07_00_00_0F   //  LDI R15, 7FFFFFFF_FFFFFFFF
FF_FF_FF_FF
7F_FF_FF_FF
07_00_00_0E   //  LDI R14, 80000000_00000000
00_00_00_00
80_00_00_00
01_0E_0E_0D   //  SUB R13, R14, R14              {R13 <- 00000000_00000000, V <- 0}
27_00_00_01   //  JNV +1
45_00_00_00   //  HALT             {no halt}
01_0E_0F_0C   //  SUB R12, R14, R15              {R12 <- 00000000_00000001, V <- 1}
26_00_00_01   //  JV  +1
45_00_00_00   //  HALT             {no halt}
28_00_00_01   //  JLT +1
45_00_00_00   //  HALT             {no halt}
2B_00_00_01   //  JLE +1
45_00_00_00   //  HALT             {no halt}
01_0F_0E_0B   //  SUB R11, R15, R14              {R11 <- FFFFFFFF_FFFFFFFF}
29_00_00_01   //  JGE +1
45_00_00_00   //  HALT             {no halt}
2A_00_00_01   //  JGT +1
45_00_00_00   //  HALT             {no halt}
01_0E_0F_0A   //  SUB R10, R14, R15              {R10 <- 00000000_00000001}
2D_00_00_01   //  JAE +1
45_00_00_00   //  HALT             {no halt}
2E_00_00_01   //  JA  +1
45_00_00_00   //  HALT             {no halt}
01_0F_0E_09   //  SUB R09, R15, R14              {R09 <- FFFFFFFF_FFFFFFFF}
2C_00_00_01   //  JB  +1
45_00_00_00   //  HALT             {no halt}
2F_00_00_01   //  JBE +1
45_00_00_00   //  HALT             {no halt}
14_0E_00_08   //  INC   R08, R14                 {R08 <- 80000000_00000001, V <- 0}
27_00_00_01   //  JNV +1
45_00_00_00   //  HALT             {no halt}
14_0F_00_07   //  INC   R07, R15                 {R07 <- 80000000_00000000, V <- 1}
26_00_00_01   //  JV  +1
45_00_00_00   //  HALT             {no halt}
15_0E_00_06   //  DEC   R06, R14                 {R06 <- 7FFFFFFF_FFFFFFFF, V <- 1}
26_00_00_01   //  JV  +1
45_00_00_00   //  HALT             {no halt}
15_0F_00_05   //  DEC   R05, R15                 {R05 <- 7FFFFFFF_FFFFFFFE, V <- 0}
27_00_00_01   //  JNV +1
45_00_00_00   //  HALT             {no halt}
1B_0E_00_04   //  ASL   R04, R14                 {R04 <- 80000000_00000000, V <- 1}
26_00_00_01   //  JV  +1
45_00_00_00   //  HALT             {no halt}
1B_0F_00_03   //  ASL   R03, R15                 {R03 <- 7FFFFFFF_FFFFFFFE, V <- 1}
26_00_00_01   //  JV  +1
45_00_00_00   //  HALT             {no halt}
1B_0B_00_02   //  ASL   R02, R11                 {R02 <- FFFFFFFF_FFFFFFFE, V <- 0}
27_00_00_01   //  JNV +1
45_00_00_00   //  HALT             {no halt}
1B_0A_00_01   //  ASL   R01, R10                 {R01 <- 00000000_00000002, V <- 0}
27_00_00_01   //  JNV +1
45_00_00_00   //  HALT             {no halt}
10_0F_00_00   //  PUSH  R15
11_00_00_00   //  POP   R0                       {R00 <- 7FFFFFFF_FFFFFFFF}
45_00_00_00   //  HALT             {Halt here!}
```

MEMORY MODULE #12

64-bit content
Lowed Endian on top
Upper Endian on bottom

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
00_00_00_01
00_00_00_00
FF_FF_FF_FE
FF_FF_FF_FF
00_00_00_02
00_00_00_00
FF_FF_FF_FD
FF_FF_FF_FF
00_00_00_03
00_00_00_00
FF_FF_FF_FC
FF_FF_FF_FF
00_00_00_04
00_00_00_00

FF_FF_FF_FB
FF_FF_FF_FF
00_00_00_05
00_00_00_00
FF_FF_FF_FA
FF_FF_FF_FF
00_00_00_06
00_00_00_00
FF_FF_FF_F9
FF_FF_FF_FF
00_00_00_07
00_00_00_00
FF_FF_FF_F8
FF_FF_FF_FF
00_00_00_08
00_00_00_00
```

Contents from memory location E0 - FF

LOG FILE #12

ASL R0, R10

PUSH R15
POP R0

INC R8, R14

```
t=17.0 ns R[00000000] -- 7fffffffffffffff R[00000008] -- 8000000000000001
t=18.0 ns R[00000001] -- 0000000000000002 R[00000009] -- ffffffffffffffff
t=19.0 ns R[00000002] -- fffffffffffffffe R[0000000a] -- 0000000000000001
t=20.0 ns R[00000003] -- 7fffffffffffffe R[0000000b] -- ffffffffffffffff
t=21.0 ns R[00000004] -- 8000000000000000 R[0000000c] -- 0000000000000001
t=22.0 ns R[00000005] -- 7fffffffffffffe R[0000000d] -- 0000000000000000
t=23.0 ns R[00000006] -- 7fffffffffffffff R[0000000e] -- 8000000000000000
t=24.0 ns R[00000007] -- 8000000000000000 R[0000000f] -- 7fffffffffffffff
.
.
.
.
.
.
.
.
t=45.0 ns M[000000e0] -- ffffffff  M[000000f0] -- fffffffb
t=46.0 ns M[000000e1] -- ffffffff  M[000000f1] -- ffffffff
t=47.0 ns M[000000e2] -- 00000001  M[000000f2] -- 00000005
t=48.0 ns M[000000e3] -- 00000000  M[000000f3] -- 00000000
t=49.0 ns M[000000e4] -- fffffffe  M[000000f4] -- fffffffa
t=50.0 ns M[000000e5] -- ffffffff  M[000000f5] -- ffffffff
t=51.0 ns M[000000e6] -- 00000002  M[000000f6] -- 00000006
t=52.0 ns M[000000e7] -- 00000000  M[000000f7] -- 00000000
t=53.0 ns M[000000e8] -- fffffffd  M[000000f8] -- fffffff9
t=54.0 ns M[000000e9] -- ffffffff  M[000000f9] -- ffffffff
t=55.0 ns M[000000ea] -- 00000003  M[000000fa] -- 00000007
t=56.0 ns M[000000eb] -- 00000000  M[000000fb] -- 00000000
t=57.0 ns M[000000ec] -- fffffffc  M[000000fc] -- fffffff8
t=58.0 ns M[000000ed] -- ffffffff  M[000000fd] -- ffffffff
t=59.0 ns M[000000ee] -- 00000004  M[000000fe] -- 00000008
t=60.0 ns M[000000ef] -- 00000000  M[000000ff] -- 00000000
```

ASL R4, R14

INC R7, R15

SUB R13, R14, R14

MEMORY MODULE #13

Starting memory locations

```
@0
07_00_00_00    // LDI   R00, E0h
00_00_00_E0
00_00_00_00
69_00_00_0F    // FLD   F15, [R0]
50_00_02_00    // ADDi  R00,  #2
69_00_00_0E    // FLD   F14, [R0]
50_00_02_00    // ADDi  R00,  #2
63_0E_0F_0D    // FDIV  F13,  F14, F15
6A_0D_00_00    // FSTO  [R0], F13
50_00_02_00    // ADDi  R00,   #2
62_0F_0E_0C    // FMUL  F12,  F15, F14
6A_0C_00_00    // FSTO  [R0], F12
50_00_02_00    // ADDi  R00,   #2
69_00_00_0B    // FLD   F11,  [R0]
50_00_02_00    // ADDi  R00,   #2
69_00_00_0A    // FLD   F10,  [R0]
50_00_02_00    // ADDi  R00,   #2
60_0A_0B_09    // FADD  F09,  F10, F11
6A_09_00_00    // FSTO  [R0], F09
50_00_02_00    // ADDi  R00,   #2
61_0A_0B_08    // FSUB  F08,  F10, F11
6A_08_00_00    // FSTO  [R0], F08
67_00_00_07    // F_1   F07
64_07_00_06    // FINC  F06,  F07
66_00_00_05    // F_0   F05
65_05_00_04    // FDEC  F04,  F05
45_1F_1F_1F    // HALT
```

```
@E0
//    Floating Point Data Storage (Little Endian)
//-----------------------------------------------
//
d70a3d71 403190a3   //  17.565
06f69446 c070e4df   // -270.30445
d7654321 c1234567   //  To be filled in
98765432 abcdef01   //  To be filled in
88000000 c18b0da9   // -56735025.0
1f000000 41bd705b   //  493902623.0
d7654321 c1234567   //  To be filled in
98765432 abcdef01   //  To be filled in
```

LOG FILE #13

Registers unaffected during program

```
t=14.0 ns R[00000000] -- 00000000000000ee R[00000008] -- xxxxxxxxxxxxxxxx
t=15.0 ns R[00000001] -- xxxxxxxxxxxxxxxx R[00000009] -- xxxxxxxxxxxxxxxx
t=16.0 ns R[00000002] -- xxxxxxxxxxxxxxxx R[0000000a] -- xxxxxxxxxxxxxxxx
t=17.0 ns R[00000003] -- xxxxxxxxxxxxxxxx R[0000000b] -- xxxxxxxxxxxxxxxx
t=18.0 ns R[00000004] -- xxxxxxxxxxxxxxxx R[0000000c] -- xxxxxxxxxxxxxxxx
t=19.0 ns R[00000005] -- xxxxxxxxxxxxxxxx R[0000000d] -- xxxxxxxxxxxxxxxx
t=20.0 ns R[00000006] -- xxxxxxxxxxxxxxxx R[0000000e] -- xxxxxxxxxxxxxxxx
t=21.0 ns R[00000007] -- xxxxxxxxxxxxxxxx R[0000000f] -- xxxxxxxxxxxxxxxx
t=22.0 ns FP[00000000] -- xxxxxxxxxxxxxxxx        (-nan)
t=23.0 ns FP[00000001] -- xxxxxxxxxxxxxxxx        (-nan)
t=24.0 ns FP[00000002] -- xxxxxxxxxxxxxxxx        (-nan)
t=25.0 ns FP[00000003] -- xxxxxxxxxxxxxxxx        (-nan)
t=26.0 ns FP[00000004] -- bff0000000000000        (-1.000000)
t=27.0 ns FP[00000005] -- 0000000000000000        (0.000000)
t=28.0 ns FP[00000006] -- 4000000000000000        (2.000000)
t=29.0 ns FP[00000007] -- 3ff0000000000000        (1.000000)
t=30.0 ns FP[00000008] -- 41c0690828000000        (550637648.000000)
t=31.0 ns FP[00000009] -- 41ba0ea5ee000000        (437167598.000000)
t=32.0 ns FP[0000000a] -- 41bd705b1f000000        (493902623.000000)
t=33.0 ns FP[0000000b] -- c18b0da988000000        (-56735025.000000)
t=34.0 ns FP[0000000c] -- c0b28be5cd53048a        (-4747.897664)
t=35.0 ns FP[0000000d] -- c02ec7121f31f3d1        (-15.388810)
t=36.0 ns FP[0000000e] -- c070e4df06f69446        (-270.304450)
t=37.0 ns FP[0000000f] -- 403190a3d70a3d71        (17.565000)
.
.
.
t=46.0 ns M[000000e0] -- d70a3d71   M[000000f0] -- xxxxxxxx
t=47.0 ns M[000000e1] -- 403190a3   M[000000f1] -- xxxxxxxx
t=48.0 ns M[000000e2] -- 06f69446   M[000000f2] -- xxxxxxxx
t=49.0 ns M[000000e3] -- c070e4df   M[000000f3] -- xxxxxxxx
t=50.0 ns M[000000e4] -- 1f31f3d1   M[000000f4] -- xxxxxxxx
t=51.0 ns M[000000e5] -- c02ec712   M[000000f5] -- xxxxxxxx
t=52.0 ns M[000000e6] -- cd53048a   M[000000f6] -- xxxxxxxx
t=53.0 ns M[000000e7] -- c0b28be5   M[000000f7] -- xxxxxxxx
t=54.0 ns M[000000e8] -- 88000000   M[000000f8] -- xxxxxxxx
t=55.0 ns M[000000e9] -- c18b0da9   M[000000f9] -- xxxxxxxx
t=56.0 ns M[000000ea] -- 1f000000   M[000000fa] -- xxxxxxxx
t=57.0 ns M[000000eb] -- 41bd705b   M[000000fb] -- xxxxxxxx
t=58.0 ns M[000000ec] -- ee000000   M[000000fc] -- xxxxxxxx
t=59.0 ns M[000000ed] -- 41ba0ea5   M[000000fd] -- xxxxxxxx
t=60.0 ns M[000000ee] -- 28000000   M[000000fe] -- xxxxxxxx
t=61.0 ns M[000000ef] -- 41c06908   M[000000ff] -- xxxxxxxx
```

DEC R5

F_0

INC R7

F_1

-270.304450
x 17.565000
-4747.897664

-270.304450 /
17.565000 =
-15.388810

Starting memory locations

MEMORY MODULE #14

```
@0
44_00_00_00    //01//          STI                          ;Set Intr. Enable Flag
07_00_00_0F    //02//          LDI   R15, E0h
00_00_00_E0    //03
00_00_00_00    //04
07_00_00_0E    //05//          LDI   R14, 0Fh
00_00_00_0F    //06
00_00_00_00    //07
0B_0F_0E_00    //08//          EXCH  R15, R14               ;swap R15, R14
07_00_00_0D    //09//          LDI   R13, 88888888_0A0A0Ah
0A_0A_0A_0A    //0A
88_88_88_88    //0B
41_00_00_00    //0C//          STC                          ;set carry
09_0D_00_0E    //0D//  c_loop: ST    [R14], R13             ;main loop to write
1A_0D_00_0D    //0E//          ASR   R13                    ;  16 patterns from
14_0E_00_0E    //0F//          INC   R14                    ;    0xE0 to 0xFF
14_0E_00_0E    //10//          INC   R14
15_0F_00_0F    //11//          DEC   R15
23_FF_FF_FA    //12//          JNZ   c_loop
07_00_00_0E    //13//          LDI   R14, FEh
00_00_00_FE    //14
00_00_00_00    //15
08_0E_00_00    //15//          LD    R0, [R14]
00_00_0E_01    //17//          ADD   R1, R0, R14
0A_01_00_02    //18//          COPY  R2, R1
45_00_00_00    //19//          HALT

@FE
89_AB_CD_EF
01_23_45_67
                               //**************
@100                           //  Actual ISR
                               //**************
07_00_00_0F    //              LDI   R15, 0Eh
00_00_00_0E    //
00_00_00_00    //
0D_0E_00_0F    //             OUT   [R15], R14
0C_0F_00_03    //             IN    R3, [R15]
07_00_00_04    //             LDI   R4, 02h
00_00_00_02    //
00_00_00_00    //
07_00_00_05    //             LDI   R5, 03h
00_00_00_03    //
00_00_00_00    //
07_00_00_06    //             LDI   R6, 04h
00_00_00_04    //
00_00_00_00    //
07_00_00_07    //             LDI   R7, 05h
00_00_00_05    //
00_00_00_00    //
00_03_04_08    //             ADD R8, R3, R4
00_05_06_09    //             ADD R9, R5, R6
00_06_07_0A    //             ADD R10, R6, R7
00_07_09_0B    //             ADD R11, R7, R9
00_0A_09_0C    //             ADD R12, R10, R9
35_00_00_00    //             RETI

@3FE            //Interrupt Vector
00_00_01_00    //ISR Address @100
00_00_00_00
```

R9 + R10 = R12
7h + 9h = 10h

R7 + R9 = R11
5h + 9h = Ch

R6 + R7 = R10
4h + 5h = 9h

R5 + R6 = R9
3h + 4h = 7h

LOG FILE #14

COPY R2, R1

R14 = FEh
FEh = 0123456789ABCDEFh
LD R0, [R14]

OUT [R15], R14
IN R3, [R15]
R14 was not set before being used

```
t=54.0 ns R[00000000] -- 0123456789abcdef R[00000008] -- xxxxxxxxxxxxxxxx
t=55.0 ns R[00000001] -- 0123456789abceed R[00000009] -- 0000000000000007
t=56.0 ns R[00000002] -- 0123456789abceed R[0000000a] -- 0000000000000009
t=57.0 ns R[00000003] -- xxxxxxxxxxxxxxxx R[0000000b] -- 000000000000000c
t=58.0 ns R[00000004] -- 0000000000000002 R[0000000c] -- 0000000000000010
t=59.0 ns R[00000005] -- 0000000000000003 R[0000000d] -- ffff111111101414
t=60.0 ns R[00000006] -- 0000000000000004 R[0000000e] -- 00000000000000fe
t=61.0 ns R[00000007] -- 0000000000000005 R[0000000f] -- 0000000000000000
.
.
.
.
.
.
.
.
```

Initial value to be Shifted Right Arithmetic Beginning of loop

```
t=86.0 ns M[000000e0] -- 0a0a0a0a  M[000000f0] -- 880a0a0a
t=87.0 ns M[000000e1] -- 88888888  M[000000f1] -- ff888888
t=88.0 ns M[000000e2] -- 05050505  M[000000f2] -- 44050505
t=89.0 ns M[000000e3] -- c4444444  M[000000f3] -- ffc44444
t=90.0 ns M[000000e4] -- 02828282  M[000000f4] -- 22028282
t=91.0 ns M[000000e5] -- e2222222  M[000000f5] -- ffe22222
t=92.0 ns M[000000e6] -- 01414141  M[000000f6] -- 11014141
t=93.0 ns M[000000e7] -- f1111111  M[000000f7] -- fff11111
t=94.0 ns M[000000e8] -- 80a0a0a0  M[000000f8] -- 8880a0a0
t=95.0 ns M[000000e9] -- f8888888  M[000000f9] -- fff88888
t=96.0 ns M[000000ea] -- 40505050  M[000000fa] -- 44405050
t=97.0 ns M[000000eb] -- fc444444  M[000000fb] -- fffc4444
t=98.0 ns M[000000ec] -- 20282828  M[000000fc] -- 22202828
t=99.0 ns M[000000ed] -- fe222222  M[000000fd] -- fffe2222
t=100.0 ns M[000000ee] -- 10141414 M[000000fe] -- 89abcdef
t=101.0 ns M[000000ef] -- ff111111 M[000000ff] -- 01234567
```

Initial value Shifted Right Arithmetically 15 times End of loop

Enhancement Memory Modules and Log Files

ENHANCEMENT MEMORY MODULE #1

```
@0
81_00_00_0F  //00  VLDI   R15,
000000E0000000E0h
00_00_00_E0  //01
00_00_00_E0  //02
81_00_00_0E  //03  VLDI   R14,
000000F0000000F0h
00_00_00_F0  //04
00_00_00_F0  //05
81_00_00_0D  //06  VLDI   R13,
0000000800000008
00_00_00_08  //07
00_00_00_08  //08
73_0D_00_0D  //09  VLSL   R13
73_0D_00_0D  //0A  VLSL   R13
81_00_00_03  //0C  VLDI    R3,
0000001800000018h
00_00_00_18
00_00_00_18
7D_03_00_04  //0D  VCOPY  R4,  R3
7D_04_00_05  //0E  VCOPY  R5,  R4
7D_05_00_06  //0F  VCOPY  R6,  R5
7D_06_00_07  //10  VCOPY  R7,  R6
81_00_00_0C  //20  VLDI  R12, 100h
00_00_01_00  //21
00_00_01_00  //22
81_00_00_00  //23  VLDI   R0,
0000001000000010h
00_00_00_10
00_00_00_10
81_00_00_01  //25  VLDI   R1,
000000ff000000FFh
00_00_00_FF
00_00_00_FF
70_0D_07_08  //26  VADDS R8, R7, R13
71_08_07_08  //27  VSUBS R8, R8, R13
77_00_01_02  //28  VXOR  R2,   R0, R1
7A_02_00_02  //2B  VNEG  R2
7E_0E_0C_00  //2E  VCMP  R14,  R12
45_00_00_00  //31  HALT   {Safety Net}

@E0
C3_C3_C3_C3
C3_C3_C3_C3
1E_1E_1E_1E
1E_1E_1E_1E
F1_F1_F1_F0
F1_F1_F1_F1
07_07_07_07
07_07_07_07
E3_E3_E3_E1
E3_E3_E3_E3
0E_0E_0E_0E
0E_0E_0E_0E
17_17_17_14
17_17_17_17
74_74_74_74
74_74_74_74
27_27_27_23
27_27_27_27
6C_6C_6C_6C
6C_6C_6C_6C
4B_4B_4B_4E
4B_4B_4B_4b
5A_5A_5A_5A
5A_5A_5A_5A
81_81_81_87
81_81_81_81
3F_3F_3F_3F
3F_3F_3F_3F
73_73_73_74
73_73_73_73
46_46_46_46
46_46_46_46
```

ENHANCEMENT LOG FILE #1

VCOPY R4, R3
Copies R3 into R4, R5, R6, R7

VLDI R15, 000000E0000000E0h
VLDI R14, 000000F0000000F0h

0x0000000800000008 << 2 =
0x0000002000000020

```
t=35.0 ns V[00000000] -- ffff0000ffff0000 V[00000008] -- 0000003800000038
t=36.0 ns V[00000001] -- 000000ff000000ff V[00000009] -- 0000002000000020
t=37.0 ns V[00000002] -- 0000fff00000fff0 V[0000000a] -- xxxxxxxxxxxxxxxx
t=38.0 ns V[00000003] -- 0000001800000018 V[0000000b] -- xxxxxxxxxxxxxxxx
t=39.0 ns V[00000004] -- 0000001800000018 V[0000000c] -- 0000010000000100
t=40.0 ns V[00000005] -- 0000001800000018 V[0000000d] -- 0000002000000020
t=41.0 ns V[00000006] -- 0000001800000018 V[0000000e] -- 000000f0000000f0
t=42.0 ns V[00000007] -- 0000001800000018 V[0000000f] -- 000000e0000000e0
t=43.0 ns M[000000e0] -- c3c3c3c3  M[000000f0] -- 27272723
t=44.0 ns M[000000e1] -- c3c3c3c3  M[000000f1] -- 27272727
t=45.0 ns M[000000e2] -- 1e1e1e1e  M[000000f2] -- 6c6c6c6c
t=46.0 ns M[000000e3] -- 1e1e1e1e  M[000000f3] -- 6c6c6c6c
t=47.0 ns M[000000e4] -- f1f1f1f0  M[000000f4] -- 4b4b4b4e
t=48.0 ns M[000000e5] -- f1f1f1f1  M[000000f5] -- 4b4b4b4b
t=49.0 ns M[000000e6] -- 07070707  M[000000f6] -- 5a5a5a5a
t=50.0 ns M[000000e7] -- 07070707  M[000000f7] -- 5a5a5a5a
t=51.0 ns M[000000e8] -- e3e3e3e1  M[000000f8] -- 81818187
t=52.0 ns M[000000e9] -- e3e3e3e3  M[000000f9] -- 81818181
t=53.0 ns M[000000ea] -- 0e0e0e0e  M[000000fa] -- 3f3f3f3f
t=54.0 ns M[000000eb] -- 0e0e0e0e  M[000000fb] -- 3f3f3f3f
t=55.0 ns M[000000ec] -- 17171714  M[000000fc] -- 73737374
t=56.0 ns M[000000ed] -- 17171717  M[000000fd] -- 73737373
t=57.0 ns M[000000ee] -- 74747474  M[000000fe] -- 46464646
t=58.0 ns M[000000ef] -- 74747474  M[000000ff] -- 46464646
```

ENHANCEMENT MEMORY MODULE #2

```
@0
92_00_00_00 //Set Zero
07_00_00_01 //LD R1, 10h
00_00_00_10
00_00_00_00
93_00_00_00 //Clear Zero
94_01_00_00 //DJNZ R1 - 1
92_00_00_00 //Set Zero
95_01_00_00 //DJZ R1 - 1
99_00_00_00 //CLEAR
45_00_00_00 //HALT
```

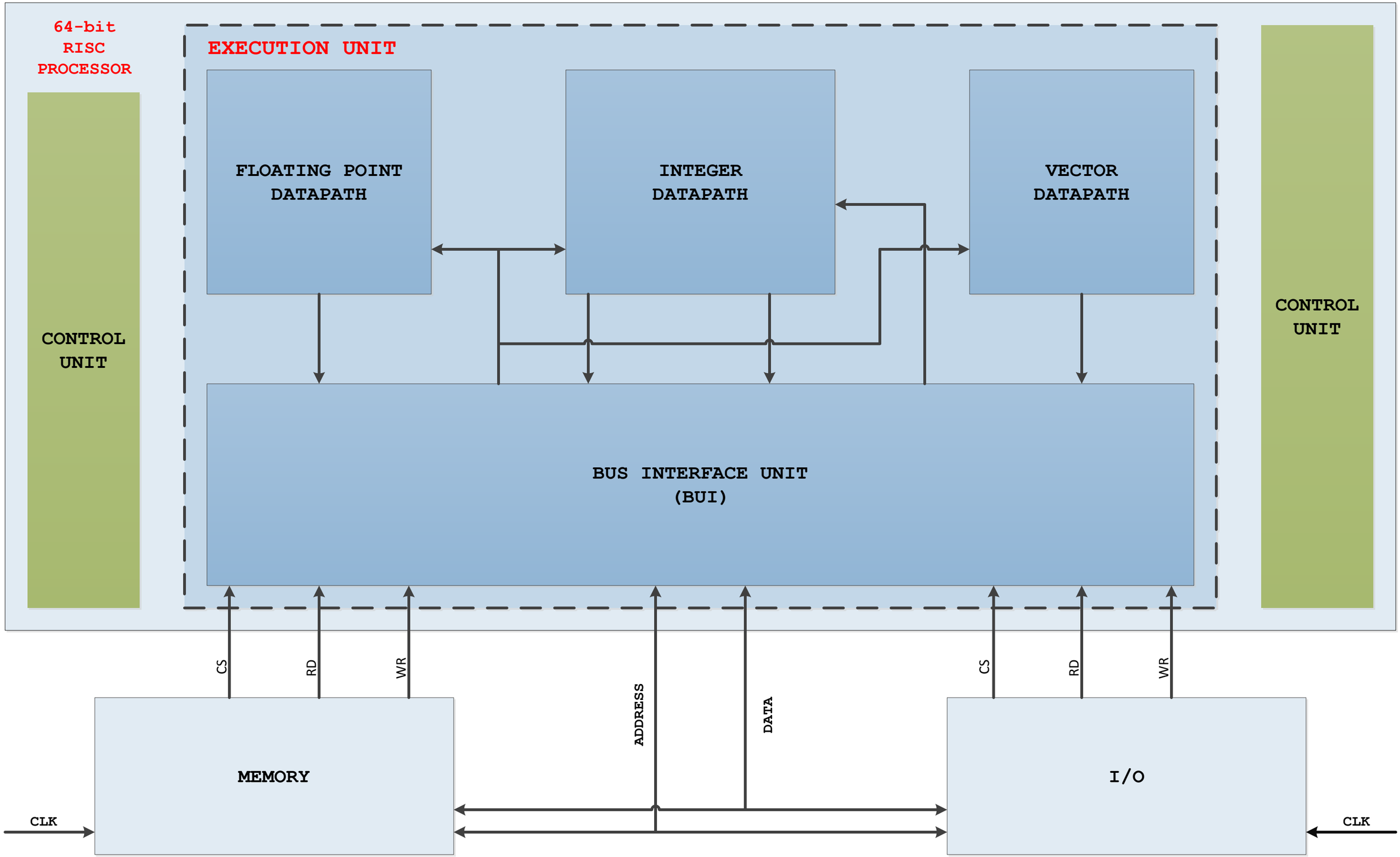
```
@E0
C3_C3_C3_C3
C3_C3_C3_C3
1E_1E_1E_1E
1E_1E_1E_1E
F1_F1_F1_F0
F1_F1_F1_F1
07_07_07_07
07_07_07_07
E3_E3_E3_E1
E3_E3_E3_E3
0E_0E_0E_0E
0E_0E_0E_0E
17_17_17_14
17_17_17_17
74_74_74_74
74_74_74_74
27_27_27_23
27_27_27_27
6C_6C_6C_6C
6C_6C_6C_6C
4B_4B_4B_4E
4B_4B_4B_4b
5A_5A_5A_5A
5A_5A_5A_5A
81_81_81_87
81_81_81_81
3F_3F_3F_3F
3F_3F_3F_3F
73_73_73_74
73_73_73_73
46_46_46_46
46_46_46_46
```

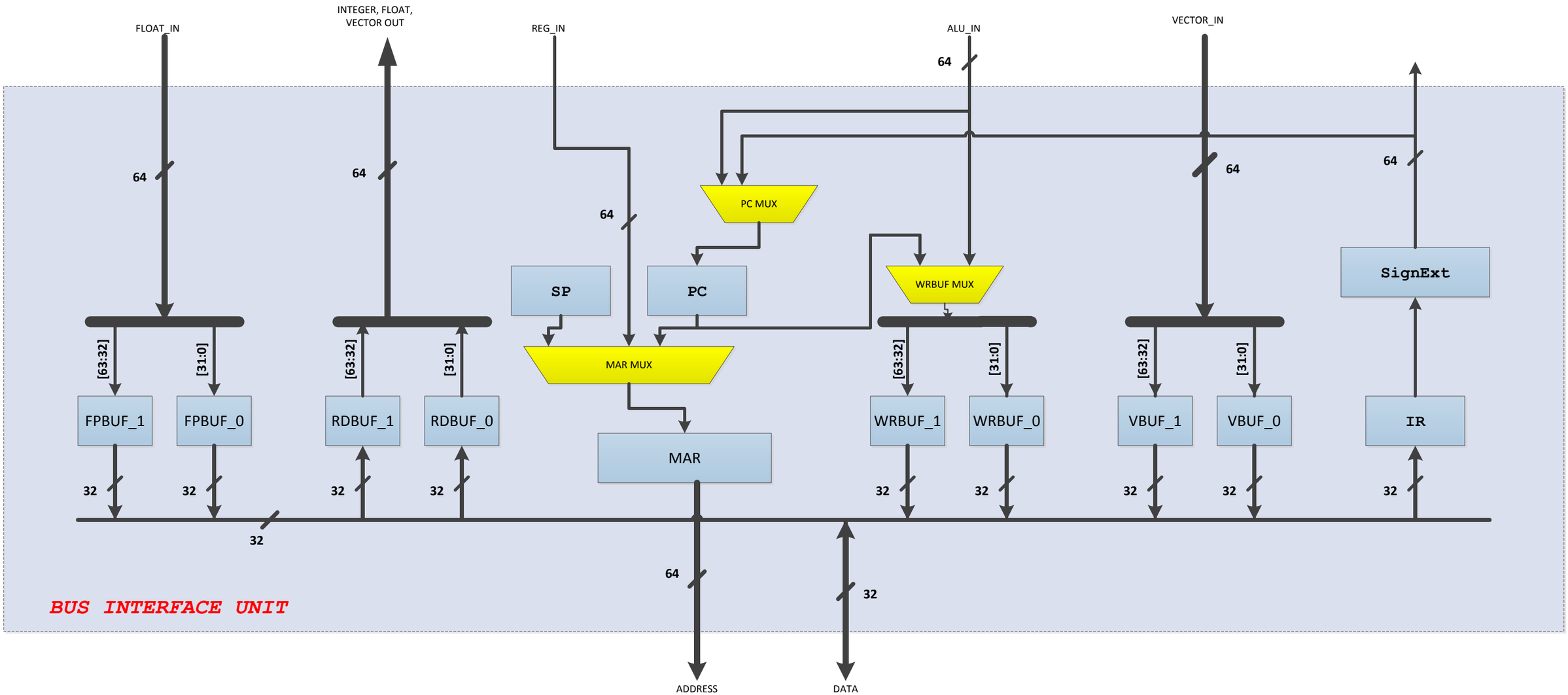
ENHANCEMENT LOG FILE #2

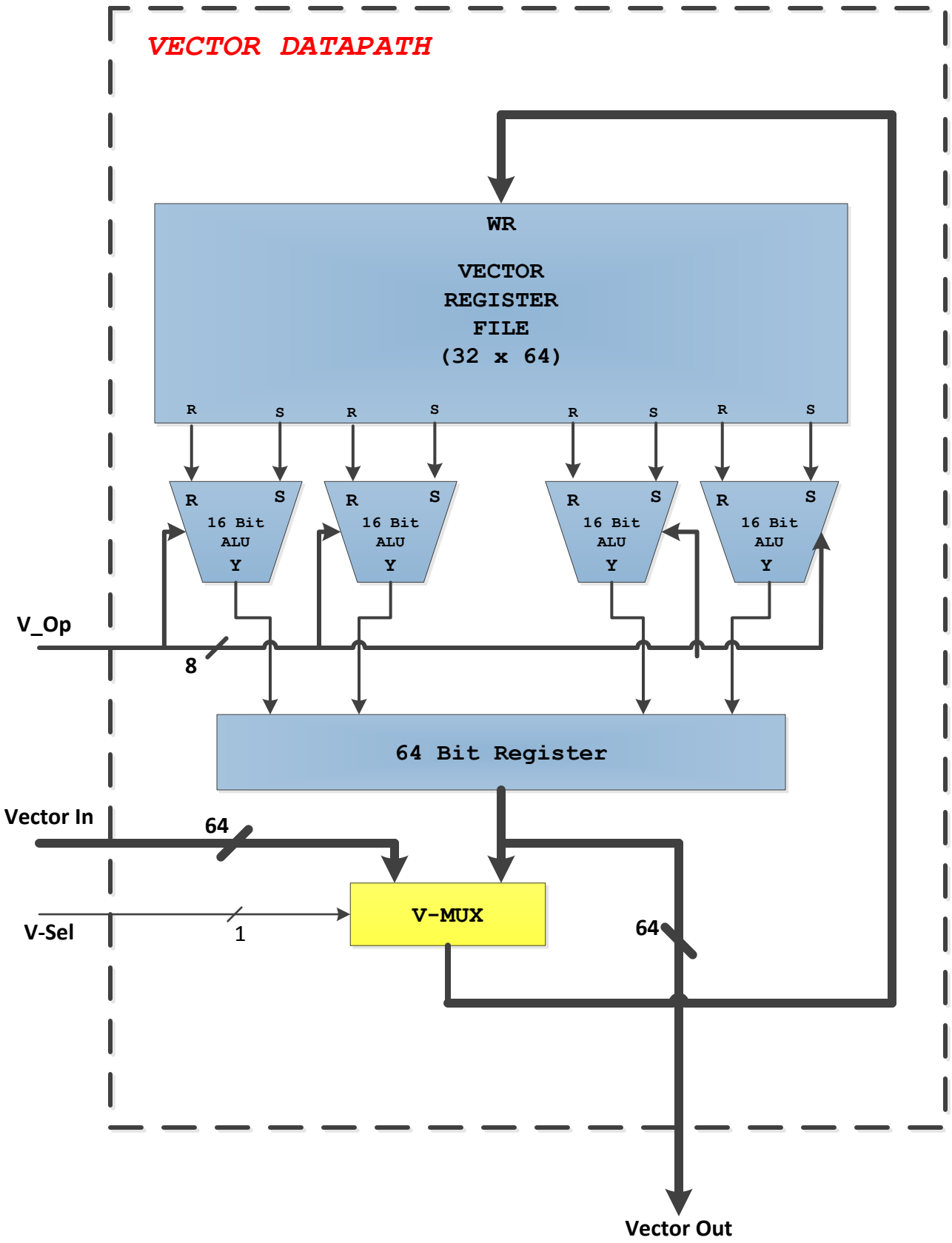
```
INT_CHK_F: T=0.0 ns, IR=xx INZOC=00000
FETCH: T=0.0 ns, IR=xx INZOC=00000
      FP_FLAGS=000000
DECODE: T=1.0 ns, IR=92 INZOC=00000
SZ: T=1.0 ns, IR=92 INZOC=00000

INT_CHK_F: T=1.0 ns, IR=92 INZOC=00100
FETCH: T=1.0 ns, IR=92 INZOC=00100
      FP_FLAGS=000000
DECODE: T=1.0 ns, IR=07 INZOC=00100
LOAD_imm1: T=1.0 ns, IR=07 INZOC=00100
LOAD_imm2: T=1.0 ns, IR=07 INZOC=00100
LOAD_imm3: T=1.0 ns, IR=07 INZOC=00100
LOAD_imm4: T=1.0 ns, IR=07 INZOC=00100

INT_CHK_F: T=1.0 ns, IR=07 INZOC=00100
FETCH: T=2.0 ns, IR=07 INZOC=00100
      FP_FLAGS=000000
DECODE: T=2.0 ns, IR=93 INZOC=00100
CZ: T=2.0 ns, IR=93 INZOC=00100

INT_CHK_F: T=2.0 ns, IR=93 INZOC=00000
FETCH: T=2.0 ns, IR=93 INZOC=00000
      FP_FLAGS=000000
```

SZ = Set Zero Flag

CZ = Clear Zero Flag


64-bit
RISC
PROCESSOR
EXECUTION UNIT
FLOATING POINT
DATAPATH
INTEGER
DATAPATH
VECTOR
DATAPATH
CONTROL
UNIT
CONTROL
UNIT
BUS INTERFACE UNIT
(BUI)
CS
RD
WR
ADDRESS
DATA
CS
RD
WR
MEMORY
I/O
CLK
CLK


FLOAT_IN
INTEGER, FLOAT, VECTOR OUT
REG_IN
ALU_IN
VECTOR_IN
64
PC MUX
WRBUF MUX
SP
PC
SignExt
MAR MUX
[63:32]
[31:0]
FPBUF_1
FPBUF_0
RDBUF_1
RDBUF_0
WRBUF_1
WRBUF_0
VBUF_1
VBUF_0
IR
32
MAR
BUS INTERFACE UNIT
ADDRESS
DATA


VECTOR DATAPATH
WR
VECTOR
REGISTER
FILE
(32 x 64)
R
S
R
S
R
S
R
S
R
S
16 Bit
ALU
Y
R
S
16 Bit
ALU
Y
R
S
16 Bit
ALU
Y
R
S
16 Bit
ALU
Y
V_Op
8
64 Bit Register
Vector In
64
V-Sel
1
V-MUX
64
Vector Out