Register-type

Takes two register addresses as input, performs a computation, stores the result in the first register address. Can be used on any pair of registers.

R-Type	Mnemonic	Opcode [8:6]	Source-Destination Register [5:3]	Source Register [2:0]	Notes
Add	Add	000	xxx	xxx	Add r0 r1 adds the contents of r0 to r1 and stores the result in r0.
Xor	Xor	001	xxx	xxx	Xor r0 r1 takes the XOR of r0 and r1 and stores the result in r0.
And	And	010	xxx	xxx	And r0 r1 takes the AND of r0 and r1 and stores the result in r0.

Memory-type

Takes a register address containing a memory address to be written to or read from and a truncated register address for the data source or destination. Truncated addresses are supplemented with a 0 in the MSB; only the lower four registers are valid data registers.

M-Type	Mnemonic	Opcode [8:6]	Truncated Data Register [5:4]	Opcode Extension [3]	Address Register [2:0]	Notes
Load	Load	011	0xx	0	xxx	Takes the address register first. To load r1 with the contents of M[r5], use Load r5 r1.
Store	Store	011	0xx	1	xxx	Takes the address register first. To store r1 to M[r5], use Store r5 r1.

Immediate-type

Takes the least significant two bits of a register address and an immediate as input, assumes a MSB of 1 for the register address and adds the signed immediate to that register; only the upper four registers are valid source-destination registers.

I-Type	Mnemonic	Opcode [8:6]	Truncated Source-Dest Register [5:4]	Signed Immediate [3:0]
Add immediate	AddI	100	1xx	xxxx [-8, 7]

Overload-type

Takes a register and a three-bit mode signal. Can address any register.

O-Type		Mnemonic	Opcode [8:6]	Operand Register [5:3]	Mode 2 [2]	Mode 1 [1]	Mode 0 [0]	Notes	Usage
Shift	Left Shift	LShift rX	101	xxx	0	0	0	Sets the shift-carry flag.	LShift r0 shifts the contents of r0 left by one bit. Sets the shift-carry flag.
				xxx	0	0	1		
	Right Shift	RShift rX		xxx	0	1	0	Sets the shift-carry flag.	RShift r0 shifts the contents of r0 right by one bit. Sets the shift-carry flag.
	Rotate	Rotate rX		xxx	0	1	1		
	Left Shift, Carry In	DPLSh rX		xxx	1	0	0	Reads and "rotates in" the shift-carry flag.	DPLSh r0 shifts r0 left by one bit and places the value of the shift-carry flag in the LSB.
				xxx	1	0	1		
	Right Shift, Carry In	DPRSh rX		xxx	1	1	0	Reads and "rotates in" the shift-carry flag.	DPLSh r0 shifts r0 left by one bit and places the value of the shift-carry flag in the LSB.
				xxx	1	1	1		
Branch	Branch if zero; relative, register	Branch IfZeroReg rX	110	xxx	0	0	0		If the zero flag is set, the value of the given register is added to the PC.
	Branch if zero; absolute, immediate	Branch IfZeroAbs X		xxx (Immediate [0, 7])	0	0	1		If the zero flag is set, the PC is set to the value of the given register address.
	Branch if negative; relative, register	Branch IfNegativeReg rX		xxx	0	1	0		If the negative flag is set, the value of the given register is added to the PC.
	Branch if negative; absolute, immediate	Branch IfNegativeAbs X		xxx (Immediate [0, 7])	0	1	1		If the negative flag is set, the PC is set to the value of the given register address.
	Branch if not zero; relative, register	Branch IfNotZeroReg rX		xxx	1	0	0		If the zero flag is not set, the value of the given register is added to the PC.
	Branch if not zero; absolute, immediate	Branch IfNotZeroAbs X		xxx (Immediate [0, 7])	1	0	1		If the zero flag is not set, the PC is set to the value of the given register address.
	Branch if not negative; relative, register	Branch IfNotNegativeReg rX		xxx	1	1	0		If the negative flag is not set, the value of the given register is added to the PC.
	Branch if not negative; absolute, immediate	Branch IfNotNegativeAbs X		xxx (Immediate [0, 7])	1	1	1		If the negative flag is not set, the PC is set to the value of the given register address.
Parity	Parity: calculate p0	Parity CalcP0 rX	111	xxx	0	0	0	Sets operand register to [0000 000(p0)]	Expects registers 0 and 1 to be in format: r0:[(b4)(b3)(b2)(p4) (p2)(b1)(p1)x] r1: [(b11)(b10)(b9)(b8) (b7)(b6)(b5)(p8)]
	Parity: calculate p1	Parity CalcP1 rX		xxx	0	0	1	Sets operand register to [0000 00(p1)0]	Expects registers 0 and 1 to be in the format: r0:[(b8)(b7)(b6)(b5) (b4)(b3)(b2)(b1)] r1:[0000 0(b11)(b10)(b09)]
	Parity: calculate p2	Parity CalcP2 rX		xxx	0	1	0	Sets operand register to [0000 0(p2)00]	Expects registers 0 and 1 to be in the format: r0:[(b8)(b7)(b6)(b5) (b4)(b3)(b2)(b1)] r1:[0000 0(b11)(b10)(b09)]
	Parity: calculate p4	Parity CalcP4 rX		xxx	0	1	1	Sets operand register to [000(p4) 0000]	Expects registers 0 and 1 to be in the format: r0:[(b8)(b7)(b6)(b5) (b4)(b3)(b2)(b1)] r1:[0000 0(b11)(b10)(b09)]
	Parity: calculate p8	Parity CalcP8 rX		xxx	1	0	0	Sets operand register to [0000 000(p8)]	Expects registers 0 and 1 to be in the format: r0:[(b8)(b7)(b6)(b5) (b4)(b3)(b2)(b1)] r1:[0000 0(b11)(b10)(b09)]

	Parity: package the LSW	Parity PackLSW rX		xxx	1	0	1	Sets operand register to [(b4)(b3)(b2)0 (b1)000]	Expects registers 0 and 1 to be in the format: r0:[(b8)(b7)(b6)(b5) (b4)(b3)(b2)(b1)] r1:[0000 0(b11)(b10)(b09)]
	Parity: package the MSW	Parity PackMSW rX		xxx	1	1	0	Sets operand register to [(b11)(b10)(b9)(b8) (b7)(b6)(b5)0]	Expects registers 0 and 1 to be in the format: r0:[(b8)(b7)(b6)(b5) (b4)(b3)(b2)(b1)] r1:[0000 0(b11)(b10)(b09)]
	Parity: unpack the LSW	Parity UnpackLSW rX		xxx	1	1	1	Sets operand register to [(b4)(b3)(b2)(b1) (p4)(p2)(p1)(p0)]	Expects register 0 to be in the format: r0:[(b4)(b3)(b2)(p4) (p2)(b1)(p1)(p0)]

Register properties

The following is a list of the special properties of each register. Long-immediate addressable registers can be addressed by Add-Immediate instructions; loadable-storable registers can be used as the source or destination for data in Load and Store instructions.

Register	Long-immediate addressable?	Loadable-storable?	Special
r0	No	Yes	The LSW of the current parity data element
r1	No	Yes	The MSW of the current parity data element
r2	No	Yes	
r3	No	Yes	
r4	Yes	No	
r5	Yes	No	
r6	Yes	No	
r7	Yes	No	

Aliasing

The Avernus assembler supports translation of more human-readable register names into machine-friendly names with preprocessor directives that must be placed contiguously, one per line, at the start of the assembly language file. These lines take the format "Alias rX name" where rX is some register name, such as r3 or r7 and "name" is the name that should be replaced with "rX" in the code. Note that it is possible for registers to have multiple aliases. The preprocessor's behaviour is undefined in the event that two registers are aliased to the same name.

Macros

The following are the macros supported by the Avernus assembler and their resolved forms.

Mnemonic	Translation	Restrictions
Zero rX	Xor rX rX	
Set rX n	Xor rX rX AddI rX #n	rX must be long-immediate addressable
Copy rX rY	Xor rX rX Add rX rY	
Swap rX rY	Xor X Y Xor Y X Xor X Y	
RShift rX n	RShift rX *(repeated n times)*	
LShift rX n	LShift rX *(repeated n times)*	
Rotate rX n	Rotate rX *(repeated n times)*	
DoubleRShift rX rY n	RShift rX DPRSh rY *(repeated n times)*	
DoubleLShift rX rY n	LShift rX DPLSh rY *(repeated n times)*	

The termination instruction

The top-level module sets the "done" signal based on the instruction 101111111. This machine code falls into the "Shift" opcode space, but does not encode an actual instruction. Note that the opcode 101 indicates a Mode-type instruction, though the instruction with opcode 101 and mode 111 is undefined. The instruction 101111111 must be present at the end of every program run on an Avernus processor.
There is no Avernus assembly statement for 101111111. The assembler automatically adds the instruction onto the end of any program it assembles without the need for the programmer to add it manually.

ALU Operations

The following are the AluOp codes and their meanings.

0000	Add
0001	XOR
0010	AND

0000	Add
0001	XOR
0011	Left Shift
0100	Right Shift
0101	Right Rotate
0110	Double-Precision Left Shift
0111	Double-Precision Right Shift
1000	p0
1001	p1
1010	p2
1011	p3
1100	p4
1101	Package MSW
1110	Package LSW
1111	Unpack LSW

Status flags

Avernus supports three status flags: Zero, Negative, and Shift-Carry. Zero and Negative are set based on the ALU output yielded by every instruction on a continuous basis. Zero and Negative are used by Branch to determine whether the branch is taken. The Shift-Carry flag is set to the value last "shifted out" of a register by either RShift or LShift. The Shift-Carry flag is "shifted in" when DPLSh and DPRSh are called.