The Javelin Computer System

By J.P.Farr, © HobbeyElectronics.net, 2026

Version 2.0 - August 2026

Introduction

The Javelin computer is a small multi-processor system. The CPU's are custom designed and will be implemented using different technology. The initial CPU(s) will be PIC processors programmed with firmware running at 64MHz. Later, I will port them to FPGA and see if I can increase the clock speed. The system has a 16-bit data bus and 24-bit address bus, will support a maximum of 16M-Words of RAM/ROM and 16M-Words of I/O address space.

The computer will reside within a standard 19" Euro card frame with a parallel backplane - all backplane sockets are wired in parallel so Javelin boards can be placed in any slots.

The allocated rack has a standard PC style PSU that is capable of producing +5v at 60 Amps, and secondary voltages of +12v, -12v and +3.3v if required. Any of one of these voltages can be connected to the AUXV rail if required. The rack is currently home to my 6502-development system, but for now, that system will be removed in its entirety. With the PSU, the rack has a total of 1 required power management slot, and 9 expansion slots available.

The plan is to eventually install 4 x CPU boards, 2 x Memory boards giving maximum memory, 1 x mass storage device (hopefully supporting at least 2 Logical Units); probably an SD card with 1 to 4Gb of storage each, and a multi-port serial board with 4 serial channels. I'm thinking of using either USB flash drives or maybe small custom EEPROM cards for back-up storage.

The hardware will be controlled and managed by the FAROS multi-processor, real-time operating system (once I've written it).

This will be my chance to write and develop a full system based on a custom CPU with custom instruction set. Starting with assembler and minimum software to get one CPU up and running, then expanding the system to support additional hardware. Eventually compiled BASIC language support will be added.

Many of the CPU instruction set features and ideas have been borrowed from the Nixdorf 8870 (1559-CPU) and from the Zilog Z80.

This is of course, a work in progress and subject to extensive change at any time, especially when I realise, I've messed something up.

Master/Slave Bus

The system will work on master(s) / slaves(s) approach. CPU's will have to request access to the backplane when wanted to access shared resources. Expansion cards raise an interrupt when they wish to communicate with a CPU. When the operating system becomes available, the CPU(s) will each be capable of running multiple programs, and these programs will service the interrupts.

At a later stage, I may implement some type of private communication channel for the CPUs to communicate with each other and not tie the backplane.

Edgeway Connector Pinout

B/C				A		
+5v		33		1		+5v
D15	I/O	34		2	IN	A23
D14	I/O	35		3	IN	A22
D13	I/O	36		4	IN	A21
D12	I/O	37		5	IN	A20
D11	I/O	38		6	IN	A19
D10	I/O	39		7	IN	A18
D9	I/O	40		8	IN	A17
D8	I/O	41		9	IN	A16
D7	I/O	42		10	IN	A15
D6	I/O	43		11	IN	A14
D5	I/O	44		12	IN	A13
D4	I/O	45		13	IN	A12
D3	I/O	46		14	IN	A11
D2	I/O	47		15	IN	A10
D1	I/O	48		16	IN	A9
D0	I/O	49		17	IN	A8
(spare)		50		18	IN	A7
/INSSTART	IN	51		19	IN	A6
/ACK	OUT	52		20	IN	A5
/NMI	OUT	53		21	IN	A4
/IRQ	OUT	54		22	IN	A3
/IRQPRI0	OUT	55		23	IN	A2
/IRQPRI1	OUT	56		24	IN	A1
/IRQPRI2	OUT	57		25	IN	A0
/INTACK	IN	58		26	IN	/RD
BUS_CPU0	IN	59		27	IN	/WR
BUS_CPU1	IN	60		28	OUT	/MREQ
/BUS_BUSY		61		29	OUT	/IOREQ
/SYS_RESET	IN	62		30	OUT	/HALT
AUXV		63		31	IN	/CPU_HALT
0v		64		32		0v

The signal direction is based on any expansion card with the exception of a CPU card. An IN signal is accepted by the expansion card, and an OUT is a signal that is generated by the expansion card.

Control Signals

/BUSBUSY Indicates that the backplane is free (HIGH) and not being used by any CPU (LOW)

A floating signal that is pulled high via a resistor.

Indicates whether the system bus is currently available. The method by which competing bus masters (CPUs) obtain ownership is defined separately by the bus arbitration protocol. This signal should not be used on its own as an indication that the bus is free and can be obtained.

/INSSTART Indicates the start of an instruction read from memory.

/INSSTART is asserted by the CPU immediately before beginning execution of a new instruction and remains asserted until the first instruction-word memory access has commenced.

This signal is only generated by CPU #0

/ACK Device acknowledgement of read, write or HALT operation

When a memory or I/O mapped device is accessed, the CPU will wait for the /ACK signal to be pulled LOW. Once the CPU sees this, it knows that the device has accepted the read/write operation and the CPU then de-asserts the active /RD or /WR thus releasing the device. With /RD and /WR now high, the /ACK is released by the device. When the CPU sees the /ACK released, the CPU releases the /BUS_BUSY signal. When /CPUHALT is asserted as a resumable internal halt and /HALT is not asserted, a HIGH-to-LOW followed by a LOW-to-HIGH /ACK handshake instructs the processor to resume execution. /ACK has no resume function for critical halt codes.

/RESET Reset the entire system

Any device or CPU can pull the /RESET signal low. This will cause all devices and CPUs that monitor the /RESET signal restart.

/HALT HALT the CPU

Pulling this signal low halts the CPU after the current instruction has executed and before the start of the next instruction. When the CPU has halted the /CPUHALT is pulled low by the CPU.

It's possible to connect the /INSSTART signal to the trigger on a latch, and connect the latch output to the /HALT signal. This would cause the CPU to halt at the start of each instruction. If a "STEP" button is connected to the latch reset signal, the code being executed by the CPU can be single stepped.

/CPUHALT The CPU has halted

This signal indicates that the CPU has halted execution. If the /HALT signal is also asserted when this signal is raised, it indicates that an external device has requested the CPU to temporarily halt. If the /HALT signal is not asserted however, this indicates that the CPU has internally detected a problem and has halted. The CPU must be RESET now to continue. In this second instance, the memory address associated with the fault is output onto A0 - A23, and the data bus contains an error code indicating the reason why the CPU halted.

Any data bus value of 255 or less is categorised by the CPU as a "resumable" halt. By pulling low the /ACK and then releasing it, the CPU will continue on from where it halted. These could be raised by the application software for debugging or other purposes. However, any halt code specified greater than 255 on the data bus is classed as a critical system halt and cannot be recovered from without a system RESET.

Notes:

Will add an assembler TRAP X instruction that can raise a 16-bit HALT event, however, when higher level languages are developed, they will only be able to generate resumable halts (0 to 255). If there is a need for external hardware to raise a critical fault, it can

use one of the interrupt channels and a value on the data bus.

/IOREQ Request I/O access

This signal from the CPU indicates that it's wanting to access an I/O device at the address placed on the 24-bit address bus. Data can then be read/written to this device address via the /RD and /WR signals and the data bus.

For a write operation, the data to be written is loaded onto the data bus

The CPU will publish the I/O address and then wait for the /ACK signal to be set by the device. Once set, the CPU will read the data from the data bus if this is a read operation. In either case, the CPU will clear the /RD and /WR signals thus releasing the device.

/MREQ Request memory access

This signal from the CPU indicates that it's wanting to access memory at the address placed on the 24-bit address bus. Data can then be read/written to this memory address via the /RD and /WR signals and the data bus.

For a write operation, the data to be written is loaded onto the data bus

The CPU will publish the memory address and then wait for the /ACK signal to be set by the device. Once set, the CPU will read the data from the data bus if this is a read operation. In either case, the CPU will clear the /RD and /WR signals thus releasing the device.

Timings for accessing the backplane

Write To Memory

WAIT until ownership of the bus confirmed.

CONFIRM /ACK = HIGH

WRITE Address onto address bus

WRITE Data onto data bus

ASSERT Byte enables

ASSERT /MREQ + /WR

WAIT for /ACK = LOW

DEASSERT /MREQ + /WR = HIGH

WAIT for /ACK = HIGH

RESLEASE ownership of bus

End

TO BE COMPLETED

Conventions

There are three sizes of register, register part, memory value or constant.

24-Bit - These have 3 parts consisting of High / Middle / Low (H/M/L)

16-Bit - These have 2 parts consisting of High / Low (H/L)

8-Bit - These have 1 part consisting of Low (L)

A Nybble = 4-bits

A byte = 8-bits

A word = 16-bits

Any number beginning with a "$" is a Hexadecimal value. Any other number unless specified, is decimal.

Since the system has a a 16-bit data bus and memory width, where ever "K" is mention in regards to memory size, this refers to K-Words.

Memory Map

On power-up or system reset, the PC (Program Counter) of all CPU's is set to $008000 and the IPL on CPU#0 starts. All other CPUs wait for the IPL to complete.

Start Address	End Address	Size	Usage	Type	Usage
$000000	$0007FF	2K	SYS	RAM	System Global Configuration Area
$000800	$001FFF	6K	SYS	RAM	Op-Sys RAM Area
$002000	$003FFF	8K	SYS	RAM	Op-Sys RAM Area
$004000	$005FFF	8K	SYS	RAM	Op-Sys RAM Area
$006000	$007FFF	8K	SYS	RAM	Op-Sys RAM Area
$008000	$00FFFF	32K	ANY	ROM	System sub-routines and boot code
$010000	$FFFFFF	16M	PARTs	RAM	Partition Storage Area

Note: All sizes are in Words as memory is 16-bits/2 bytes/1 word, wide

Memory is dynamically allocated and depends on the number of CPUs configured in the system, the number of partitions and the configuration settings specified.

System Global Configuration Area

Start Address	End Address	Size	Usage
$000000	$000000	1	Number of configured CPUs (1 to 4)
		2	Start address of CPU#0 Configuration area (CCA)
		2	Start address of CPU#1 Configuration area (CCA)
		2	Start address of CPU#2 Configuration area (CCA)
		2	Start address of CPU#3 Configuration area (CCA)

CCA - CPU Configuration Area

Relative Offset	Size	Usage
$000	2	Start Address of PCT (Partition Control Table)
$002	1	IRQ scan rate delay (ms)
$003	1	THZ interval (ms)
$004	1	Number of partitions (1 to 31 per CPU)
		Size of partition 1 (K-Words)
		Size of partition 2 (K-Words)
		Size of partition 3 (K-Words)
		" " "
		Size of partition 31 (K-Words)

PCT - Partition Control Table

The PCT is dynamic and automatically built during the CPU IPL phase.

Relative Offset	Size (Words)	Usage
$000	1	Partition Size (K-Words)
$001	1	Source File Logical Unit number (LU)
$002	20	Source File Name (40 characters max)
$016		

Registers

Register Index	Name	Mode	
00	CV8	READ	8-bit constant value (Virtual Register)
01	CV16	READ	16-bit constant value (Virtual Register)
02	CV24	READ	24-bit constant value (Virtual Register)
03	FLAGS	R/W*	24-bit FLAGS register
04	MADDR	R/W	24-bit Memory Address (See note on accessing memory map)
05	MD	R/W	Memory Data - Direct Data-relative access 16-bits
06	MI	R/W	Memory Data - Indirect Data-relative access 16-bits
07	IX	R/W	24-bit Index register
08	R0	R/W	24-bit General Purpose Register
09	R1	R/W	24-bit General Purpose Register
0A	R2	R/W	24-bit General Purpose Register
0B	R3	R/W	24-bit General Purpose Register
0C	R4	R/W	24-bit General Purpose Register
0D	R5	R/W	24-bit General Purpose Register
0E	R6	R/W	24-bit General Purpose Register
0F	R7	R/W	24-bit General Purpose Register

Memory Addressing

All addresses visible to a program are relative addresses. A program cannot directly access or store to a physical 24-bit memory address.

Note: Depending on the FLAGS.PROTECT bit, MADDR has two modes of operation. With the FLAGS.PROTECT = 1 (protection is in force), then MADDR contains a 24-bit offset relative to the current context's DBASE.

However, when FLAGS.PROTECT = 0 (no protection in force) then MADDR contains a physical memory address.

For direct memory access (MD), the CPU calculates the physical effective address as:

EA = DBASE + MADDR + IX

where IX is included only when indexed addressing is selected.

For indirect memory access (MI), the CPU first reads a 24-bit relative pointer from:

PA = DBASE + MADDR

The value read from this location is itself treated as a relative address. The final effective address is therefore:

EA = DBASE + MEMORY24[PA] + IX

where IX is again optional.

Because addresses stored by programs are relative rather than physical, a complete program and its associated data can be relocated in physical memory without modifying pointers stored within its data structures, even when the program is currently executing.

A system TRAP will occur if a program attempts to access memory that is protected.

FLAGS Register

Bit	Name	Mode	Usage
23	PART4	READ	Current partition number, 1-31. Zero indicates no valid partition.
22	PART3	READ	
21	PART2	READ	
20	PART1	READ	
19	PART0	READ	
18	FLAG0	R/W	User defined flags. These are just temporary storage locations.
17	FLAG1	R/W	
16	FLAG2	R/W	
15	C	READ	Carry
14	IE	R/W	Interrupts Enabled for this partition
13	IBUSY	READ	An interrupt is in progress. Waiting for IRET to clear
12	PROTECT	R/W	This defaults to 1 and prevents memory outside of the current partition from being accessed. Auto set = 1 after memory access.
11	PTYPE0	READ	These two bits specify the category of the program and control its system resource access rights.
10	PTYPE1	READ	
9	-		
8	ALT	R/W	Using the alternate mirror set of registers (excluding the FLAGS register)
7	INT7	READ	These bits are read-only, setup by the loader and indicate if these interrupts are serviced by this program. An interrupt cannot be serviced by multiple executing programs. A TRAP message is issued if an attempt is made to break this rule.
6	INT6	READ	
5	INT5	READ	
4	INT4	READ	
3	INT3	READ	
2	INT2	READ	
1	INT1	READ	
0	INT0	READ	

Note: Certain bits in the FLAGS register are read-only and cannot be updated by the program.

PART0 to PART4 - Current Partition Number

This is the current partition number. It's assigned by the loader when the program is loaded into memory points into the PCT (partition control table)

Partition 0 indicates empty/undefined partition. This means that Partition 0 should never be scheduled, swapped, or own interrupts.

FLAG0 to FLAG2 - Storage Flags

These flags mean nothing to the CPU but are a convenient place to store bits of information. They are especially useful when switching to and from the Alternative set of registers as there is only the one FLAGS register, so these flag bits are available irrespective of which set of registers is selected.

C - Carry Flag

IE - Interrupts Enabled for this partition

Interrupts can be enabled/disabled for the current partition. When a partition is loaded with a new program, the FLAGS.IE is cleared indicating interrupts for this program are disabled. The program must enable them when ready.

IBUSY - Interrupt busy

Indicates that one of the interrupts managed by this partition is currently running. The FLAGS.IBUSY is set as soon as the interrupt routine is called, and then cleared when IRET is executed.

PTYPE0 to PTYPE1 - Program Type

This information is typically populated from the program file header and indicates the type of program that is currently active.

ALT - Alternate register set

To enable faster interrupt processing within a program, it's possible to switch to the alternate mirror set of registers. These registers are separate from the main register set and the CPU can alternate between the two sets as required. The FLAGS register DOES NOT have an alternate.

INT7 to INT0 - Interrupt serviced bits

These 8-bits indicate which, if any, of the 8 available interrupts are being serviced by this program. The bits are set up by reading information from the program file header. If an interrupt is not enabled for this program, the program will not receive interrupt notifications for that interrupt. However, a program MUST service the interrupt if it's configured to do so.

Note:

You will notice there is no Zero flag. This is because typically you store the zero flag after an operation; a subtraction for example, and then use the zero flag to control a conditional branch. However, for Javelin, the branch is built directly into the subtraction; and most other instructions, so no separate Zero flag is required.

Carry is typically different as it is often needed for multiple, successive operations.

Accessing Memory Map

Under normal conditions the program cannot access RAM outside of its own partition, so no calculated RAM physical address below CBASE or greater than or equal to CBASE + TOTALSIZE is valid. The CPU will TRAP if this rule is broken.

However, there are exceptions.

The instruction OSCALL which is used to call operating system routines, always has its target address specified as an absolute address, and can call anywhere within $008000 to $00FFFF and allows execution of routines held within the system ROM(s).

When cleared (=0), the FLAGS.PROTECT flag grants read/write/execute access to any absolute RAM address, or read/execute access to any ROM address.

PTYPE	Ident.	PROTECT Memory Access	Description
0	$	SYS Only	System Driver Can access own partition, any RAM marked as for SYSTEM use, and any system ROMs Can also lock I/O port ranges Will still need to clear the FLAGS.PROTECT bit to access outside its own partition.
1	S	Any	System Component Can access any RAM or ROM within the system including all other program partitions. Will still need to clear the FLAGS.PROTECT bit to access outside its own partition.
2	*	Partition	Everything Else Can access own partition and any system ROM areas. Will not be able to manipulate the FLAGS.PROTECT bit.
3		Not valid	

Accessing I/O Devices

The I/O map covers $000000 to $FFFFFF and with the exception of $000000 to $000FFF, all CPUs can access any I/O port.

There will be provision at some point to "lock" I/O port ranges by programs, meaning that only the program locking the I/O port range can access it. This will be useful when, for example, a disk driver requires exclusive access to a disk controller board. It can lock the required I/O range preventing any other program accidently accessing the hardware. (Will probably add an OSCALL to return the owner of an I/O port and to lock it)

I/O Map

Start Address	End Address	Size	Usage	Usage
$000000	$000FFF	4K	CPU	Access to the current CPUs JSC
$001000	$FFFFFF			User for system devices (to be decided)

Each CPU has access to its own JSC (Javelin Service Controller) via I/O ports $000000 to $0000FF (these are mapped internally by the CPU to the JSC)

These I/O ports allow the JSC to be configured and interrogated.

Exact usage to be defined later.

Stacks

There are two stacks for each program partition.

There is the DSP (Data Stack Pointer) which is typically used by the program running in the partition to store data. These data can be 8, 16, or 24-bit values.

There is also the RSP (Return Address Stack Pointer) which is used to store relative return addresses generated by interrupts and subroutine calls.

Bytes, Words and 24-bit values can all be PUSHd / POPd to these stacks, but this causes a difficulty as system RAM is 16-bits (1 word) wide and there is no provision to indicate what type of value is stored at each slot in the stack.

There are four stack related registers maintained internally by the CPU and not directly accessible from programs.

RSP & DSP

These hold the current relative start address of the stacks and doesn't typically change during the life of the program.

RSPCOUNT & DSPCOUNT

These hold a count of the number of bytes, not words, that are currently on each stack.

When reading or writing to the stacks, the CPU will do something very specific. Using CurrentWord = StackBase + (COUNT / 2), points to the next free Data Stack location.

If Count = 0 or an even number, this means that the related stack pointer is pointing to a completely empty word in memory, otherwise it is pointing to a half-used word.

The value to be written or read can either be an 8/16 or 24-bit value.

One byte of data is written to the High byte for a currently empty word, and one byte of data can be written to the Low byte for a half-used word.

The above process is repeated for the following bytes if there are any.

Note: Since the RSP only ever contains 24-bit addresses, the packing used for DSP is not strictly required, however, keeping both stacks operating in the same way means the same internal routines within the CPU can be used for processing both stacks.

Instruction Set Format

Instructions have a pre-defined format and this is described using an instruction syntax.

XOR <1st Reg/RegPart>, <2nd Reg/RegPart, CV8, CV16, CV24>, <UPDATE, NONE>, <BM>, <BC>, <ADDR 8/24>

The above is the XOR instruction.

1st Operand - <1st Reg/RegPart>

This indicates that this operand must be a Register or Register Part.

2nd Operand - <2nd Reg/RegPart, CV8, CV16, CV24>

This operand must be either a Register or Register Part, or a constant value of either 8, 16 or 24 bits.

3rd Operand - <UPDATE, NONE>

The 3rd operand in this instruction indicates if the product of the XOR operation should be stored back into the 1st operand.

<BM> This is the branch mode

<BC> This is the branch condition

<ADDR 8/24> This is an address and must be either an 8 or 24 bit value. It's type will be dictated by the branch mode <BM> specified above.

The instruction will be stored with the instruction number first, followed by the operand types and then the operand values.

For example, the instruction:

AND R4M, 21, NONE, JUMP, ZERO, -120

This means AND the R4M register/part, with decimal 21, do not store the result of the AND back into R4M, and then JUMP to relative address -120 if the result of the AND was ZERO

AND <1st Reg/RegPart>, <2nd Reg/RegPart, CV8, CV16, CV24>, <UPDATE, NONE>, <BM>, <BC>, <ADDR 8/24>

This would be physically encoded as:

Word	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0
1	0	0	0	0	1	0	1	0	0	0	0	1	0	1	0	1
2	1	0	0	0	1	0	0	0	-	-	-	-	-	-	-	-

00100	This is the instruction number
1100	1st Register (R4)
01	1st Register Part - Middle Byte
0000	2nd Register CV8
00	2nd Register Part - Low Byte
0	UPDATE/NONE Flag
001	JUMP 8-bit Relative (Signed)
010	Jump on ZERO
00010101	21
11111000	-120 Jump Address

As can be seen, all the numeric constants are stored at the end of the instruction. The size of each of these constants is known by the operand description supplied earlier in the instruction. The extension values are always in the same order as the operands so the CPU knows exactly where to find them and how large they are.

Instruction Branch Modes

There are several jump types or "Branch Modes" supported by the instructions set.

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

0 - No Jump

This indicates that there is no jump of any type for this instruction and that there will be no bytes/words reserved for any type of jump destination address.

1 - JUMP 8-Bit Relative (signed)

Indicates that there will be an 8-bit signed value that indicates the number of addresses to jump either forwards or backwards from the current PC. If the 8^{th} bit is set = 1, this is a jump backwards, else it's a jump forwards.

2 - JSUB 8-Bit Relative (signed)

Almost identical to BM 1 except that this is a jump to a subroutine, so the return address is automatically placed on the RSP stack.

3 - OSCALL 24-Bit Absolute to System ROM (unsigned)

This is a jump to a subroutine at an absolute location in memory. The return address is placed on the RSP stack. The destination address MUST reside within a System ROM. A TRAP will be issued if an OSCALL is attempted to any other destination.

4 - FJUMP 24-Bit Relative (signed)

This is just an extension to the BM 1 in that it allows a relative 24-bit destination within the current partition to be selected. This is useful for larger programs where a +/- jump of 127 addresses using the smaller and faster BM 1 will not reach in one step.

5 - FJSUB 24-bit Relative (signed & only within partition)

This is just an extension to the BM 2 in that it allows a relative 24-bit destination within the current partition to be selected. This is useful for larger programs where a +/- jump of 127 addresses using the smaller and faster BM 2 will not reach in one step.

Note: In all but the OSCALL jumps, the destination MUST reside within the current partition otherwise a TRAP will be issues.

Note: Relative branch offsets are calculated from the address immediately following the complete instruction.

NOP - No Operation

No registers are affected other than the PC being incremented by +1

RET - Return from subroutine

When a subroutine completes, RET should be called to return program control back to the next instruction after the one that made the subroutine call.

Note: See section on Stacks for information on how the stack pointers are manipulated and calculated.

IRET - Interrupt Return

When an interrupt subroutine completes, IRET should be called to return back to the next instruction after the one that made the subroutine call.

Just before the IRET completes, the IBUSY flag in the FLAGS register is cleared.

All return addresses stored on the RSP are 24-bit values, relative to the CBASE. This means that these addresses are relocatable in the event that the program is relocated in memory during its execution.

Note: See section on Stacks for information on how the stack pointers are manipulated and calculated.

TRAP - Raise a system TRAP and optionally HALT CPU

TRAP <11-bit numeric value>

n = TRAP number, and can be in the range 0 to 2047

TRAP's 0 to 255 are resumable and are not classed as critical system stops so are useful for application programs. Typically, the application program will terminate; not always cleanly.

Any TRAP from 256 to 2047 is treated as a critical system TRAP and the entire CPU will be suspended. This can have adverse effects on

other CPUs in the system depending on what they were currently doing. Use this range of TRAPs with caution.

PUSH - Push data onto the Data Stack

PUSH <CV, Reg/RegPart>

PUSH R0L ' Push the Low byte (8-bits) of the 24-bit R0 register onto the stack.

PUSH MD ' Reads 1 word of memory from the memory address stored in the MADDR register, and then push that fetched word, High byte first, onto the stack

Values are always pushed, High byte first

Note: See section on Stacks for information on how the stack pointers are manipulated and calculated.

POP - Pop data from the Data Stack into a register

POP <Reg/RegPart>

Allows a value (2 or 3 bytes) to be POP'd off the top of the DSP and into the specified register.

Multi-byte values are pushed High byte first. Consequently, when popped byte-by-byte, the Low byte is returned first.

Note: See section on Stacks for information on how the stack pointers are manipulated and calculated.

BSET - Set bit in register or memory

BSET <Reg/RegPart>, <bit number 0 - 23>

Allows the specified bit in a register to be set = 1

Attempting to set an invalid bit for the target register will cause a TRAP message.

BCLR - Clear bit in register or memory

BCLR <Reg/RegPart>, <bit number 0 - 23>

Allows the specified bit in a register to be set = 0

Attempting to set an invalid bit for the target register will cause a TRAP message.

BNOT - Toggle bit in register or memory

BNOT <Reg/RegPart>, <bit number 0 - 23>

Allows the specified bit in a register to toggled or inverted.

Attempting to set an invalid bit for the target register will cause a TRAP message.

SHFL - Shift or Rotate register left 1 bit

SHFL <Reg/RegPart>, <RC>, <CCB>, <CCA>, <ROT>

Shifts or rotates a target register or memory location, 1 bit left.

RC = When rotating, should the register be rotated through Carry. This only works if the full register width is rotated

CCB = Should the Carry bit be cleared before the rotate or shift (even though you cannot shift through carry, you can clear it)

CCA = Should the Carry bit be cleared after the rotate or shift (even though you cannot shift through carry, you can clear it)

ROT = Should the register be SHIFTED = 0 or ROTATED = 1

SHFR - Shift or Rotate register right 1 bit

SHFR <Reg/RegPart>, <RC>, <CCB>, <CCA>, <ROT>

Shifts or rotates a target register or memory location, 1 bit right.

RC = When rotating, should the register be rotated through Carry. This only works if the full register width is rotated

CCB = Should the Carry bit be cleared before the rotate or shift (even though you cannot shift through carry, you can clear it)

CCA = Should the Carry bit be cleared after the rotate or shift (even though you cannot shift through carry, you can clear it)

ROT = Should the register be SHIFTED = 0 or ROTATED = 1

INC - Increment a register or memory location by 1

INC <source Reg/RegPart>, <dest Reg/RegPart>, <BM>, <BC>, <ADDR 8/24>

Takes the value in the source register and increments it, storing the result in the destination register.

Depending on the resulting value, it's possible to execute a branch instruction.

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

BC	Branch Condition
0	Jump Always
1	Carry Set
2	Zero
3	Negative
4	Never Jump
5	Carry Clear
6	Not Zero
7	Not Negative

Negative ADDR numbers are relative jump backwards.

Note: there are no 16-bit addresses, only relative 8-bit and relative/absolute 24-bit ones.

DEC - Decrement a register or memory location by 1

DEC <source Reg/RegPart>, <dest Reg/RegPart>, <BM>, <BC>, <ADDR 8/24>

Takes the value in the source register and decrements it, storing the result in the destination register.

Depending on the resulting value, it's possible to execute a branch instruction.

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

BC	Branch Condition
0	Jump Always
1	Carry Set
2	Zero
3	Negative
4	Never Jump
5	Carry Clear
6	Not Zero
7	Not Negative

Negative ADDR numbers are relative jump backwards.

Note: there are no 16-bit addresses, only relative 8-bit and relative/absolute 24-bit ones.

AND – Logical AND

AND <1st Reg/RegPart>, <2nd Reg/RegPart, CV8, CV16, CV24>, <UPDATE, NONE>, <BM>, <BC>, <ADDR 8/24>

<UPDATE> Copies the result of the AND operation into the 1st Register specified.

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

BC	Branch Condition
0	Jump Always
1	Carry Set
2	Zero
3	Negative
4	Never Jump
5	Carry Clear
6	Not Zero
7	Not Negative

Negative ADDR numbers are relative jump backwards.

Note: there are no 16-bit addresses, only relative 8-bit and relative/absolute 24-bit ones.

OR – Logical OR

OR <1st Reg/RegPart>, <2nd Reg/RegPart, CV8, CV16, CV24>, <UPDATE, NONE>, <BM>, <BC>, <ADDR 8/24>

<UPDATE> Copies the result of the OR operation into the 1st Register specified.

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

BC	Branch Condition
0	Jump Always
1	Carry Set
2	Zero
3	Negative
4	Never Jump
5	Carry Clear
6	Not Zero
7	Not Negative

Negative ADDR numbers are relative jump backwards.

Note: there are no 16-bit addresses, only relative 8-bit and relative/absolute 24-bit ones.

XOR – Logical XOR

XOR <1st Reg/RegPart>, <2nd Reg/RegPart, CV8, CV16, CV24>, <UPDATE, NONE>, <BM>, <BC>, <ADDR 8/24>

<UPDATE> Copies the result of the XOR operation into the 1st Register specified.

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

BC	Branch Condition
0	Jump Always
1	Carry Set
2	Zero
3	Negative
4	Never Jump
5	Carry Clear
6	Not Zero
7	Not Negative

Negative ADDR numbers are relative jump backwards.

Note: there are no 16-bit addresses, only relative 8-bit and relative/absolute 24-bit ones.

INV – Logical 1's or 2's Complement

INV <(Source) Reg/RegPart>, <(Destination) Reg/RegPart>, <(mode)NOT, NEG>, <UPDATE, NONE>, <BM>, <BC>, <ADDR 8/24>

Mode

NOT	One's complement	Invert all bits of the source
NEG	Two's complement	Arithmetically negate the source

The result of the INV operation is copied into the destination register if UPDATE is set; the resulting output width is determined by the width of the source.

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

BC	Branch Condition
0	Jump Always
1	Carry Set
2	Zero
3	Negative
4	Never Jump
5	Carry Clear
6	Not Zero
7	Not Negative

Negative ADDR numbers are relative jump backwards.

Note: there are no 16-bit addresses, only relative 8-bit and relative/absolute 24-bit ones.

SWAP - Swap Registers/RegParts

SWAP <1st Reg/RegPart>, <2nd Reg/RegPart>

Allows registers or parts of registers to be swapped.

Be carful if the two operands are not the same size.

Trying to SWAP a 24 bit register with an 8-bit register part will cause the 8-bit equivalent parts to be exchanged.

You can also use SWAP to exchange bytes within the same register

COPY - Copy source to destination

COPY <src Reg/RegPart, CV8, CV16, CV24>, <dest Reg/Regpart>, <BM>, <BC>, <ADDR 8/24>

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

BC	Branch Condition
0	Jump Always
1	Carry Set
2	Zero
3	Negative
4	Never Jump
5	Carry Clear
6	Not Zero
7	Not Negative

<ADDR 8/24> This is an address and must be either an 8- or 24-bit value. Its type will be dictated by the branch mode <BM> specified above.

Negative ADDR numbers are relative jump backwards.

Note: there are no 16-bit addresses, only relative 8-bit and relative/absolute 24-bit ones.

CMP – Compare Two Values

CMP <1st Reg/RegPart>, <2nd Reg/RegPart, CV8, CV16, CV24>, <BM>, <BC>, <ADDR 8/24>

Compares the two values.

If the two values are the same, the result is Zero

If the first value is smaller than the 2nd, the result is Negative

No registers are updated with this instruction

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

BC	Branch Condition
0	Jump Always
1	Carry Set
2	Zero
3	Negative
4	Never Jump
5	Carry Clear
6	Not Zero
7	Not Negative

<ADDR 8/24> This is an address and must be either an 8- or 24-bit value. Its type will be dictated by the branch mode <BM> specified above.

Negative ADDR numbers are relative jump backwards.

Note: there are no 16-bit addresses, only relative 8-bit and relative/absolute 24-bit ones.

IN - Read a value from an Input port.

IN <(Port Number) Reg/RegPart, CV8, CV16, CV24>,<(Destination) Reg/RegPart>), <(Timeout ms) Reg/RegPart, CV8, CV16, CV24>, <Index With IX>, <BM>, <BC>, <ADDR 8/24>

Inputs a value from an input port and stores the value in the destination. If the "Index with IX" is set and the destination is a memory address, then the value of IX is added to the destination address. After the word is stored, IX is automatically incremented.

Because I/O requests need to be acknowledged by the hardware in the same was as memory access requests, it's possible for the system to hang whilst waiting for a slow or none-existent I/O port to respond. For this situation a time-out value can be specified. Should a time-out occur, the branch will be executed. Only the branch Mode is evaluated; the branch condition is defaulted to "jump always".

OUT - Output a value to an Output port.

OUT <(Port Number) Reg/RegPart, CV8, CV16, CV24>,<(Source) Reg/RegPart>, CV8, CV16, CV24), <(Timeout ms) Reg/RegPart, CV8, CV16, CV24>, <Index With IX>, <BM>, <BC>, <ADDR 8/24>

Outputs a value read from the source to an output port. If the "Index with IX" is set and the source is a memory address, then the value of IX is added to the destination address. After the word is output, IX is automatically incremented.

Because I/O requests need to be acknowledged by the hardware in the same was as memory access requests, it's possible for the system to hang whilst waiting for a slow or none-existent I/O port to respond. For this situation a Time-out value can be specified. Should a time-out occur, the branch will be executed. Only the branch Mode is evaluated; the branch condition is defaulted to "jump always".

MATH - Perform a maths operation

MATH <(Value1) Reg/RegPart, CV8, CV16, CV24>, (Destination/Value2) Reg/RegPart), <(Maths Mode)MM>, <UPDATE, NONE> <BM>, <BC>, <ADDR 8/24>

The result of the MATH operation is copied into the destination register if UPDATE is set. The resulting output width is determined by the width of the destination.

MM	Maths Mode
0	ADD
1	ADDC
2	SUB
3	SBC
4	MUL
5	DIV
6	(Could be used for RND later on)
7-15	Reserved

BM	Mode Description
0	No Jump
1	JUMP 8-Bit Relative (signed)
2	JSUB 8-Bit Relative (signed)
3	OSCALL 24-Bit Absolute to System ROM Sub (unsigned)
4	FJUMP 24-bit Relative (signed & only within partition)
5	FJSUB 24-bit Relative (signed & only within partition)
6	Reserved
7	Reserved

BC	Branch Condition
0	Any Condition
1	Carry Set
2	Zero
3	Negative
4	[spare]
5	Carry Clear
6	Not Zero
7	Not Negative

Instruction List

Instruction Number	Instruction
0	NOP
1	RET
2	IRET
3	TRAP
4	PUSH
5	POP
6	BSET
7	BCLR
8	BNOT
9	SHFL
10	SHFR
11	INC
12	DEC
13	MATH
14	AND
15	OR
16	XOR
17	NOT
18	INV
19	SWAP
20	COPY
21	CMP
22	IN
23	OUT
24-31	Reserved

There are 5-bits available to help uniquely identify the CPU instruction, hence there are only 32 available top-level instructions. However, many instructions have sub functions; like the MATH CPU instruction, so in reality there are many more instructions than 32.

I've opted for a CISC (Complex Instruction Set Computer) to try and reduce the number of backplane accesses needed for each instruction; specifically, to reduce the number of memory hits required.