Computer System Support for Scientific and Engineering Computation

Guest Lecture 15 by Tom Anderson - June 21, 1988 (notes revised August 26, 1988)

1 Introduction

The design philosophy of the Cydra-5 was to build a balanced system that was strong in all applications, not just in number crunching. The machine had to be able to handle real-world problems. Thus, it was important that the machine not only be able to handle numerical problems well, but also be a fast, general-purpose computer.

2 General Overview

2.1 Architectural overview

The Cydra-5 can be divided into three parts:

Interactive Processors: The front end of the machine provides support for general-purpose computing. It consists of Motorola 68020-based cpu boards, each with a private cache. There are six boards that sit on a proprietary bus. A proprietary bus was developed in order to provide special communications features to support multi-processing.

There are also separate I/O processors. Up to two I/O processors can be supported, each with its own VME bus. Each I/O processor can control three VME card cages. Here a standard I/O interface was chosen because Cydrome didn't want to commit resources in developing controllers for the many high-speed I/O devices available on the market.

There is also a service processor that is used for diagnostic information gathering and maintenance. A PC-based system console is used to communicate with the service processor.

Numeric Processor : The numeric processor is a "directed dataflow processor". It is a high-speed, ECL-based system that uses dataflow techniques to gain the maximum parallel execution from standard FORTRAN code. At the heart of the numeric processor is the "context register matrix", a partial crossbar that partially connects all the processor elements together through a common register file. The numeric processor is the focus of this lecture, and much more detail will be presented later.

System Memory: There are two main parts to the system memory. There is the main memory, which can be as large as 512 megabytes, and the support memory, which has a maximum size of 64 megabytes.

The main memory is designed to hold large data structures, such as the large vector and matrix structures common in numerical code. It is accessed in a pipelined fashion, where accesses are overlapped to provide fast access to program data. It is connected to the numeric processor across a 400 megabit/second bus. This memory is designed to have a high throughput, but long latency, to support large, data-intensive computations in the numeric processor.

The support memory is used to support more latency-dependent data, such as the process tables for the operating system. This memory is optimized for latency, and has a lower throughput than the main memory.

All the memory resides in the same memory space. However, throughput-critical data will reside in the main memory, and latency-critical data will reside in the support memory.

2.2 Numeric Processor Overview

As stated before, the numeric processor uses dataflow techniques to achieve parallelism in programs. The numeric processor relies on the compiler to schedule this parallelism. The numeric processor can start more than one operation per cycle. Up to seven operations can be started in a VLIW-like fashion, where a single long instruction contains operations on various functional units, and these operations are fired off in parallel. The operations start, execute, and complete in parallel. Compiler scheduling controls how results are written into the register file and prevents conflicts during result writes. Some of the functional units are pipelined. The "context register matrix" is used to channel data between the functional units. Conditional branches are handled in a special way: both ways of a branch sequence are scheduled, and at runtime one or the other branch sequence is disabled, depending on the control condition. There will be more discussion on the context register matrix and the conditional scheduling control later.

The machine is built using air-cooled, 100K ECL technology. It is designed to operate in a typical computer room environment; it requires no special air conditioning. It runs at 25 MHz, without any internal clock divisions to allow parts of the hardware to run at faster speeds.

The Cydra-5's peak performance is 175 million operations per second (MOPS), 50 million floating-point operations per second (MFLOPS) with 32-bit data, and 25 MFLOPS with 64-bit data. The machine conforms to the IEEE 754 binary floating-point standard and all its recommendations. Most of the compatibility is provided by the hardware.

The machine has a four gigabyte virtual address space available for each process. The numeric processor has a 32 kilobyte instruction cache, and no data cache. The reason no data cache was provided was because of the presence of the high-speed memory, and the fact that the compiler takes memory latency into account when scheduling code.

2.3 Main memory

The main memory can be as large as 512 megabytes, using 1 megabit DRAMs. It uses 8- to 64-way interleaving. It has a 400 megabit per second sustained bandwidth.

The memory is designed to be stride insensitive. In ordinary interleaving schemes, the memory is vulnerable to significant performance degradation if the interleaving size happens to equal the stride size of a particular data structure. This unfortunate coincidence results in consecutive hits in the same memory bank, dramatically decreasing the performance of the memory system.

The Cydra-5 uses a "transformed interleaving" scheme, using a pseudo-random pattern to scatter banks such that any regular pattern of accesses will not access the same bank several times in succession. This transformed interleaving scheme works well for real programs; of course, contrived cases can be generated to access the memory system in such a way as to defeat the interleaving scheme. The transformed interleaving on a machine can be disabled; thus, the performance improvement of the transformed scheme can be measured. A dramatic performance improvement was observed.

2.4 General purpose subsystem

The general purpose subsystem uses a 100 megabit/second system bus. From one to six interactive processors can be connected to the bus. Each processor has a 16 kilobyte private cache. System-wide cache coherency is always maintained. Each processor has from eight to 32 megabytes of support memory. One or two I/O processors can be on the bus, and each can support up to three VME buses. I/O operations can be sustained at a rate of 40 megabytes/second for each I/O processor. Up to 60 concurrent I/O operations can be handled. The I/O system is memory mapped. Floating-point in the interactive processors is done with the 68881. The service processor is used to monitor system operation. Diagnostics can be run through the service processor to debug problems with the system.

2.5 Operating system

The Cydra-5 runs a Unix-like operating system called Cydrix. It is fully compliant with the System V interface definition. The kernel is a "symmetrically distributed kernel", meaning that the same kernel runs on all of the interactive processors. The multi-processing is synchronized such that the processors access the process table coherently; this synchronization is achieved by having a process lock out other processes while it is accessing the process table.

The Cydra-5 offers transparent management of a heterogeneous multiprocessing environment. The numeric processor is just another processor as far is the operating system is concerned, although it runs with different object code. The Cydra-5 is a heterogeneous environment because two different processor types exist in the system. Also, because the numeric processor is considered to be just another processor, it can be time-shared and managed just like the interactive processors. The time-sharing is flexible; however, the time between process switches in the numeric processor can and should be longer than that in the interactive processor. Unlike the interactive processor, the numeric processor does not need to offer the illusion of real-time to the user. Also, because the numeric processor has so many registers, it is good to reduce having to spend the time saving the state of the numeric processor on context switches.

The machine uses an extent-based file system, which allows files to be laid in contiguous blocks. Also, disk striping is supported to increase disk access times. I/O is unbuffered and asynchronous, allowing the I/O processing to run along in parallel with processor operations.

2.6 Compiler technology

The Cydra-5 supports multiple language front-ends. Currently, only FORTRAN is available. Other languages will be supported by the end of the year. A common, machine-independent optimizer is used; the optimizer operates on an intermediate language format. Two different back-ends then operate on the intermediate code to generate code for the numeric processor or the interactive processor.

The key to Cydrome's success is its directed dataflow scheduling concept, and this concept is implemented by the compiler. This scheduling allows parallel execution of overlapped iterations. It overlaps parts of each iteration and executes those parts in parallel. This scheme is better than vectorization because vectorization cannot handle recurrences and other interdependencies between array or vector elements. The directed dataflow techniques also allows scalar code to run fast by allowing independent scalar instructions to execute in parallel. The parallelization of scalar code is done to produce "eager" execution, where operations are scheduled as early as possible. Because of the directed dataflow scheduling, there is no need to keep branch statistics.

2.7 Architecture philosophy

Cydrome has shown a commitment to standards. This commitment has resulted in a savings in development time. By not having to develop system components that are available off-the-shelf, Cydrome has been able to concentrate its resources on the core of the machine. Standards that are followed include the ATT System V Unix operating system, TCP/IP, NFS and DECNET networking architectures, the ANSI FORTRAN 77 programming language, the IEEE 754 standard for floating-point arithmetic, and the VME bus standard.

Cydrome has developed an architecture that provides support for a heterogeneous multiprocessing environment. Multiprocessing is done at the process level. The numeric processor is optimized for large, numerically-intensive applications, and the interactive processor is designed for interactive, non-numerically-intensive programs, such as terminal drivers and screen editors. The system architecture also supports I/O processors. Because such a variety of hardware is present in the system, Cydrome refers to it as a "heterogeneous" environment.

2.8 "Performance that Counts"

The peak performance of the Cydra-5 is at about the same level as that of the mini-supers. But Cydrome's goal is to have the performance hold up as other, non-numerical work loads are added on to the system. Here it can be shown that the high performance of the Cydra-5 is sustained, unlike in the other mini-super computers where the performance starts degrading as non-numerical workloads are added.

2.9 Questions and comments

At what level does the multiprocessing occur? The level of multi-processing is at the process level. The compiler cannot break up one process across processors.

Comment on the consistency of results between the interactive and the numerical processors. It is possible to get different results between the interactive processor and the numerical

processor, but no customers have complained so far. There is a compiler switch that can force the 68881 to be used in all situations.

Comment on future generations and their impact on compiler scheduling. A faster clocked version means that the memory latency appears longer, assuming that the memory speed doesn't scale. Memory latency is determined at compile time, so code could be recompiled on newer machines. Also, there is checking for dependencies on memory data, so nothing disastrous would happen if the memory latency were to change.

Why couldn't the compiler do the memory transformation? It was found to be easier to debug the memory system in hardware, since it was easier to switch off a bad memory board in order to isolate a problem.

What is the latency of the floating-point units? The ALU and the multiplier take 4 cycles for single precision operations and eight cycles for double precision operations. Both units are fully pipelined. The divide and square root units take about 15 cycles for single precision and about 30 cycles for double precision, and these units are not pipelined.

3 Detailed description of the Cydra-5

3.1 Parallelism

There are two main types of parallelism. There is coarse-grain parallelism, where the parallelism is at the process level or higher. The interactive processors operate at this level. The other type of parallelism is fine-grain parallelism. Vector and array processors operate at this level, but these types of processors can only handle a specific subset of the types of problems that machines need to operate on. A dataflow machine is more general and can offer fine-grain parallelism across general-purpose code.

3.2 Dataflow

The first step in generating code for a dataflow machine is to generate a dataflow graph of the source code. Such a graph shows how data from various operations flow to subsequent operations. The graph will show interdependencies between code fragments; these dependencies will show what parts of the code can or cannot be parallelized. Independent fragments can then be scheduled to execute in parallel.

The Cydra-5 is a unique dataflow machine in that the parallelism is generated at compile time, unlike other dataflow machines which have specialized hardware to schedule code at run time. The Cydra-5 hardware knows nothing about data dependencies; all dependencies are resolved by compiler scheduling. Some hardware interlocking is provided, however, to protect against a subset of programming errors, and to allow for the fact that memory access latencies are non-deterministic.

Technically, the Cydra-5 is not a "dataflow machine" per se, but a machine that "supports the dataflow concept" by providing simple, powerful hardware and relegating the dataflow scheduling to the compiler. Thus, Cydrome's concept of directed dataflow is a combination of two powerful philosophies. It combines the comprehensive exploitation of fine-grained parallelism provided by the dataflow concepts with simple, efficient hardware. The result is a compiler directed dataflow computing machine with support from hardware

features such as the context register matrix and conditional scheduling control (both discussed later). The compiler is optimized for currently existing languages, although currently only FORTRAN is available.

3.3 Numeric processor datapaths

The numeric processor has seven functional units: a combination floating-point and integer ALU, a combination floating-point and integer multiply/divide/square root unit, two memory ports, and three units for address calculation. The ALU provides arithmetic and logical instructions for integer data, and add, subtract, compare and convert for floating-point data. The multiplier provides multiplication and division for both integer and floating-point data. It also provides a modulus operation for integer data, a square root function for floating-point data, and partial remainder primitive to aid in a software-implemented remainder function for floating-point data. The two memory ports provide both read and write access between the register file and the memory system. There are three address units: one is a simple displacement adder for address calculation, the second is a multiplier to aid in array indexing operations, and the third is a bit-reversing unit to help in certain types of applications, such as fast Fourier transform algorithms.

There is also an instruction unit that provides the basic branch and control operations, as well as supervisor instructions.

At the heart of the numeric processor is the context register matrix. This matrix is a partially connected crossbar, providing partially orthogonal access to the register file from all the functional units. There are seven banks of registers in the register file, each consisting of 64 32-bit registers. There is one general purpose register bank (GPR), four banks on the data side of the machine, and two banks on the address side. All seven register file banks are visible to the programmer.

On the data side, any unit can read operands out of any of the four data register file banks, as well as the GPR bank. However, results must be written to the bank that is dedicated to the particular unit generating that result. Thus, there are four register file banks, one for each of the functional units. The GPR bank is common to all four units, and results from any of the units can be written into the GPR. The register file can be viewed as a one write and six read port register file, where there are two read ports each for the multiplier and the ALU and one read port to each of the memory units. The GPR is a potential choke point; the compiler must schedule code such that units don't contend with one other when writing results to the GPR.

The address side has a similar arrangement. The displacement adder, the multiplier, and the bit-reverse units can read out of any of the two register banks dedicated to the address side of the numeric processor, as well as from the GPR. The address multiplier and the address bit reversal unit can only write into its dedicated register bank, or the GPR. The result of the displacement adder goes directly into the memory units on the data side of the processor.

All buses between the units are 32-bit buses; thus, double precision transfers require two cycles to execute.

Although the GPR is a potential bottleneck point, writes to the GPR rarely occur because the GPR usually holds special purpose constants that rarely change.

An interesting statistic to examine, considering how many registers the machine has, is the vacancy rate of the registers; that is, how often is a particular register in the register file empty. This metric is difficult to obtain. A more interesting measure would be how

busy the functional units can be kept.

Subroutine calls in the numeric processor are expensive in that the hardware doesn't save much state automatically; the software must be sure that registers are saved if they are to be modified. Because so many registers exist, context switching can be expensive; thus, context switches in the numeric processor are made less frequent to avoid the heavy overhead of saving the processor state.

3.4 Compiler scheduling

The scheduling of operations by the compiler is the key to making the machine perform at its best. By recognize opportunities for parallelism, operations can be started at a rate of more than one per cycle. The compiler scheduling is covered in detail in [1]. Some key points will be discussed here.

Loop handling: Operations of different iterations are overlapped to maximize the utilization of the functional units. Instead of generating prologue and epilogue code to handle the loop startup and closing operations, there is an instruction control register (ICR) that is used to disable operations that shouldn't occur during the first and last passes through the loop body.

Conditional branch scheduling: When a conditional branch is encountered, the compiler generates and schedules code for both directions of the branch. During execution, the ICR is used to nullify instructions on the path that is not taken. Currently, the compiler can only handle one branch, but it will be possible to handle multi-way branches in future releases.

Iteration counters: Iteration counters are used to increment register addresses to aid register accessing during loop iterations.

3.5 Performance

An example was presented to demonstrated the power of the compiler scheduling. A code fragment from a Viterbi decoder was used, compliments of the Jet Propulsion Lab in Pasadena, California. This program is used in signal processing applications. It is not meant to be a real benchmark, but just an example of the compiler at work.

In uni-op mode, where at most only one operation can start each cycle, a total of 154 cycles must be spent to get through the code fragment. Many dead cycles are wasted waiting for data to arrive from memory. In multi-op mode, however, the cycle count drops to 23 cycles. During each cycle at least one functional unit is busy. In fact, the limiting unit is the ALU; if the functions of the ALU were distributed across different functional units, it may be possible to improve the cycle count even further.

In terms of overall performance, the Cydra-5 can run the double precision Livermore loops at 4.5 MFLOPS; this number is the harmonic mean of the performance of the individual loops [2]. The 100x100 double precision linpack benchmark can run at 14 MFLOPS.

To date 15 systems or system-equivalents have been shipped.

3.6 IEEE options and extensions

The Cydra-5 offers full hardware support for denormalized operands with no penalty. Arithmetic operations on a mixture of precision is not allowed, although conversions too and from

formats are supported. There are no complicated arithmetic operations, such as transcendental functions. There is no instruction to return the fraction part of a floating-point number. An instruction that truncates the integer-to-floating-point convert result was added to support the FORTRAN definition. Since this operation with this rounding mode is used frequently, this instruction was added to alleviate the need to set rounding modes whenever this conversion was needed. Finally, a "flush-to-zero" mode is provided which turns denormalized results into zero.

3.7 Floating-point algorithms

Addition: The addition algorithm is basically the standard, textbook algorithm. A right shift of the smaller operand is first performed to align the operands on the binary point, then the numbers are added. A final shift of one may be needed to normalize the result.

Subtract: The traditional subtraction algorithm requires a (potentially) large shift after the standard align-and-subtract operation to re-normalize the result, since cancellation of many leading digits may result in many leading zeros in the result. The Cydra-5 has logic that predicts the cancellation and pre-normalizes operands; thus, the resulting difference will require at most a 1 bit shift to normalize. This cancellation prediction saves time and space by eliminating two levels of shifters in the datapath, although the leading shifter is now more complex because it must be able to shift both left and right.

Compare: The compare operation is implemented as a subtract operation that generates a boolean result.

Multiply: The multiplier can generate a complete 64 × 64 result in two cycles. It uses a single shifter to fix up the result at the end; a full shifter is needed because the multiplier can accept denormalized numbers as operands. The multiplier is made more complicated by having to support different data types:

Integer: In an integer multiply the lower 64 bits are kept for the result and the upper 64 bits are folded into the overflow detection logic.

Normalized floating-point: In a floating-point multiply the upper 64 bits are to be returned, and the lower 64 bits are accumulated into the sticky bit.

Denormalized floating-point: Supporting operations on denormalized numbers requires the use of a leading zero counter at the input to locate where the binary point should be in the product; this need complicates the datapath by requiring hardware at the front end to count the leading zeros and a large shifter at the back end to normalize the result.

Divide: The division algorithm develops 12 quotient bits every three cycles. The basic equations governing the divide algorithm are:

$$\frac{dividend}{divisor} = quotient + remainder$$

Let

$$q = approximate\ quotient,\quad r = true\ remainder$$

then

$$q = dividend \times \frac{1}{divisor}$$
$$r = dividend - (1 \times divisor)$$

The iterations are controlled by:

$$q(i) = r(i-1) \times \frac{1}{divisor}$$
$$r(i) = r(i-1) - divisor \times q(i)$$

Truncated values of the remainder and the divisor reciprocal are used to produce quotient bits; thus, the first few iterations are faster because they use a narrower datapath. The initial truncated divisor reciprocal is in a lookup table.

Square root: The square root algorithm develops seven bits each three cycles; thus, it is slower than the division algorithm. If we let S be the approximate square root and r be the remainder, then the following equations describe the square root process:

$$2 \times S(i) = R(i-1) \times \frac{1}{S(0)}$$
$$R(i) = R(i-1) - (2 \times X(i-1) + S(i)) \times S(i)$$

where

$$X(i-1) = S(0) + S(1) + \ldots + S(i-1)$$

The truncated values of the remainder and the initial guess reciprocal are used to produce square root bits, just as in the divide algorithm. The truncated initial square root guess reciprocal is located in a lookup table.

3.8 Influence of architecture on implementation

The combination of parallelism and pipelining complicates the implementation of the hardware. Out-of-order completion affects exception handling. Some parallel operations may have completed when an exception is detected; thus, there is a danger of other operations overwriting inputs of the exceptional operation. For example, suppose operation A starts at time 0, and operation B starts at time 1. Suppose the latency of operation A is longer, so operation B completes at time 3, overwriting one of operation A's source operands. Then operation A finishes and generates an exception. Operation A's inputs are lost, so the instruction cannot be reissued!

The compiler must enforce protection of source registers during the entire latency of the operation. Although this seems like an enormous task, this job is not as big a limitation on the compiler as other problems, such as scheduling memory latencies.

When IEEE exceptions occur, the inputs to the exceptional instruction must be intact; therefore, results of an exceptional operation are not written for fear of clobbering one of the inputs. Constraining the compiler to never generate instructions that overwrite its own input registers is too restrictive in this case.

When an exception is signaled, the machine is stopped (the PC is frozen) and currently executing operations are allowed to finish. Thus, several cycles may be spent waiting for the machine to wind down before the exception handler is called.

The large number of registers definitely impacted overall cycle time, since register access time is large and the register file is in a critical path.

3.9 Exception handling

All IEEE exceptions are supported: overflow, underflow, inexact, invalid operation, and divide by zero. All IEEE exceptions have trap enables. Two integer exceptions are also provided: overflow and divide by zero. A queue stores PC values during normal execution. During exceptions, the queue contains a record of exceptions that occurred while the pipeline was flushing, in case subsequent operations also cause exceptions. If subsequent operations during the pipeline flush also complete exceptionally, the exception handler can use the information in the queue to backtrack and recreate all the exceptional instructions. A hierarchy of flags is used to identify the source of exceptions. This exception handling scheme gets more complicated when loop counters are considered, too. So far no real exception handlers have been written, so the impact of the complexity is not fully understood yet.

3.10 Implementation issues

Full support of denormalized operands is complex, and it may have been desirable to avoid this complexity. However, support of both integer and floating-point in the same unit reduced the impact of this complexity because the hardware to handle denormalized data is already contained in the hardware to handle integer data.

The complexity of the context register matrix structure had a significant impact on the design of the functional units. But since the matrix is the most important hardware feature of the Cydra-5, care was spent in making sure the design was done properly.

Only a few hardware interlocks are provided to protect the user from programming errors. Also, interlocks are provided for memory dependencies, since memory accesses are non-deterministic; two consecutive accesses may contend on the same memory bank, delaying the second access. Latencies such as these cannot be determined at compile time.

4 Mistakes and future issues

The value of hardware denormalized data support is still controversial.

The instruction count could be reduced further by limiting the number of possible comparisons. For example, $A < B$ is the same as $B > A$. The compiler could reorder such a comparison to use the other instruction.

A single precision integer to double precision floating-point convert operation, not provided by the hardware, turns out to be an important operation in FORTRAN code because often the loop index is needed in some computation in the body of the loop.

A better balance of operations between the different functional units should be considered. Typical code shows that the ALU is usually the bottleneck. A future machine should consider splitting up integer ALU operations from floating-point ALU operations. The exact distribution of functions across the functional units will need to be studied in detail. Possible schemes include providing more than one complete integer ALU unit. A set of fully orthogonal functional units will probably not be implemented, however.

5 References

[1] "Cydra-5 Directed Dataflow Architecture", Cydrome Inc.
[2] "The Correct Mean", Cydrome Performance Brief, Cydrome Inc.

Computer System Support for Scientific and Engineering Computation

Lecture 15 - June 21, 1988 (notes revised July 6, 1988)

1 Summary of Cydrome Architecture

Here is a brief summary of aspects of the Cydrome architecture relevant to floating point, taken from the guest lecture of Tom Anderson.

Many Registers The numeric processor has 7 register sets, each containing 64 registers for a total of 448 registers. The large number of registers helps support the parallel operation of multiple processing elements. In each cycle as many as 7 different registers can be written simultaneously. However, the large number of registers makes context switching expensive. The Cydrome architecture has *interactive* processors independent of the numeric processor for processes with high context switching rates.

Conditional Execution Each instruction has conditional execution bits that control whether the instruction is actually executed based on the loop iteration count or the evaluation of a conditional expression. Thus a loop can fetch a word from memory that won't be used until a later iteration. The final iteration through the loop the conditional execution bit can inhibit the memory fetch (and other operations), thus avoiding a memory reference that will never be used, and which could have caused a spurious page fault.

Pipelined Floating Point Operations The add and multiply units are pipelined. Like most pipelined processors, this means that the floating point operations always take the exact same number of cycles, independent of the operand values.

Software Scheduling Suppose the result of one floating point operation is used as the input to another operation. The hardware does not stall waiting for the first operation to complete. It is up to the compiler to schedule instructions so that a floating point operation isn't executing until its operands have been computed. Since the floating point latencies are compiled into the code, future machines implemented in different technologies must maintain the same latencies (in terms of cycle counts) to allow binary portability.

Compare done via Subtract Comparison is done by subtracting, and then putting out a single bit based on the sign of the result. This can cause problem for compares involving NaN's, however.

Gradual Underflow in Hardware Gradual underflow is implemented in hardware, with no performance penalty. Dealing with denormalized numbers makes floating point hardware more complicated. However, on the Cydrome machine the floating point hardware was already doing double duty for integer arithmetic, so support for denormalized numbers didn't add much extra.

Subtract Pre-Normalize When subtracting numbers, the traditional hardware must first do a potentially long shift to align the numbers, then subtract, then another potentially long shift to normalize the result. The Cydrome hardware increases performance by doing a pre-normalize. That is, when it aligns the binary points, it also estimates the amount of cancellation and shifts both operands to account for it. After the subtract, the significand will have to be shifted by at most one bit.

Support for Trap Handlers When a floating point trap occurs, the trap handler needs to be able to access the operands causing the trap. Since the Cydrome machine can perform multiple floating point operations in parallel, it will complete any other floating point operations in progress (i.e. flush the pipeline) before executing the trap handler. One of those pending operations might overwrite the arguments to the operation that trapped. The Cydrome compiler is responsible for scheduling operations so that this doesn't happen. Similarly, operations that trap do not produce any output, since that might also overwrite operands.

Type Conversion There is an instruction for truncating a floating point number to convert it to an integer, since this is required by FORTRAN. However, one operation which occurs frequently in programs is not supported by hardware, namely convert from single integer to double precision floating point. This was probably a mistake.