

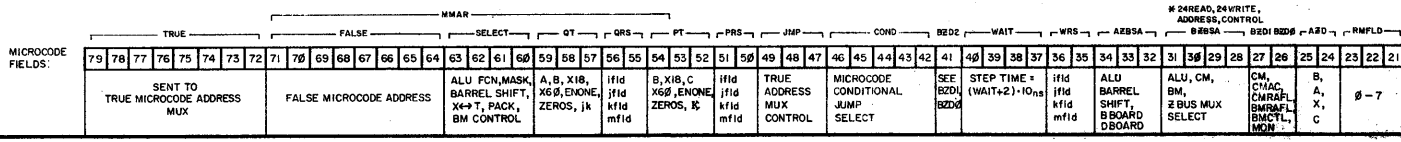
ZEPHYR HARDWARE MANUAL
SECOND EDITION



Lippold Haken & Sarah Williams

CERL

UNIVERSITY OF ILLINOIS



**COMPUTER-BASED EDUCATION
RESEARCH LABORATORY**
UNIVERSITY OF ILLINOIS
BB-028-4

Zephyr Hardware Manual

Lippold Haken and Sarah Williams

**September 1989 (Second Edition)
Computer-based Education Research Laboratory
University of Illinois
Urbana, Illinois**

**Copyright © October 1988, September 1989
by the Board of Trustees, University of Illinois**

DEDICATION

We dedicate this manual to Donald Bitzer, whose vision, fortitude, and limitless optimism has inspired us on many long days; and to Don Lee, whose preliminary Zephyr design has evolved into a functioning mainframe.

ACKNOWLEDGEMENTS

The introduction of this manual describes the purpose of the Zephyr, a hardware design which is part of a much larger project undertaken by the University of Illinois Computer-based Education Research Laboratory. The staff of the laboratory has created a computer system and network unlike any other. This was only possible because of the long hard work of many individuals, people strongly dedicated to the users of their system. This dedication to users may be seen in the long overtime hours put in by both hardware and software experts whenever emergencies arise; it may be seen in the care taken to provide excellent on-line consulting and documentation; and it may be seen in the new capabilities which are added to the system in response to users' needs.

This manual was started in July of 1988. Parts of this manual are based on documents which were written earlier: Section 1.8 and Chapter 9 are based on Mark Zvilus's thesis [3], Doug Hahn's thesis [4] was used in writing Chapters 7 and 8, and Section 1.1 is based on a document by Mike Davis [5]. Other references may be found in Appendix 3.

The following is a list of people who were directly involved in the development of the hardware described in this manual. This list does not include those who worked on the communications network, disk system, student terminal hardware, microcomputer software, or systems software. Since this is an ongoing project, many more people will be added to this list in the future.

Director of the Laboratory:	Donald Bitzer
Preliminary Design of the Zephyr:	Don Lee
Hardware Design and Implementation:	Lippold Haken, Doug Hahn Dominic Skaperdas, Mark Zvilus Tony Schooler
Microcoding:	Mark Zvilus, Lippold Haken Doug Hahn, Don Lee
Microcode Assembler:	Don Lee
Wirelist Assembler:	Don Lee
Optimized Wire Routing Programs:	Kevin Nomura
Low-level Zephyr Systems Debugging:	Mike Walker, Chris Alix
Compass Floating Point Simulations:	Ed Carney
Peripheral Port Microcomputer Programming:	Ed Carney, Jenny Barrett David Andersen
Compass Test Programs:	Charles Bridges
Circuit Diagram Draftsmen:	Roy Lipshutz, Wayne Wilson Lora Donahue
Wirewrapping Corrections and Soldering:	Paul Balga, Tom Cortese Vera Gabbard, Daniel Eisenstein Mark Lill, Wes Marquess Bryan Holloway

TABLE OF CONTENTS

1. INTRODUCTION	1
1.1. Purpose of the Zephyr.....	1
1.2. How to use this manual	2
1.3. Notation for hardware drawings	3
1.4. Overview of the Zephyr hardware	6
1.5. Levels of languages: Tutor, Compass, Microcode.....	9
1.6. Executing TUTOR and time slices.....	10
1.7. Executing Compass and Exchange Packages.....	11
1.8. Overview of the microcode fields.....	13
1.9. One's complement numbers	16
1.10. Floating point numbers.....	18
1.11. ECL Chips, Busses, Gates	21
1.12. Wirewrapping	27
2. B BOARD.....	33
2.1. Microcode Hardware	33
2.2. CLOCK CIRCUIT	37
2.3. COMPASS INSTRUCTION PIPELINE.....	42
2.4. COMPASS PC COMPUTATION.....	46
2.5. CM ADDRESS.....	48
2.6. SPECIAL CM ADDRESS DECODE.....	51
2.7. BULK MEMORY ADDRESS CIRCUIT.....	55
2.8. TIMER.....	59
2.9. ZBUS MUX.....	59
3. A BOARD.....	63
3.1. Registers	63
3.2. REGISTER ADDRESS MUX	65
3.3. ALU and BARREL SHIFTER Inputs.....	67
3.4. ALU and BARREL SHIFTER.....	68
3.5. ALU and BARREL SHIFTER Output.....	72
3.6. X Register Test.....	73
3.7. B Register Test.....	74
4. D and E BOARDS	75
4.1. D and E BOARD CONTROL	75
4.2. MBUS COMPLEMENT.....	84
4.3. EXPONENT LATCHES	85
4.4. NORMALIZE SHIFT COUNT LATCH	85
4.5. EXPONENT COMPARATOR.....	85
4.6. ADD/SUBTRACT SELECTION	86
4.7. MULTIPLY/DIVIDE EXPONENT PROCESSING	86
4.8. MULTIPLY/DIVIDE MANTISSA PROCESSING.....	87

4.9.	ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING	91
4.10.	ADD/SUBTRACT/NORMALIZE MANTISSA LATCHES	93
4.11.	PACK/UNPACK.....	93
4.12.	POPULATION COUNT	94
5.	CM BOARD.....	95
5.1.	EVEN BANK AND ODD BANK.....	95
5.2.	ADDRESS FAN-OUT.....	95
5.3.	BM DRIVER AND BM RECEIVER.....	95
5.4.	IOCM RECEIVER	96
5.5.	COUNTER.....	96
5.6.	TOP 12 BITS MUX.....	97
5.7.	CM MUX.....	97
5.8.	ERROR CORRECTION BITS GENERATION.....	98
5.9.	ERROR CORRECTION.....	99
5.10.	ERROR FLIP-FLOP.....	99
5.11.	B BOARD RECEIVER	99
6.	IOCM BOARD.....	101
6.1.	EVEN BANK AND ODD BANK.....	101
6.2.	IO REQUEST PROCESSOR	101
6.3.	MEMORY TIMING GENERATOR	102
6.4.	ADDRESS MUX.....	102
6.5.	ADDRESS FAN-OUT.....	103
6.6.	PP ADDRESS RECEIVERS and PP OFFSET ADDERS.....	103
6.7.	B BOARD DATA RECEIVER.....	103
6.8.	ERROR CORRECTION BIT GENERATION	103
6.9.	COMBINE ERROR CORRECTION BITS	104
6.10.	LATCH/BUFFER TO CM BOARD	105
6.11.	PP DATA RECEIVER AND PP DATA SHIFTER	105
6.12.	PP QUARTER WORD SELECT.....	105
6.13.	PP LATCH/DRIVERS.....	106
6.14.	INTERRUPT BUFFER	106
7.	BULK MEMORY DATA BOARDS	107
7.1.	TTL DYNAMIC MEMORY: BANK 0 and BANK 1	107
7.2.	TTL TO ECL CONVERTERS.....	107
7.3.	ECL TO TTL CONVERTERS and RESISTORS	108
7.4.	SCRATCH PAD LATCHES.....	108
7.5.	DATA IN RECEIVER and DATA IN SHIFTER.....	108
7.6.	DATA OUT SHIFTER	109
7.7.	DATA OUT DRIVERS	109
8.	BULK MEMORY CONTROLLER BOARD.....	111
8.1.	BM REQUEST PROCESSOR.....	111
8.2.	REFRESH REQUEST COUNTER.....	111

8.3.	REFRESH ADDRESS COUNTER	112
8.4.	RECEIVERS and ADDRESS/CONTROL MUX.....	112
8.5.	SHIFTER CONTROLS.....	113
8.6.	SCRATCH PAD CLOCKS	113
8.7.	ACTIVE SCRATCH PAD FLIP-FLOPS.....	113
8.8.	DYNAMIC RAM TIMING.....	114
8.9.	FAN-OUT	115
9.	MICROCODE TO IMPLEMENT COMPASS	117
9.1.	Compass instructions.....	117
9.2.	Start-up.....	117
9.3.	Set Instructions.....	117
9.4.	60-bit Integer and Boolean Instructions.....	122
9.5.	Shift and Mask Instructions	122
9.6.	Jump Instruction	122
9.7.	Conditional Jump Instructions	123
9.8.	Return Jump Instruction	123
9.9.	Exchange Jump Instruction.....	124
9.10.	Interrupts.....	127
9.11.	Traps	128
9.12.	Bulk Memory Instructions	128
9.13.	Population Count Instruction.....	133
9.14.	Pack Instruction.....	133
9.15.	Unpack Instruction	133
9.16.	Normalize Instructions	134
9.17.	Multiply Instructions	135
9.18.	Floating Add and Subtract Instructions	137
9.19.	Double Precision Floating Add and Subtract Instructions	139
9.20.	Divide Instructions.....	139
9.21.	Program Stop Instruction	143
9.22.	Transfer X to T and T to X Instructions	144
9.23.	Transfer X to P and P to X Instructions (Breakpoints)	144
9.24.	Transfer X to C and C to X Instructions.....	145
9.25.	Add CM to X Instruction	145
	APPENDICES	147
1.	MICROCODE LISTING	147
2.	SYMBOL DEFINITION LISTING	215
3.	REFERENCES	223
	INDEX.....	225

1. INTRODUCTION

1.1. Purpose of the Zephyr

The Zephyr was designed for the NovaNETSM computer network. NovaNET is a sophisticated computer network utilizing satellite communications to connect many terminals to a central computer and memory.

The Zephyr is used to offer PLATO^R software products and system features on NovaNET. PLATO and NovaNET were created by the University of Illinois Computer-based Education Research Laboratory (CERL). PLATO has evolved from a single interactive terminal in 1960 to a nationwide network of 1400 graphics terminals in 1985. Over the past decade, CERL PLATO has delivered more than fifteen million hours of use with better than 98 percent system reliability. PLATO has long stood as the premier educational computing system.

Each Zephyr mainframe connected to NovaNET consists of up to four Zephyr CPUs, and supports up to 8000 users, 4000 of whom may be signed on simultaneously. Costs per NovaNET user can be low enough that, even with its additional communications and centralized database features, NovaNET can compare favorably in price to the use of stand-alone microcomputer workstations.

On NovaNET the software and databases are centrally stored and maintained. Since files containing user-generated information may be centrally stored, floppy disks need never be used. On microcomputers all software must be loaded by the user from floppy disks; on local area networks the responsibility for file management, backups, and maintenance is on the users.

NovaNET's computer-based educational materials includes 12000 hours of instruction in over 100 subject areas, ranging from the basic skills level to the college level. An on-line catalog organizes lessons by subject, as well as by author and title. NovaNET's courseware routers permit teachers to select lessons, structure them into fully or partly automated curricula, and assign them to students. The lessons return data on student performance to the teacher automatically.

New lessons are continually under development, and users who wish may also develop their own. On-line manuals, a language which supports sophisticated answer judging as well as graphics and general purpose computing, and real-time consultation with the system staff and other programmers aid in the development of software. Since NovaNET uses a centralized computer, it avoids the usual problems associated with distributing floppy disks to users spread around the country, it allows easy feedback from the students to the programmers, and it avoids compatibility problems between microcomputers.

NovaNET's communications features are so advanced that other multi-user systems have borrowed their format and function. Features include an easy-to-use electronic mail system, general notesfiles with system-wide and inter-system access, private notesfiles set up by individual user groups for internal use, real-time means for users to "talk" to each other and consult with the system staff, and a facility which enables distant users to link their terminals so that they can see and discuss a program as it executes.

Standard personal computers may be used as NovaNET terminals, giving the user access to all the NovaNET services. NovaNET may also be used to download and upload personal computer software, and allows programs running in the personal computers to communicate with programs running on the central computer.

1.2. How to use this manual

This manual describes hardware involved in the Zephyr CPU and BM (Bulk Memory). Chapter 1 is an introduction and gives background needed for the other chapters; chapters 2 through 6 describe each of the boards in the Zephyr CPU; chapter 7 describes the BM data boards; chapter 8 describes the BM controller board; and chapter 9 describes the Zephyr's microcode from a hardware standpoint. Appendix 1 provides a listing of the microcode which was in use at the time this manual was printed.

The manual includes a block diagram of the CPU and BM. This block diagram should be studied in detail; the reader should always follow along in the block diagram when reading any part of this manual.

The manual does not include the detailed hardware drawings for each board, nor does it include the wirewrap listings. Readers doing hardware maintenance should refer to the detailed hardware drawings only after referring to the block diagram. Those doing hardware maintenance should also refer to up-to-date listings of the microcode, and should not rely on the microcode listing provided in Appendix 1.

This manual does not discuss the communications network hardware or the communications protocol. It also does not describe any details of the software which runs on the Zephyr and its peripheral port microcomputers. The references listed in this manual, as well as the on-line Aid program, may be used for further information.

1.3. Notation for hardware drawings

This section describes notations used in hardware drawings for the Zephyr. If the reader is familiar with circuit diagrams, the explanations given here together with the F100K ECL Databook [6] should make the Zephyr hardware drawings understandable.

The hardware drawings are not the only descriptions of the Zephyr hardware. The wirewrap language and the wire listings (see §1.12) also describe the hardware.

1.3.1. Block diagram notation

An arrow into a rectangle indicates an input.

A rectangle represents a group of related chips. The function of a rectangle is usually notated inside the rectangle; if a rectangle represents chiefly one type of chip, the rectangle may be labeled with that chip type. Next to each rectangle is a page number; this is the page number in the detailed board diagrams. These page numbers allow the block diagram to act as an index for the detailed board diagrams.

On the block diagram, the hardware for each board is surrounded by heavy black lines. The board name is given in the bottom right corner. When there are several identical boards, they are drawn stacked on top of each other diagonally.

[] around a board name is used to indicate that a signal goes to, or comes from, that board.

< > around a signal name indicates that a control signal comes directly from the MICROCODE LATCH. Most control signals do not come directly from the MICROCODE LATCH; they pass through a decoder first. These decoders are generally not shown on the block diagram.

/ through a line indicates it is a bus. The number next to the / indicates the number of bits in the bus. Slanted lines drawn to a bus line indicate a part of the bus "peels off"; for detailed information on which bits are peeling off, the detailed board diagrams may need to be referenced.

D is used to indicate differential signals. Differential signals are only used for connections with BM (Bulk Memory) and PPs (Peripheral Ports) (see §1.11.3).

Small circles at the inputs and outputs are used to indicate inverting inputs and outputs.

Lines over signal names are used to indicate active low signals.

Bits on a bus are numbered starting with number 00 for the least significant bit.

1.3.2. Detailed board drawing notation

The detailed board drawings are only available to those doing hardware maintenance. The detailed board drawings use the same conventions as block diagrams, except that rectangles always represent one chip or a subpart of a chip, and arrows are not used to indicate inputs

to chips. If it is not clear whether a signal is an input or an output, look up the chip in the F100K ECL Databook [6].

OR, exclusive OR, buffer, and flip-flop chips may be drawn as one rectangle, or subparts of the chip may be drawn separately, on one or more pages of drawings.

If a pin is not drawn on a chip, it is not used. Unused inverted outputs are sometimes drawn to aid in scoping while debugging the hardware. Input pins which are not connected are pulled low by internal resistors in the ECL chips; so control pins which are not on the drawings are always low.

Page numbers given in parentheses next to signals indicate other drawing pages which have one or more occurrences of the signal. If the signal appears in another place on the same page, the page number is also listed.

Signals which come from the inverted output of a chip usually have a "k" appended to the beginning of their name. This is a function of the wirewrap language (see §1.12.2). This should not be confused with active low signals, which are indicated by a line over the signal name.

Signals which are differential signals have a "B\$" appended to the beginning of their name. This is a function of the wirewrap language. See §1.11.3 for a explanation of differential signals; they should not be confused with normal twisted pairs, which consist of a signal twisted with a ground. No special notation for normal twisted pairs is used on the drawings.

Signals on a bus are named by the bus name followed by a bit number. In some cases there are several identical copies of a bus. The various copies of the bus have an additional digit appended after the bus bit number. For example, bit 34 of the Z2 bus on the B board will be called Z342.

The notation for wire ORs is shown in figure 1.11.9a.

A shorthand notation is sometimes used to indicate signals which go to several chips; see figure 1.3.2 for an example.

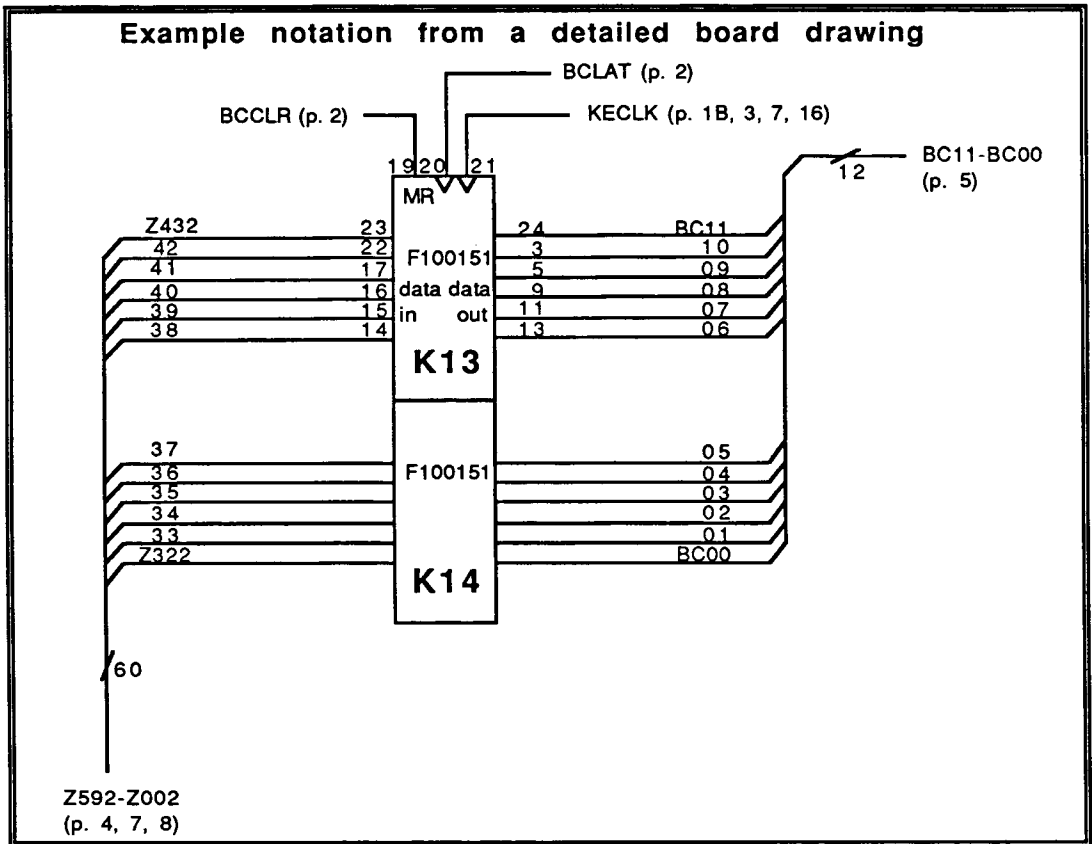


figure 1.3.2

In figure 1.3.2 BCCLR is a single signal; it is not a bus since there is no / indicating a bus, and it is drawn with sharp corners instead of slanted corners. The "p. 2" in parentheses indicates that BCCLR is connected to one or more places on page 2. BCCLR is an input signal because it is connected to pin 19, the Master Reset pin. K13 and K14 are vertically connected, and BCCLR is drawn vertically, indicating that BCCLR goes to pin 19 on both K13 and K14.

BCLAT comes from page 2 and is connected to pin 20 on both K13 and K14.

KECLK appears on 4 pages (1B, 3, 7, 16). If we assume that this figure is from page 3, then the notation indicates that this signal appears at least once more on page 3.

The name of the output bus is BC. The BC bus goes to page 5. The bus signals are named BC11-BC00. BC11 is connected to pin 24 of K13; BC10 is connected to pin 3 of K13.

BC05 is connected to pin 24 of K14. If pins are not labeled (as in the case of K14), look for the pattern on a nearby chip.

The input bus is called Z2. Since it is a 60 bit bus, only a subset (12 bits) are used by these 2 chips.

1.4. Overview of the Zephyr hardware

The Zephyr CPU consists of six wirewrap boards: the A Board, the D Board, the E Board, the B Board, the CM Board, and the IOCM Board (see the block diagram). The Zephyr BM consists of seven wirewrap boards: six BM Data Boards and one BM Controller Board.

1.4.1. The A Board

The A Board contains the ALU and Barrel Shifter. The ALU performs 60-bit integer add and subtract operations using the one's complement number system (see §1.9 for details on one's complement). The ALU also performs bit-wise logical operations on 60-bit values. The Barrel Shifter performs circular left shifts, arithmetic right shifts, and other shifting operations.

The inputs to the ALU and Barrel Shifter come from the QBUS and PBUS via two COMPLEMENT AND SIGN EXTEND circuits (see A Board on the block diagram). The output from the ALU and Barrel Shifter go to the ZBUS via a COMPLEMENT AND SIGN EXTEND circuit.

The A Board contains the general-purpose CPU registers. The QBUS has the A, B, X, and T register sets. The PBUS has the B, C, X, and T register sets (see the block diagram). Each register set contains 8 registers, numbered 0 through 7. The B, X, and T register sets for the QBUS and PBUS have identical values; register X3 on the QBUS will always have the same value as register X3 on the PBUS. The A, B, and C registers are 18 bits, and the X and T registers are 60 bits (see §3.1 for more details on registers).

The outputs of the registers go to the PBUS and QBUS. The inputs to the registers come from the ZBUS via a buffer (see the block diagram).

1.4.2. The D and E Boards

The D and E Boards contain hardware for floating point operations. The D and E Boards are used together with the A Board to implement floating point add, subtract, multiply, divide, and other operations. The floating point numbers are 60 bit quantities, including a sign bit, an 11-bit exponent, and a 48-bit mantissa (see §1.10).

Integer multiplies are performed on the D and E Boards. Unlike integer adds and subtracts, which allow 60-bit integers, integer multiplies are restricted to 48-bit integers (see §4.1 for details).

The D Board contains the POPULATION COUNT hardware; this counts the number of high (1) bits in a 60-bit value.

Input to the D Board comes from the A Board's PBUS and QBUS. Output from the D Board may go to the A Board's QBUS, PBUS, and ZBUS (see the block diagram). The E Board is used for multiplies and divides, and is connected to the D Board's MULTIPLY AND DIVIDE MANTISSA PROCESSING circuit.

1.4.3. The B Board

The B Board's CLOCK CIRCUIT provides the basic timing information for the Zephyr using a 10 ns clock. The MICROCODE circuit sequences the Zephyr's microcode; the bottom of the block diagram shows the microcode fields in each microcode word. The CM ADDRESS and BM ADDRESS circuits compute absolute Central Memory and Bulk Memory Addresses; they also check for addresses which are out of the legal range (see §1.7 for details). The SPECIAL CM ADDRESS DECODE circuit is used to control special memory-mapped functions. The TIMER is an interval timer, and the ZBUS MUX provides status information onto the B Board's ZBUS. It will be useful to learn the locations of these circuits on the block diagram.

The COMPASS PC COMPUTATION circuit computes the execution address for Compass assembly language programs, and the COMPASS INSTRUCTION PIPELINE is used to sequence through all the Compass instructions in each 60-bit word. There are up to four Compass instructions in each 60-bit word, so the COMPASS PC COMPUTATION circuit computes a memory address (18 bits) plus two more bits to select the Compass instruction within a word.

1.4.4. The CM Board

Each Zephyr CPU has 256K of CM (Central Memory). The CM Board contains half of the CM; this is private CM and cannot be accessed by any other computers. Each word in CM is 72 bits: 8 of these are error correction bits, 60 are data bits actually used by the A and B Boards, and 4 are unused data bits.

The CM Board contains hardware to generate error correction bits and hardware to perform error correction.

1.4.5. The IOCM Board

Each Zephyr CPU has 256K of CM. The IOCM Board contains half of the CM; this is shared CM and can be accessed by the A and B boards, as well as four PP microcomputers (Peripheral Port microcomputers). Each word in IOCM is 96 bits: 32 of these are error correction bits, and 64 are data bits. The A and B boards only use 60 of the 64 data bits; the PP microcomputers may access all 64 data bits, 16 bits at a time.

The IOCM Board contains hardware to generate the 32 error correction bits for data coming from the A and B Boards. The PP BUS INTERFACE cards generate error correction bits for data coming from the PP microcomputers (see the block diagram). The IOCM Board contains hardware to combine the 32 check bits into 8 check bits, which allows the use of the CM Board's ERROR CORRECTION circuit when the A and B Boards read data from IOCM. The PP BUS INTERFACE cards contain hardware for error correction when the PP microcomputers read IOCM.

1.4.6. The BM Boards

The Zephyr's BM Boards (Bulk Memory Data Boards) contain 64 megawords of memory. Each word in BM is 72 bits: 8 of these are error correction bits, 60 are data bits used by the A and B Boards, and 4 are unused data bits.

Data may be sent to BM from the CM Board or from the IOCM Board via the CM Board. Data may be sent from BM to the CM Board or to the IOCM Board via the CM and B Boards (see the block diagram). Whenever possible, BM data is transferred two words at a time; this is called a "double-word transfer". When a double-word transfer is not possible, single-word transfers are used instead.

Error correction is not performed when data is transferred from the CM or IOCM Boards to BM; error correction is also not performed when data is transferred from BM to the CM Board. During these data transfers, the error correction bits are copied along with the data bits; error correction is delayed until the data is transferred to the A and B Boards or to the IOCM board via the CM and B Boards.

1.4.7. The BMC Board

The BMC Board (Bulk Memory Controller Board) generates memory timing for the BM Data Boards, and it multiplexes addresses for the BM Data Boards. The BM is shared among four Zephyr CPUs; the BMC Board includes a PRIORITY CIRCUIT which receives requests for BM from the CPUs, and decides which request is to be granted next.

1.5. Levels of languages: Tutor, Compass, Microcode

The Zephyr uses 3 levels of computer languages: TUTOR is the high level programming language, Compass is the assembly language, and the lowest level is Microcode.

When a TUTOR program is running, each TUTOR command results in the execution of many Compass instructions. When each Compass instruction is executed, this results in the execution of one or more words of microcode.

1.5.1. TUTOR

TUTOR is a high level programming language. Almost all programmers will exclusively use TUTOR to program the Zephyr. In TUTOR, the programmer writes commands which might print text or graphics on the screen, compute the sine of a number, ask the user for input, or write data to a file.

1.5.2. Compass

Compass is the assembly language for the Zephyr. Only system staff members will use Compass to program the Zephyr. In Compass, a programmer writes instructions which might add two registers, read a value from CM to a register, or transfer a block of data from BM to CM.

The condensor and the executor are two important Compass programs. The condensor "condenses" TUTOR text written by a programmer to a binary understandable by the executor. This conversion need only be done once, unless the programmer changes the TUTOR text again. The executor "executes" TUTOR binaries; it executes many Compass instructions to perform the operations specified in each TUTOR command.

1.5.3. Microcode

The microcode is the lowest level language to program the Zephyr. Only hardware staff members will use microcode to program the Zephyr. The microcode is used to do all the substeps involved in executing each Compass instruction. For example, a Compass instruction may specify "transfer data from BM to CM"; the microcode will perform this operation, starting with requesting access to BM.

Each microcode word is 80 bits wide. These 80 bits are broken into 25 microcode fields, as shown on the bottom of the block diagram.

To say a microcode word is "executing" means that it is controlling all parts of the Zephyr. The microcode word being executed is in the MICROCODE LATCH (see B Board on the block diagram). One microcode word may perform an add by simultaneously: controlling

multiplexers to select 2 registers, reading those registers, specifying add for the ALU, controlling multiplexers to select a destination register, and performing a write to the destination register.

Each Compass instruction is implemented with a sequence of microcode words; many of the Compass instructions have only one microcode word in their sequence, while others have many microcode words.

1.6. Executing TUTOR and time slices

A Zephyr mainframe consists of up to four CPUs. The CPUs communicate with each other through BM; the block diagram shows one of four CPUs connected to BM.

The CPUs may read data from BM to their private CM, process the data, and then write the modified data back to BM. To avoid having several CPUs modify the same data at the same time, interlocks are used (see §2.7.9 for details).

BM holds information needed by the CPUs. This includes information specific to each user: the user's student bank (includes the terminal number, the place within a TUTOR program, and private variables), the TUTOR binary of the user's program, and any shared variables that are common to simultaneous users of the program. (Shared variables are common in multi-user games and in some lessons.)

A CPU running the TUTOR executor (see §1.5.2) processes one user at a time. The CPU grants "time slices" to each user; most time slices are just a few milliseconds long. When a user is granted a time slice, the user's student bank, the current unit of the user's TUTOR program and any shared variables are copied from BM to CM. In CM the executor processes part of the user's program (see figure 1.6). If, within the same time slice, the user moves into a different unit of TUTOR, the new unit is copied from BM to CM. Not all of the executor is in CM at all times; some parts of the executor, called "overlays", must be copied from BM to CM when the TUTOR binary requires them.

When the user's time slice is over, the student bank and any shared variables are copied back from CM to BM. The TUTOR binary and the overlays need not be copied from CM back to BM, since they were not changed by the CPU.

**Executing TUTOR:
What is in CM when a time slice is going on**

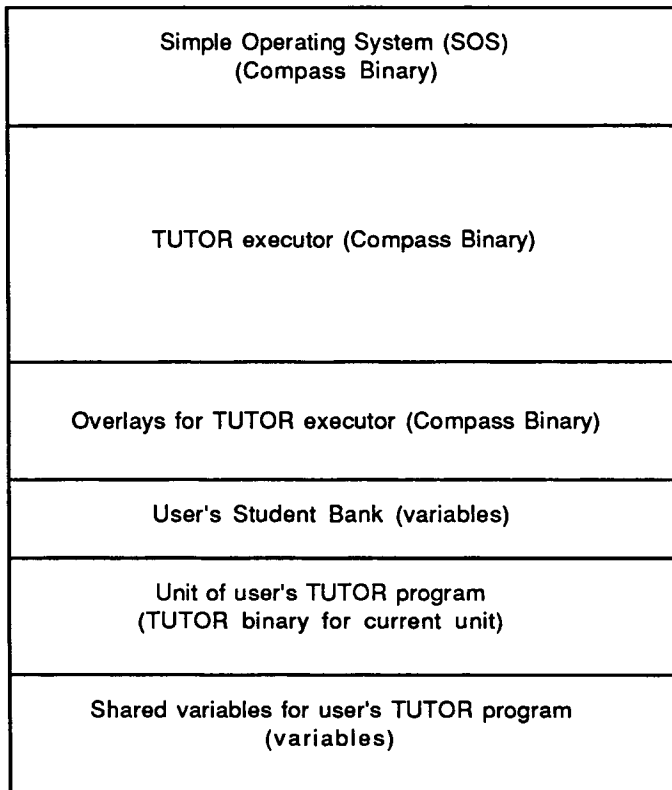


figure 1.6

1.7. Executing Compass and Exchange Packages

A Zephyr CPU may run in monitor mode or in problem mode. In monitor mode, the Compass program has access to all of BM, to all of private CM, to all of IOCM, and to all special CM addresses used in communicating with the Peripheral Port microcomputers. In problem mode, the Compass program is more restricted in what it is allowed to do. An exchange package indicates where the Compass program is to start executing, what privileges the Compass program has, and which parts of memory the program is allowed to access. The format for an exchange package is shown in figure 1.7.

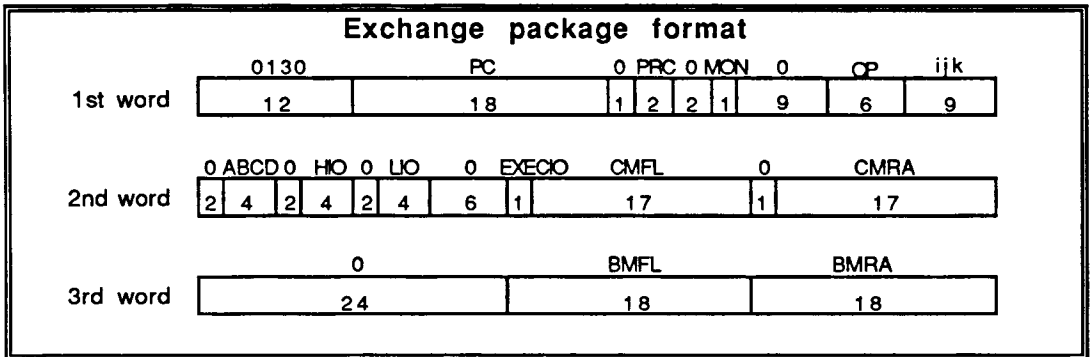


figure 1.7

The XJ Compass instruction (exchange jump) is used to invoke an exchange package. The Compass instruction for XJ is octal 0130; this is always found at the beginning of an exchange package (see figure 1.7). When a Compass program jumps to the beginning of an exchange package, the sequence of Microcode words for XJ is executed (see §9.9).

Following the 0130, the first word of the exchange package provides the PC and the PRC field; these specify all 20 bits of the Compass PC. This is where the program will start executing.

The MON field is high (1) if the program is to be in monitor mode. If the program is to be in monitor mode, then only the PC, PRC, MON, and BMRA fields of the exchange package will be used. All other fields will be ignored. The CMRA will be disabled (see §2.5.7) and no checks will be made against the CMFL, HIO, LIO, or BMFL fields. Further details on monitor mode may be found in §2.6.1. Normally programs run in problem mode, so the MON field is normally low (0).

The OP and ijk fields are only valid during interrupts and traps; these are fields from the Compass instruction which was executing when the interrupt or trap occurred (see §2.3 for details on Compass instruction fields, and §9.11 for details on traps).

The ABCD field has 4 bits: one bit for each PP microcomputer (Peripheral Port microcomputer) connected to IOCM. If a bit is high (1), this indicates that the Compass program may send interrupts to the corresponding PP microcomputer. The most significant bit in the ABCD field corresponds to PP microcomputer A (see §2.6.6 for details).

The HIO field specifies the top of the range of IOCM that this program may access. If bit 17 of a CM address is high (1) this indicates an address in IOCM; bits 16-13 of the address must be less than the HIO field. If the HIO field is 0, the program has no access to IOCM. The last part of IOCM (address bits 16-13 high) can never be accessed in problem mode;

this part of IOCM (not including the special CM addresses, §2.6.6) may only be used in monitor mode.

The LIO field specifies the bottom of the range of IOCM that this program may access. If bit 17 of a CM address is high (1) then bits 16-13 must be greater than or equal to the LIO field.

The EXECIO field is a one-bit field which gives the program permission to execute Compass out of IOCM. If this bit is low (0), then the program can read and write within the IOCM address range of HIO and LIO; however, the program may not fetch Compass Instruction Words (CIWs) from IOCM (see §2.3.3)

The CMRA and CMFL fields specify the beginning and length of private CM which the program may use. If bit 17 of a CM address is low (0) this indicates an address in private CM; bits 16-00 of the address must be less than the CMFL field, and the CMRA field will be added to the address before it is sent to CM.

The BMRA and BMFL fields specify the beginning and length of BM which the program may use. Bits 26-09 of a BM address must be less than the BMFL field, and the BMRA field will be added to bits 26-09 of the address before it is sent to BM.

1.8. Overview of the microcode fields

Each word of Zephyr microcode is 80 bits wide. These 80 bits are broken into 25 fields, as shown on the bottom of the block diagram. This section gives an overview of the functions of these fields; detailed operations of these fields will be found in the chapters of this manual which describe each of the Zephyr boards; detailed use of these fields in microcode programming will be found in Chapter 9; a microcode listing is in Appendix 1; and the numerical values for these fields are in Appendix 2.

In the microcode listing in Appendix 1, microcode words do not have values specified for all the microcode fields. The microcode assembler assumes "default" values for fields which had no values specified; in most cases the default value is all bits high (1).

1.8.1. TRUE, FALSE, COND, JMP, WAIT

The TRUE, FALSE, COND, JMP, and WAIT microcode fields are used to decide which microcode word is the next to be executed.

While a microcode word is executing, one of 32 conditions is selected by the COND field (bits 46-42 in the microcode word). If the condition is true, the address for the next microcode word will be specified by the TRUE field (bits 79-72 in the microcode word).

If the condition is false, the address for the next microcode word will be specified by the FALSE field (bits 71-64 in the microcode word).

The JMP field (bits 49-47 of the microcode word) is used to specify specially computed addresses which are used in place of the TRUE microcode field. These special addresses are computed by adding fields from the Compass instruction to an offset; they are used in selecting the proper sequence of microcode words for each Compass instruction (see §2.1.7).

The WAIT field (bits 40-37 of the microcode word) specify how long a microcode word is to execute. The value for the WAIT field should be as small as possible so that the microcode executes quickly, but it must be large enough to allow the hardware to complete its operations. The microcode listing in Appendix 1 does not have minimized WAIT field values; this process of optimizing the microcode was not completed at the time this manual was printed.

1.8.2. QT, PT, QRS, RMFLD, PRS, AZBSA, AZD, WRS, WMFLD

The QT field (bits 59-57 of the microcode word) is used to select a source for the A Board's QBUS. The source may be the A, B, X or T register set, it may be part of the Compass instruction, it may be a value from the D Board, or it may be a constant 1 (see the A Board on the block diagram). The PT field (bits 54-52 of the microcode word) is used to select a source for the A Board's PBUS. The source may be the B, C, X or T register set, it may be part of the Compass instruction, it may be a value from the D Board, or it may be a constant 1.

If the QT field specifies one of the A, B, X or T register sets, the QRS field (bits 56-55 of the microcode word) is used to select a register from within the set. The QRS field may specify part of the Compass instruction, or it may specify the RMFLD (bits 23-21 of the microcode word). The PRS field (bits 51-50 of the microcode word) performs the same function for the PBUS.

The AZBSA field (bits 34-32 of the microcode word) specifies the source for the A Board's ZBUS. The source may be the ALU, the BARREL SHIFTER, the D Board, or the B Board (see the A Board on the block diagram).

The AZD field (bits 25-24 of the microcode word) is used to select a destination for the A Board's ZBUS. This may be the A, B, X, or C register set. The WRS field (bits 36-35 of the microcode word) is used to select a register from within the destination register set. The WRS field may specify part of the Compass instruction, or it may specify the WMFLD (bits 20-18 of the microcode word). If the AZD field specifies B registers, and the WRS field specifies the WMFLD value, and the WMFLD value is 0, then the destination register

is register B0; since register B0 is always equal to 0 (see §3.1.4), this results in no change to any registers.

1.8.3. BZBSA, BZD

The BZBSA field (bits 31-28 of the microcode word) specifies the source for the B Board's ZBUS. BZBSA may specify the A Board, the CM Board (for either CM or BM data), or the ZBUS MUX (see the B Board on the block diagram).

The BZD field (bits 41, 27, 26 of the microcode word) specifies the destination for the B Board's ZBUS. Destinations include the CM Board, the CM ADDRESS hardware, the BM ADDRESS hardware, and other control functions.

1.8.4. FFPU

The FFPU field (bits 06-02 of the microcode word) selects the floating point function on the D Board. It is also used in the B Board's BULK MEMORY ADDRESS CIRCUIT .

1.8.5. ENABLE, SELECT

The ENABLE field (bits 17-14) specifies special functions for the Zephyr. These include setting control flip-flops and registers, latching new Compass Instruction Words (CIWs), and overriding the function of other microcode fields. The ENABLE field controls how the SELECT field (bits 63-60 of the microcode word) is to be interpreted.

On the A Board, the SELECT microcode field (bits 63-60 of the microcode word) is used to specify one of sixteen possible functions of the ALU and BARREL SHIFTER (see §3.4.1). It is also used to control transfers between X and T registers (see §3.1.2).

On the D Board, the SELECT field is used to control the PACK/UNPACK hardware, as well as rounding and doubling in floating point operations (see §4.1.1).

On the B board, the SELECT field is used in the BM BLOCK TRANSFER CONTROL LOGIC.

1.8.6. SPC, PCI

The SPC field (bits 11-09 of the microcode word) controls how the Compass PC (Program Counter) is to be updated. The SPC field may specify that all 20 bits of the PC are loaded from the B Board's ZBUS, that only 18 of the PC bits are loaded from the ZBUS, that 4 be added to the PC, that the PCI field (bits 01-00 of the microcode word) be added to the PC, or that the PC be updated conditionally on the result from the A Board's X or B REGISTER TEST hardware. For details, see §2.4.

If the SPC and PCI fields are not specified, the microcode assembler sets the PCI field to zero, and sets the SPC field so that the PCI is added to the PC. In this way, the PC remains unchanged.

1.8.7. SCMAD, MMAR, MEMCHK, BMWRITE

The SCMAD field (bits 13-12 of the microcode word) specifies the source for the CM address. SCMAD may select the CMAC (CM Address Counter), the Compass PC (for fetching CIWs), or the MAR latch. The MAR is loaded from the MMAR field (bits 71-54 of the microcode word) under control of the ENABLE field. The MMAR field overlaps with other microcode fields; it is the only microcode field which overlaps with others. Microcode words which use the MMAR field are limited in what else they can do.

The MEMCHK field (bit 08 in the microcode word) is used to indicate that a valid memory access is completing during this microcode word, and that the CM Board's ERROR FLIP-FLOP and the B Board's ERROR LATCH should store information on any single or double bit errors. This field is active both during CIW fetches and CM reads.

The BMWRITE field (bit 07 of the microcode word) is used during transfers of data between BM and CM. It indicates whether the BM is being read or written.

1.9. One's complement numbers

The Zephyr is a one's complement machine. This means there are two representations for 0: positive zero and negative zero. Figure 1.9a shows some examples of 18-bit one's complement numbers given in decimal and octal. To negate a number in one's complement all bits are inverted.

Decimal	Octal
+ 3	000003
+ 2	000002
+ 1	000001
+ 0	000000
- 0	777777
- 1	777776
- 2	777775
- 3	777774

figure 1.9a

When doing adds in one's complement, the carry-out from the adder is called the end-around-carry. The end-around-carry must be added to the output of the adder to get the final result, as shown in figure 1.9b. In the Zephyr ALU, adds and subtracts are done on inverted operands; in other words, the data is inverted before it is added, and then the result is inverted again. This is done to get a positive zero result when adding a value and the negative of the value (see the third example of figure 1.9b).

	Decimal	Octal	
Example #1	1	000001	
	<u>+2</u>	<u>+000002</u>	
	3	000003	output from adder
		<u>+ 0</u>	end-around-carry=0
		000003	final result
Example #2	- 3	777774	
	<u>+6</u>	<u>+000006</u>	
	+ 3	000002	output from adder
		<u>+ 1</u>	end-around-carry=1
		000003	final result
Example #3	- 3	777774	
	<u>+3</u>	<u>+000003</u>	
	- 0	777777	output from adder
		<u>+ 0</u>	end-around-carry=0
		777777	final result
Example #3, Inverting Inputs and outputs			
	- 3	000003	-3 inverted
	<u>+3</u>	<u>+777774</u>	+3 inverted
	- 0	777777	output from adder
		<u>+ 0</u>	end-around-carry=0
		777777	inverted result
		000000	final result

figure 1.9b

The logical operations provided by the Zephyr ALU are bit-wise operations. The ALU performs logical operations on 60-bit quantities; figure 1.9c shows the logical operations for 8-bit quantities.

Logical Operations	
OR	10010100 <u>01110100</u> 11110100
AND	11000101 <u>10000101</u> 10000101
exclusive OR	10100100 <u>10010100</u> 00110000

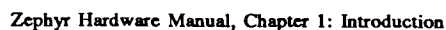
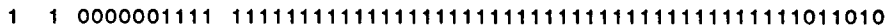
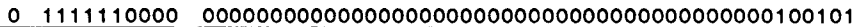
figure 1.9c

1.10. Floating point numbers

Floating point numbers consist of a mantissa, exponent, sign, and an exponent sign (see figure 1.10). When the mantissa (with an extended sign) and the exponent (with an extended exponent sign) are kept in two separate registers, this is called an "unpacked" floating point value. When the two registers are combined, this is called "packing" the floating point number. An example of packing a floating point number is shown in figure 1.10; the procedure is as follows:

- (1) Put the absolute value of the mantissa and exponent into their fields.
- (2) If the exponent sign is positive, put a 1 into the exponent sign field. If the exponent sign is negative, put a 0 into the exponent sign field and invert the exponent field. (This is like normal one's complement notation except the sign bit is inverted.)
- (3) If the sign is positive, put a 0 into the sign field. If the sign is negative, put a 1 into the sign field and invert all the other bits.

A diagram illustrating the components of a floating-point number. The number is written as $37.2 \cdot 10^{-15}$. Labels with arrows point to specific parts: 'sign' points to the first '3', 'exponent' points to the '2' in the coefficient, 'sign' points to the '-' sign, 'mantissa' points to the '37', and 'exponent' points to the '15' in the power of 10.

[illegible]

Un-normalized +1 • 2⁺⁰

OCTAL	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
BINARY	0	1	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	001
	1'	1'	10'	48'																	

Normalized 1.0 represented as +140737488355328 • 2⁻⁴⁷

OCTAL	1	7	2	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
BINARY	0	0	1111	01000	100000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	
	1'	1'	10'	48'																		

Confusion arises over the difference between underflow (a floating point value with the most significant 12 bits all equal), floating zero (a floating point value having a mantissa with all bits equal the sign bit, but with an exponent which is not overflow, indefinite, or underflow), and integer zero (all bits equal). Note that integer zeros are not interpreted as zero by the floating point machinery; rather, they are interpreted as "underflow" values. This will be important in understanding the special treatment some values receive in floating point operations (see §4.1.3 for details).

1.11. ECL Chips, Busses, Gates

This is an introduction to some of the most common ECL chips and circuitry. Readers learning just the basics of the block diagram should find this description of ECL chips adequate. Those more interested in hardware should reference the F100K ECL Databook [6].

1.11.1. Hex Latches and Hex Flip-flops (150, 151)

Both 150s and 151s store one 6-bit word, and both have a master reset pin which clears the 6 bits (sets the bits to 0). 150s acquire data whenever the two latch control signals are low, while a 151s acquire data when the OR of the latch control signals goes from low to high.

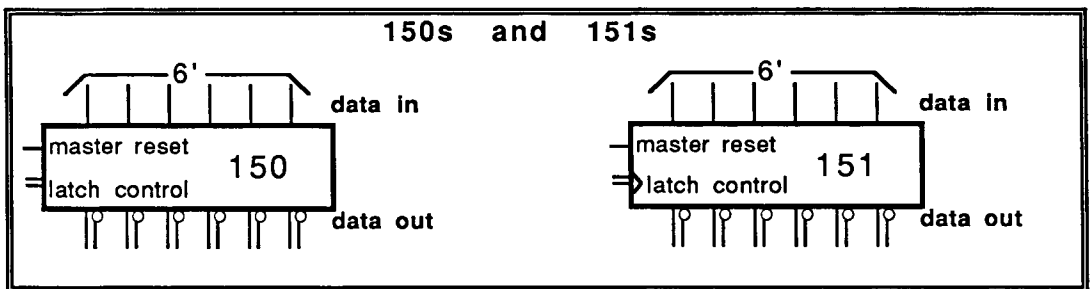


figure 1.11.1a

150s (and any other chips which have master resets) are often used as multiplexers. The outputs of several 150s may be wire-ORed; multiplexing is achieved by activating the master reset on all but one of the 150s.

On 150s and 151s, both latch controls are often used together to get a precise signal with clean edges (see figure 1.11.1b). Latch control 1 is an "imprecise" signal which determines approximately when to acquire data. Latch control 2 is a precise clock signal used to "clean up" latch control 1. If the timing between latch controls are off, glitches may occur in the OR of the two latch controls; since the OR of the latch controls is internal to the chip, such glitches may be difficult to detect.

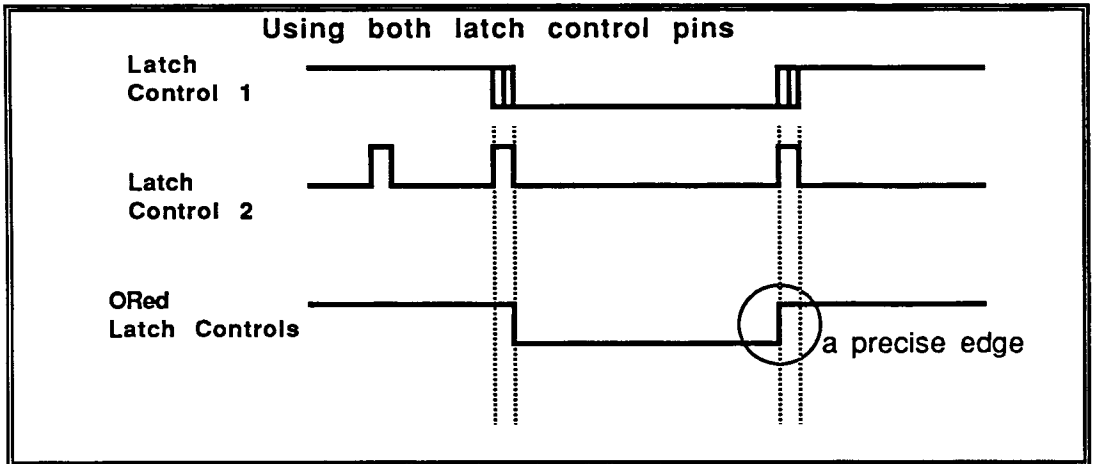


figure 1.11.1b

1.11.2. Fan-out (112, 122)

In the Zephyr hardware, an output from one chip is usually connected to 4 or fewer inputs of other chips. The limit of 4 is occasionally exceeded, but to avoid too many connections to the output of a chip, fan-out (or "buffering") is used. Fan-out is the process of making several identical copies of the same signal, as shown in figure 1.11.2a.

Most signals going to a connector or coming from a connector pass through fan-out chips. This is to protect the rest of the circuit from noise which might be acquired on the connectors.

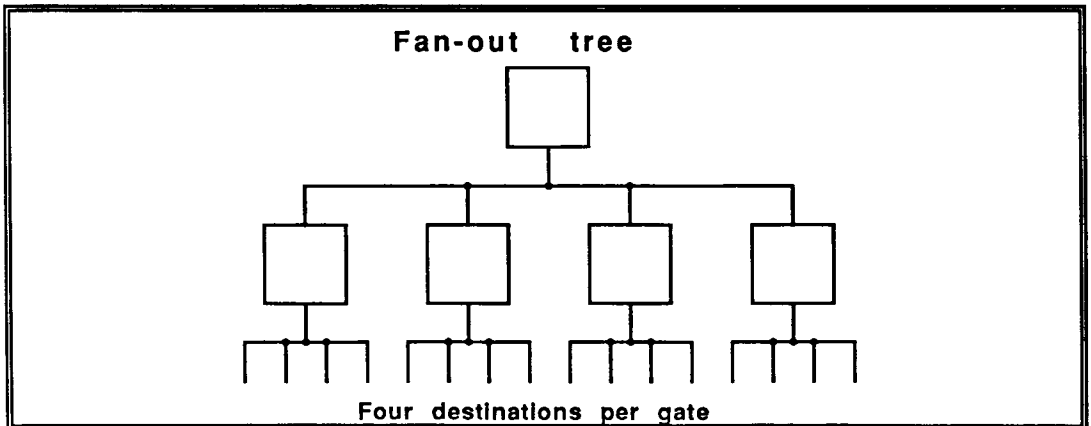


figure 1.11.2a

The Zephyr uses many different chips for fan-out. A 150 with its master reset pin unconnected (low) is transparent and could be used to implement figure 1.11.2a. 122s and 112s are commonly used for fan-out. 122s have 9 buffers, and are always transparent. 112s may be used to make 8 copies of a signal; if inverted signals are acceptable, 8 more inverted copies may be used. Also, 112s can be used to make 2 copies of each of 4 different signals carried by individual input pins (see figure 1.11.2b).

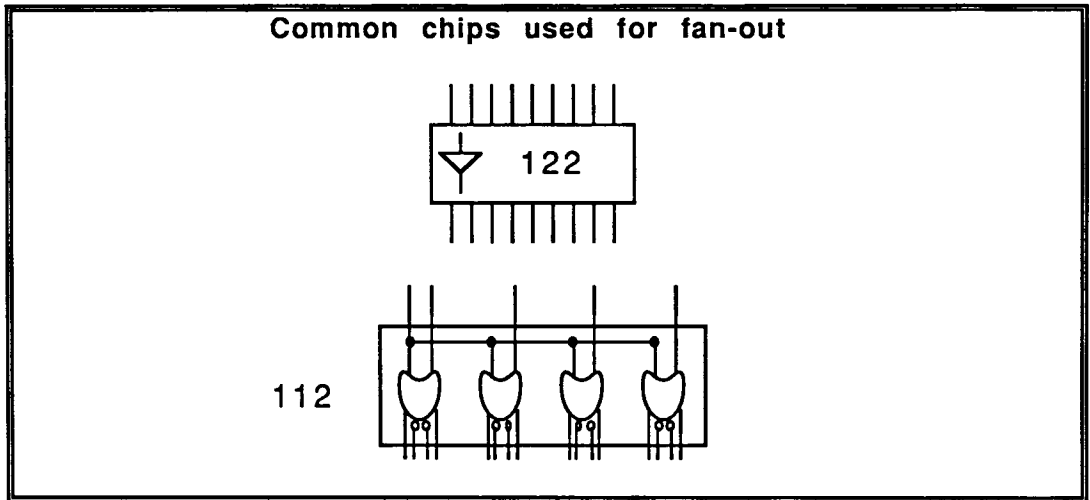


figure 1.11.2b

1.11.3. Twisted pairs and differential signals (114)

All signals shorter than 3 inches are single wires; almost all signals longer than 3 inches are twisted pairs. Most twisted pairs have one wire carrying the signal, and the other ground. Signals which go between chassis are also a twisted pair, but one wire carries the signal, and the other carries the inverse of the signal. This is called a differential signal (see figure 1.11.3).

On the block diagram, differential signals are indicated with a **D**. On the detailed board drawings, differential signals have a "B\$" appended to the beginning of their name.

114s are used with connectors between chassis. One 114 holds five differential receivers. The non-inverted input and the inverted input together are considered the same signal. The inverted output signals of 114s are usually not used.

Differential signals are commonly generated using 150s or 151s, since they have inverting and non-inverting outputs for each of the 6 data bits.

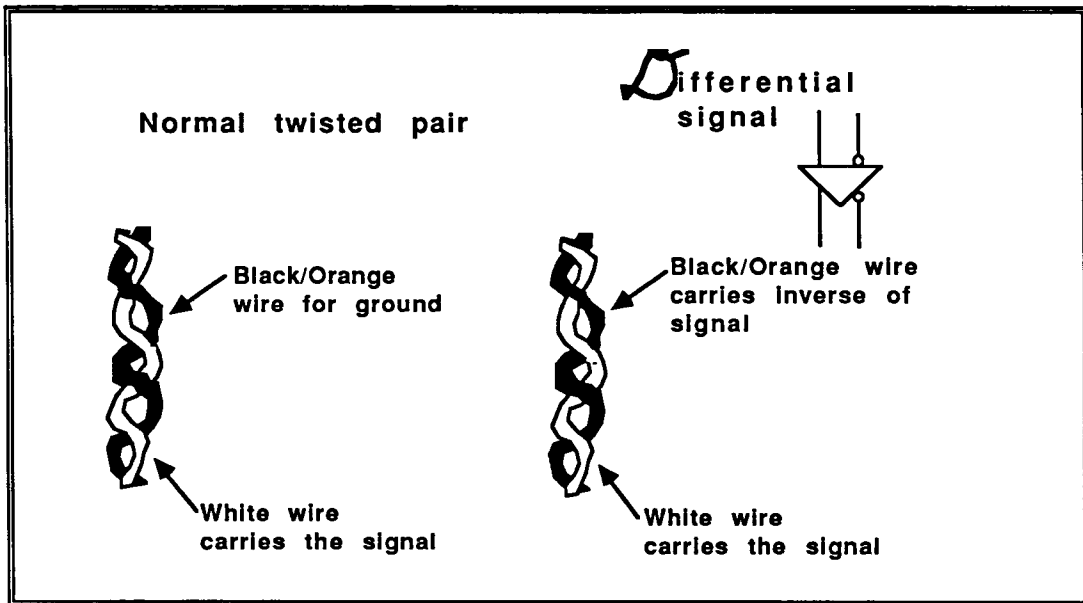


figure 1.11.3

1.11.4. Decoders

A 1-of-8 decoder will activate one of 8 output pins given an address in binary. The 170 selects one of 8 output lines by generating a high signal on one line and low signals through the others; a control pin complements all the outputs.

The 170 can function in another way: when the mode pin is not connected, the 170 changes from a 1-of-8 decoder into dual 1-of-4 decoders (see figure 1.11.4).

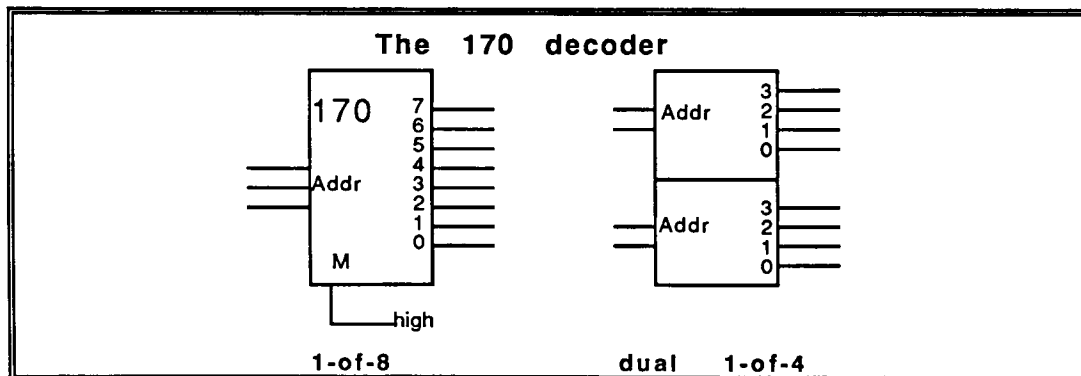


figure 1.11.4

1.11.5. Individual Latches and Flip-flops (130, 131)

Both 130s and 131s store 3 bits, and both have individual control over each of the three bits. They have latch control pins similar to 150s and 151s; they have a master set pin as well as a master reset pin; and they have separate latch control pins, set data pins, and clear data pins for each bit.

1.11.6. Multiplexers

There are four types of multiplexer chips:

164	16 input
163	dual 8 input
171	triple 4 input
155	quad 2 input with latches

1.11.7. TTL/ECL voltage levels (124, 125)

Most of the Zephyr is built with emitter coupled logic (F100K ECL) chips. These chips are fast and have few noise problems. But the BM Boards, the E Board and the PP hardware use transistor-transistor logic (TTL and CMOS) chips. These are low cost chips and some perform very complex functions.

The TTL chips require +5 volt power supplies. The voltage level for a high (1) signal is about 4 volts, and the voltage level for a low (0) signal is about 0 volts. TTL chip data input pins may be directly connected to ground for a low voltage, and directly connected to +5 volts for a high voltage.

The ECL chips require -5 volt and -2 volt power supplies. The voltage level for a high (1) signal is about -1.2 volts, and the voltage level for a low (0) signal is about -1.8 volts. ECL chip input pins may be left unconnected for a low voltage. The inputs may not be directly connected to any power supply; a constant high input voltage is generally supplied by the inverting output of an OR gate which has its inputs unconnected. These constant high voltages are called KUP or KHIGH in the detailed diagrams.

124s take TTL input and output ECL signals. 125s take ECL input and output TTL signals. Both 124s and 125s give non-inverted and inverted output.

1.11.8. OR/NOR gates (101, 102)

101 chips have three independent 5-input OR gates, with inverting and non-inverting outputs. 102 chips have five independent 2-input OR gates, with inverting and non-inverting outputs.

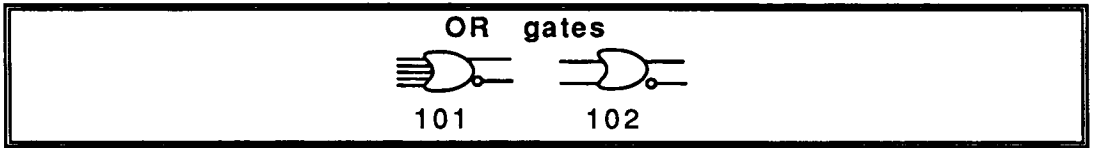


figure 1.11.8a

There are no AND gates in the F100K ECL chip series, so the AND function must be implemented with OR gates. To make an AND function using OR gates, invert the inputs and invert the outputs of an OR gate, as shown in figure 1.11.8b. If an inverted output signal is available from the source of the signal (say a 150), use that inverted output pin and avoid the extra NOR gate used to invert the signal.

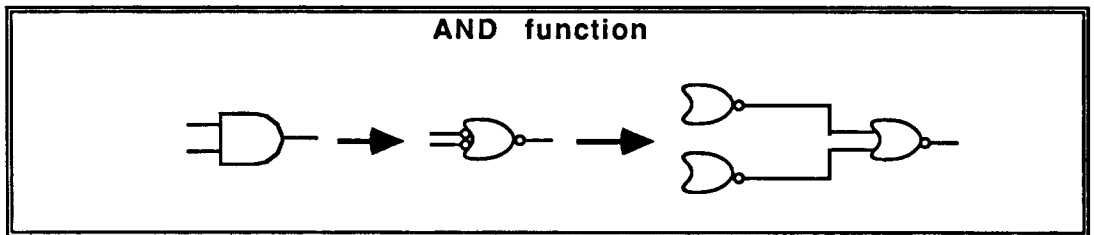


figure 1.11.8b

1.11.9. Wire ORing and busses

In figure 1.11.9a, the OR gate laid over the junction of two signals indicates the resulting OR function. When two output pins on a chip are wired together, the signals are ORed. Buffers are used in wire ORs to keep the three signals separate in figure 1.11.9a. Real OR gates are usually preferred over wire ORing, because the signals settle in a more predictable way.

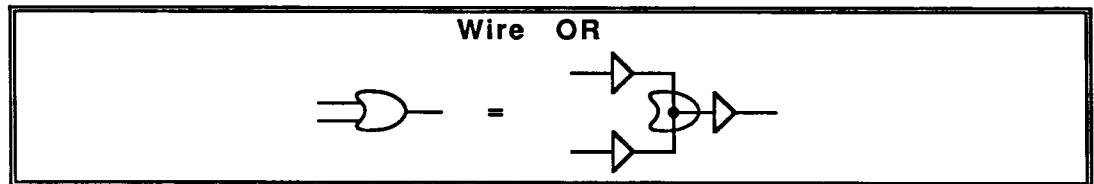


figure 1.11.9a

A bus has many sources and many destinations. In figure 1.11.9b, the 112 is an example of a destination and the 150s are examples of sources. Busses use a special case of wire ORing where only one output is active at a time, and all the other wire ORed outputs are low (inactive). In this example, the MR pins of the 150s are used to control which source is active (see §1.11.1).

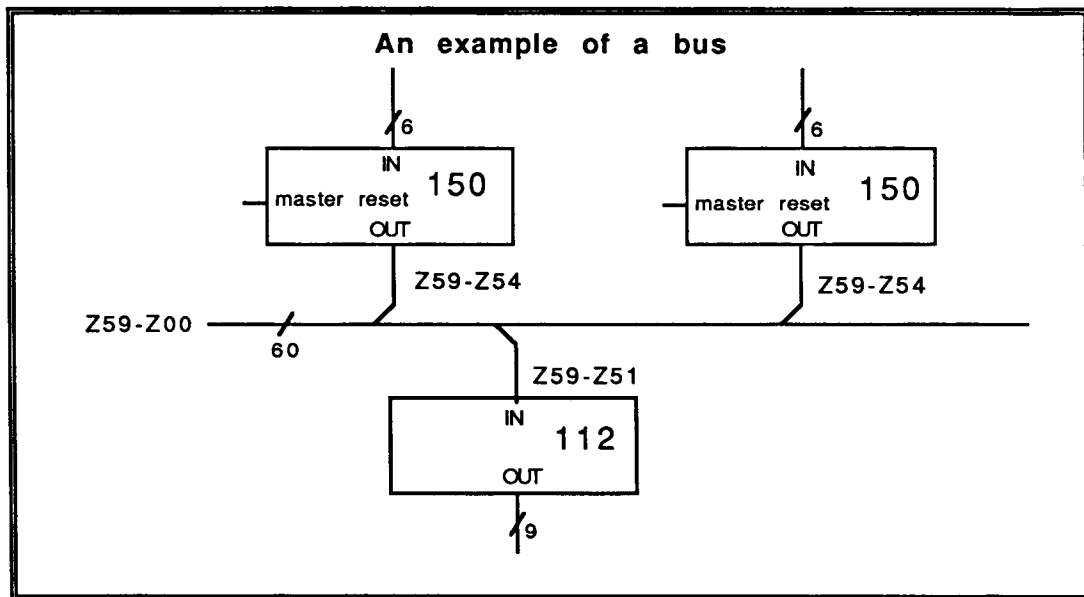


figure 1.11.9b

1.12. Wirewrapping

A wirewrap language is used to describe the Zephyr board drawings. This language is sometimes called TASP. This description of how chips are connected, kept in on-line computer files, must be updated after each change to the drawings. From these files a wire listing is generated that lists a board's pins and the signals which go to and from each pin. The wire listing contains the shortest possible connections between the pins, and it includes automatically assigned termination resistors for all ECL signals.

The wire listing is sent to a wirewrap company. The wirewrapping company wraps the boards, and produces further wire listings sorted in different ways: by signal name, by chip location and by wire number.

If mistakes were made in the design of the board, in typing the wirewrap language description of the board, or by the wirewrapping company, then they must be fixed by hand. The next sections will give some of the background required to do hand modifications, and an introduction to the wirewrap language.

1.12.1. Wirewrapping rules

Any wire longer than 3 inches must be a twisted pair. Grounds are only used with twisted pairs, and are never connected to chip input or output pins. When attaching a twisted pair, the white (orange) wire is wrapped to a chip pin while the black wire is wrapped to a ground just beneath that pin (see figure 1.12.1).

Sip pins are termination resistors pins, and are the last destination for each signal. They are 100 ohm resistors to -2 volts, and prevent reflections at the end of the line. If the line has multiple sources (e.g. wire OR lines or bus lines) then both ends of the signal must have sips.

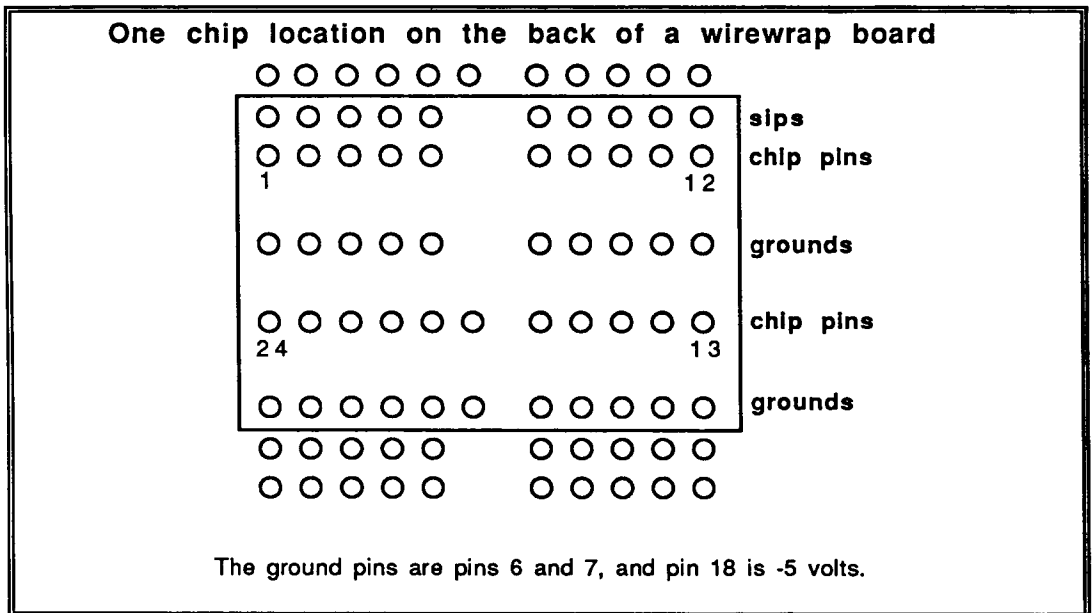


figure 1.12.1

1.12.2. Wirewrap language examples

Three examples will be given as an introduction to the wirewrap language. Consult the detailed board drawings corresponding to each of these examples; anyone interested in the wirewrap language should begin by studying these examples in detail. The wirewrap language is implemented with macros; the on-line definitions of the macros should be checked for information for each type of macro, and existing wirewrap board descriptions may serve as further examples. When changing the wirewrap files, be sure to compile the files and carefully check the output with drawings.

This language is oriented towards chips, not towards signals. In other words, all connections to a chip are listed in one place; the same signal name may appear in many different places in a file.

The language provides ways to describe individual chips, as well as ways to describe many chips of the same type which are connected to a wide bus. The first and third examples show descriptions for many chips of the same type, and the second example shows a description of a single chip.

F180 example	
LOCN	M01,M02,M03,M04,M05,M06,M07,M08,M09,M10,M11 ,,M12,M13,M14,M15,M16
GENSIG	AC15,AC14,AC13,AC12,AC11,AC10,AC09,AC08
GENSIG	AC07,AC06,AC05,AC04,AC03,AC02,AC01
F180	A=ADDA,B=ADDB,F=ADDO,BITS=96,PRE=A,SQ=10
CHECK	

figure 1.12.2a

In figure 1.12.2a, LOCN is followed by a list of chip locations on the wirewrap board, to be used by the F180 macro which follows. The comma at the start of the second line means that it is a continuation of the previous line.

GENSIG is used to list signal names that are "forward references", in other words, signal names that are used as an input to this F180 macro but will not be defined until later.

The F180 macro defines 16 chips that can be found on page 11 of the detailed D Board drawings; these are type 180 adder chips. BITS=96 means that each bus has 96 bits. The ADDA bus goes to the A inputs of the adder chips, the ADDB bus goes to the B inputs of the adder chips, and the ADDO bus comes from the F output of the adder chips (see the F100K ECL Databook[6]). PRE=A sets the prefix to A for the input and output Carry Look Ahead Signals (see page 11 of the detailed D Board drawings).

CHECK is used to indicate that all the chip locations specified in the previous LOCN lines should have been used up by now; if not, the wirewrap language assembler will give an error.

G101 example	
LOCN	A16
G101	DA=(PP1,PP2,PP31),DA=TEC1,DB=(PP1,PP3,PP4), _DB=TEC2,DC=(ROUND,KPP51),DC=XTTLC,SQ=10

figure 1.12.2b

Figure 1.12.2b shows the wirewrap language description of a single 101 OR gate chip. The chip is at location A16 on the wirewrap board; this is on page 10 of the detailed D Board drawings.

Using the notation in the F100K ECL Databook, DA refers to the OR gate A inputs: PP1 goes to pin 12, PP2 goes to pin 13, PP31 goes to pin 14. The output of A, through pin 11, is TEC1. Pin 10 is the source for KTEC1. The K in front of the signal name means that KTEC1 came out of an inverted output pin. It does not mean that this signal is active low. On the drawings, an active low is indicated by a bar over the signal name. DB is the input for gate B. OB is the output for gate B. Signals TEC2 and KTEC2 come from pin 8 and 9 respectively. DC=(ROUND, KPP51) means that ROUND and KPP51 are input signals for gate C. The output signals are XTTL and KXTTL.

SQ=10 sets the sequence number to 10. This means that none of the non-inverting outputs of this chip are wired ORs or busses (i.e. all have this chip as their sole source); this chip will be at one end of the signal, all the destinations of this signal will be connected in the most efficient linear order, and then the line will terminate in a termination resistor. SQ=20 would indicate wire ORed signals are included in non-inverting outputs; all the non-inverting outputs would have termination resistors at both ends of the signals (see §1.12.1) and they would all have been previously listed in a GENSIG statement. KSQ=20 would indicate wire ORed signals are included in the inverting outputs; all the inverting outputs would have termination resistors at both ends of the signals. If a PRE=I were present, I's (rather than K's) would precede the names of signals on the inverting outputs.

The macro name G101 starts with a G; by convention this indicates that the macro specifies a chip one gate at a time; macro names starting with F indicate the macro defines several chips at a time. In general, G macros (with the exception of the G122 macro) assign pins in the same order as they are listed in the F100K ECL Databook. F macros start with the most significant data bits assigned to the most significant data pin on the first chip; they go through the list of chips specified in the LOCN line until all data bits are used up. If the last chip is only partially used, the most significant data bits (according to the notation in the databook) will be the ones which were used. The JACK macros (used for connector jacks) assign signals to pins according to the BIAS and PINC tags; use existing JACK macros as models when defining new ones.

Figure 1.12.2c shows an example of an F155 and a G155 macro. The chips defined by these macros are on page 12 of the detailed D Board drawings. 155s are quad 2 input multiplexers, so the F155 line describes 12 chips, or 48 multiplexers. The A inputs for the multiplexer are the COMO bus, starting with COMO47. The B inputs for the multiplexer are the same bus, but start at bit 46. The output goes to the ZBUS, starting at bit 47.

F155 and G155 example

```
LOCN      S14,S15,S16,S17,S18,S19,S20,S21,S22,S23,S24,S25
F155      A=(COMO=47),B=(COMO=46),O=(ZBUS=47),
,S=KLONOR,SQ=20,BITS=44,PRE=C,EN=KOMDP,LOADS=2
G155      A=(COMO00,COMO01,COMO02,COMO03,
,B=(KSIGN,COMO00,COMO01,COMO02),
,O=(ZBUS00,ZBUS01,ZBUS02,ZBUS03),
,S=KLONOR6,EN=KOMDP6,SQ=20,PRE=C
CHECK
```

figure 1.12.2c

The select signal starts at KLONOR1 and goes to KLONOR6. SQ = 20 indicates that these multiplexers are one of several sources to the ZBUS. PRE=C specifies that the inverting output has a C (rather than the usual K) at the beginning of each signal name; such special prefixes for the inverting outputs from chips connected to a bus are common, to avoid signal name conflicts with inverting outputs from other sources on the bus. Since the inverted outputs will not be used, they will be tied directly to termination resistors.

BITS=44 indicates the number of bits in each bus. The A inputs for the multiplexers go from COMO47 to COMO04, and the B inputs for the multiplexers go from COMO46 to COMO03. EN=KOMDP starts with KOMDP1. LOADS=2 means that each control signal goes to 2 chips before the number at the end of the signal name is incremented.

1.12.3. Wire listing

Figure 1.12.3 shows the wire listing resulting from the wirewrap language description in figure 1.12.2b. Figure 1.12.3 is taken from the "sorted by chip location" section of the wire listing. These listings will vary in format depending on which wirewrap company generated the listing; this is a typical example from Augat Wiring Services.

"L" stands for level on the pin. Each pin has up to three wiring levels, the first being closest to the board. "LNTH" refers to the length of the wire connection, in inches. The last column describes the type of wire: AW=normal, TW=twisted pair.

Figure 1.12.3 shows that pin 8 of A16 is connected with a 2.1 inch wire at level 2; and KPP51 goes from pin 24 to A16 sip 1 on level 1 and to D19 pin 10 on level 2.

Wire Listing

(by chip location)

WIRE # FROM PIN	:L: TO PIN	:L: SIGNAL	:LNTH:TYPE
1541: A 16 O04	:2: B 16 O16	:2: XTTLCL	: 2. 3:AW
3778: A 16 O05	:2: A 16 S05	:2:KXTTLCL	: 0. 1:AW
1542: A 16 O08	:2: B 16 O14	:2:TEC2	: 2. 1:AW
2239: A 16 O09	:2: A 16 S11	:2:KTEC2	: 0. 3:AW
2240: A 16 O10	:2: A 16 S12	:2:KTEC1	: 0. 3:AW
1764: A 16 O11	:2: B 16 O21	:2:TEC1	: 1. 1:AW
36: A 16 O12	:1: A 16 O17	:1:PP1	: 0. 8:AW
1293: A 16 O12	:2: C 16 O13	:2:PP1	: 4. 2:TW
1294: A 16 O13	:2: C 16 O11	:2:PP2	: 4. 2:TW
1314: A 16 O13	:1: C 16 O16	:1:PP2	: 2. 7:AW
1300: A 16 O14	:1: B 16 O12	:1:PP31	: 1. 9:AW
997: A 16 O14	:2: C 20 O23	:2:PP31	: 6. 2:TW
36: A 16 O17	:1: A 16 O12	:1:PP1	: 0. 8:AW
2609: A 16 O17	:2: B 16 O23	:2:PP1	: 1. 0:AW
1187: A 16 O19	:2: C 16 O09	:2:PP3	: 4. 7:TW
1869: A 16 O19	:3: C 16 O17	:3:PP3	: 4. 2:TW
1283: A 16 O20	:2: C 16 O05	:2:PP4	: 4. 2:TW
1166: A 16 O20	:1: C 16 O22	:1:PP4	: 2. 8:AW
1408: A 16 O23	:3: E 08 O19	:3:ROUND	:13.7:TW
333: A 16 O23	:2: E 25 O04	:2:ROUND	:13.7:TW
204: A 16 O24	:1: A 16 S01	:1:KPP51	: 0. 7:AW
757: A 16 O24	:2: D 19 O10	:2:KPP51	: 8. 2:TW
1293: A 16 G12	:2: C 16 G13	:2:PP1RT	: 4. 2:TW
1294: A 16 G13	:2: C 16 G11	:2:PP2RT	: 4. 2:TW
977: A 16 G14	:2: C 20 G23	:2:PP31RT	: 6. 2:TW
1187: A 16 G19	:2: C 16 G09	:2:PP3RT	: 4. 7:TW
1869: A 16 G19	:3: C 16 G17	:3:PP3RT	: 4. 2:TW
1283: A 16 G20	:2: C 16 G05	:2:PP4RT	: 4. 2:TW
1408: A 16 G23	:3: E 08 G19	:3:ROUNDRT	:13.7:TW
333: A 16 G23	:2: E 25 G04	:2:ROUNDRT	:13.7:TW
757: A 16 G24	:2: D 19 G10	:2:KPP51RT	: 8. 2:TW
204: A 16 S01	:1: A 16 O24	:1:KPP51	: 0. 7:AW
3778: A 16 S05	:2: A 16 O05	:2:KXTTLCL	: 0. 1:AW
2239: A 16 S11	:2: A 16 O09	:2:KTEC2	: 0. 3:AW
2240: A 16 S12	:2: A 16 O10	:2:KTEC1	: 0. 3:AW

figure 1.12.3

2 . B BOARD

2.1. Microcode Hardware

There are 256 microcode words in MICROCODE MEMORY. Each word is 80 bits wide. The address in MICROCODE MEMORY comes from the multiplexers to the right of the MICROCODE MEMORY on the block diagram. Microcode addresses are written in hexadecimal.

2.1.1. Loading MICROCODE MEMORY

At start-up, the MICROCODE MEMORY must be loaded from a microcomputer connected to the LOADER Board. The microcode is sent from the microcomputer to the LOADER Board and then on to the B Board.

The LOADER microcomputer supplies the microcode address while microcode is being loaded. The LOADER microcomputer writes 16 bits of data at a time into an 80-bit latch on the LOADER Board. Then the 80 bits are stored into MICROCODE MEMORY.

The LOADER microcomputer may read the MICROCODE MEMORY: 80 bits at a time are sent to the LOADER Board, and the LOADER Board has a multiplexer which chooses 16 bits at a time.

2.1.2. Loader Controls

The microcomputer connected to the LOADER Board may dictate in which mode — single-step or run mode — the Zephyr operates. The LOADER Board also allows resetting of the Zephyr (initializing flip-flops within the computer) and it gives various controls for loading MICROCODE MEMORY.

These are the control signals generated by the LOADER Board's decoder:

- (1) Execute one instruction in single-step mode (see §2.1.5)
- (2) Go to run mode (see §2.1.6)
- (3) Go to single-step mode
- (4) Microcode addresses are to be computed by the microcode hardware (see §2.1.4)
- (5) Microcode addresses are to come from the LOADER microcomputer (see §2.1.1)
- (6) Reset
- (7) Write data from LOADER board to MICROCODE MEMORY (see §2.1.1)

2.1.3. MICROCODE LATCH

The MICROCODE LATCH holds the microcode word which is being executed. To say that a microcode word is being executed means that it is controlling all parts of the computer.

2.1.4. Starting-Up

After MICROCODE MEMORY is loaded, the LOADER microcomputer must load the first microcode word into the MICROCODE LATCH. In this operation the LOADER microcomputer supplies the first microcode address. After this initial word, the microcode address is not supplied by the peripheral computer. Instead, it is computed by microcode hardware. The 150 which receives the microcode address from the LOADER is deactivated by its Master Reset (see the block diagram).

2.1.5. Single-step mode

When the Zephyr is in single-step mode, one microcode word is executed at full speed. Then the Zephyr waits for a signal from the LOADER before it executes the next microcode word.

2.1.6. Run mode

In run mode the Zephyr executes each microcode word at full speed. Then, when the execution is completed, the Zephyr immediately goes on to the next microcode word.

2.1.7. Microcode Address

The microcode address is computed by microcode hardware using several multiplexers:

The COND field (bits 46-42 of the microcode word) controls the CONDITION MUX. The 32 inputs to the CONDITION MUX report on the status of various parts of the computer. One of the inputs to the CONDITION MUX is a constant "true." The COND field chooses which condition to pay attention to.

The output of the CONDITION MUX indicates a choice between a true and false address. It is a one-bit signal that is used to control the TRUE/FALSE MUX.

The TRUE/FALSE MUX chooses between the true address and false address under the control of the signal sent from the CONDITION MUX. The false address comes straight from the MICROCODE LATCH's FALSE field (bits 71-64 of the microcode word). The true address comes from the TRUE ADDRESS MUX.

The TRUE ADDRESS MUX is controlled by the JMP field (bits 49-47 of the microcode word). The JMP field selects one of these TRUE ADDRESS MUX inputs (listed by JMP field value):

(0) "NEXTOP BIASED BY C0H" (NEXTOP JUMP)

The true address is C0 hex plus the NEXTOP field from the PARCEL SELECT MUX. This is used when a new Compass instruction is to be executed. The NEXTOP field from the PARCEL SELECT MUX is the Compass instruction number (the 6 most significant bits of the instruction) for the next Compass instruction.

The sequence of microcode words for each Compass instruction starts at address C0 hex plus the instruction number. Compass instruction 0 executes the sequence starting in MICROCODE MEMORY address C0; Compass instruction 1 executes the sequence starting in MICROCODE MEMORY address C1.

The INTERRUPT OVERRIDE circuit will change NEXTOP JUMPs to INTERRUPT JUMPs if an interrupt is pending and the program is running in problem mode (see §2.1.8). Since INTERRUPT JUMPs require a WAIT field value of 3 or more, NEXTOP JUMPs require a WAIT field value of 3 or more (see §2.2.1).

(1) "i FIELD BIASED BY B8H"

The true address is B8 hex plus the i field from the PARCEL REGISTER. The i field is part of the Compass instruction being executed (see §2.3.1). This is used for Compass instructions numbers 50-57 octal; these instructions must execute different microcode words, depending on the i field value (see table 9.1).

(2) "i FIELD BIASED BY B0H"

The true address is B0 hex plus the i field from the PARCEL REGISTER. This is used for Compass instruction number 01 octal; this instruction must execute different microcode words, depending on the i field value (see table 9.1).

(3) "INTERRUPT VECTOR BIASED BY 80H"

The true address is 80 hex plus the IV field (Interrupt Vector field). The IV field specifies the highest priority device with a pending interrupt. 80 hex plus IV is the address in MICROCODE MEMORY where the sequence of microcode words starts for that interrupt device (see §2.1.7(0) and §2.1.8).

(4) "TRUE ADDRESS" Microcode Field

The true address is the TRUE ADDRESS microcode field (bits 79-72 of the microcode word).

(5) "NEXTOP BIASED BY C0H -- NO INTERRUPT"

The true address is C0 hex plus the NEXTOP field from the PARCEL SELECT MUX. Unlike the "NEXTOP BIASED BY C0H," the INTERRUPT OVERRIDE circuit is not active; thus this may be used with a WAIT of 2 (see §2.1.7(0) and §2.1.8).

(6) "i FIELD BIASED BY A8H"

The true address is A8 hex plus the i field from the PARCEL REGISTER. This is used for Compass instruction number 00; this instruction must execute different microcode words, depending on the i field value (see table 9.1).

2.1.8. INTERRUPT OVERRIDE

The INTERRUPT OVERRIDE hardware implements interrupts. If an interrupt is pending, if a NEXTOP jump is specified in the JMP microcode field (bits 49-47 of the microcode word), and if the program is running in problem mode (see §1.7), then INTERRUPT VECTOR BIASED BY 80H is sent to the TRUE ADDRESS MUX (see §2.1.7). Otherwise the value specified in the JMP field is forwarded to the TRUE ADDRESS MUX.

Interrupts do not occur in monitor mode; pending interrupts will be ignored until the CPU returns to problem mode.

2.1.9. Microcode Hardware Details

Detailed B Board drawings, page 3

H07 and K07 are priority encoders for the IV field (Interrupt Vector field). H07 and K07 have internal latches. These are used to keep IV stable during the last part of microcode instructions.

H10 and F04 are the INTERRUPT OVERRIDE circuit. F04 is disabled in monitor mode (MON=high).

The inputs for the CONDITION MUX are active low.

E02 has an internal latch used to keep the output of the CONDITION MUX stable during the last part of fetch of the next microcode word. E02 need not be latched as early as H07 and K07. E02 is latched on a KECLK signal which did not pass through the ECLK fan-out circuit, making it earlier than other KECLK signals.

B02 and A02 are 150s which are used to enable and to disable the microcode address from the LOADER PP. While loading, B02 and A02 are active and C02 and D02 are disabled.

Detailed B Board drawings, page 4

D01 is used to buffer the microcode address going to the LED DISPLAY. The LED DISPLAY is always showing the next microcode address, not the address being executed.

B03, C03, D03, and E03 are fan-out chips for MICROCODE MEMORY. The input to C03, as well as input data to all MICROCODE MEMORY chips, come from the LOADER PP microcomputer. The chips at the right of the drawing buffer the data out to the LOADER PP microcomputer.

Detailed B Board drawings, page 5

The clock signals for the MICROCODE LATCH are fanned-out on page 1 by L02. Whenever timing is measured in the rest of the machine, it should be compared to these clock signals at the MICROCODE LATCH. More will be said in §2.2.3.

All bits of the microcode word are loaded into the MICROCODE LATCH except for the WAIT field (bits 40-37 of the microcode word), which goes directly to the CLOCK CIRCUIT.

2.2. CLOCK CIRCUIT

The CLOCK CIRCUIT provides timing for the Zephyr. The inputs to the CLOCK CIRCUIT are:

- a 10 ns clock from the LOADER Board,
- the WAIT field (bits 40-37 of the microcode word),
- control signals from the LOADER Board, and
- control signals from the IOCM (IO Central Memory).

The CLOCK CIRCUIT generates several different signals, some of which are active high and some of which are active low. All of these signals pulse at most once during the execution of each microcode word.

2.2.1. WAIT field

The WAIT field (bits 40-37 of the microcode word) specifies how long a microcode word executes. In run mode, a microcode word stays in the MICROCODE LATCH for 10 times (WAIT+2) ns. Thus a WAIT field value of 3 indicates a 50 ns execution time. This execution time will be automatically increased if IOCM is accessed during the execution of a microcode word.

When a memory access is completed during the execution of a microcode word, the minimum WAIT value is 4 (this is 60 ns) (see §2.2.7). NEXTOP jumps require a WAIT of 3 or more (see §2.1.7(0)). In all other cases, the minimum WAIT value is 2 (this is 40 ns). Most microcode words specify WAIT field values which are larger than these minimum values, to allow all parts of the Zephyr enough time to complete their operations.

Memory accesses to private CM may occur completely during one microcode word execution or they may begin in one microcode word and finish in the next.

2.2.2. COUNTER

Within 10 ns of loading the MICROCODE LATCH with a new microcode word, the COUNTER is loaded with the new WAIT field.

Note that the WAIT field comes directly from MICROCODE MEMORY. All other control of the Zephyr from the microcode word comes through the MICROCODE LATCH.

The COUNTER has the following values during the execution of a microcode word with a WAIT of 4: 15, 4, 3, 2, 1, 0. During the first 10 ns of the execution of a microcode word the COUNTER has a value 15. During the last 10 ns of execution the COUNTER has a value of 0.

2.2.3. CLOCK LATCH

The CLOCK LATCH is an edge-triggered 151 which is updated at 10 ns intervals.

Although the outputs of the CLOCK LATCH are synchronous, large timing skews are introduced into clock signals. These timing skews are due to varying loads on the signals, varying fan-out delays, varying wire length, and, in some cases, wire loops. The clock signals at the MICROCODE LATCH should be taken as a reference that marks the beginning of the execution of the microcode word.

2.2.4. SINGLE-STEP FLIP-FLOP

The SINGLE-STEP FLIP-FLOP is controlled by the LOADER Board. The SINGLE-STEP FLIP-FLOP is used to guarantee that only one microcode word at a time is executed in single-step mode (see §2.1.5).

2.2.5. XR/S

The XR/S signal comes from the LOADER Board. It is low for run mode and high for single-step mode (see §2.1.5 and §2.1.6).

2.2.6. Clock Signals Generated by the CLOCK CIRCUIT

Figure 2.2.6 shows the various clock signals generated by the CLOCK CIRCUIT. The first timing diagram shows the clocks during the execution of a microcode word with a WAIT field equal to 4. The break in the timing diagram indicates where extra 10 ns clock cycles are inserted if an IOCM memory cycle completes during the execution of this microcode word; the extra 10 ns clock cycles are inserted until the acknowledge (UGOTIT signal) from the I/O memory has a positive edge, indicating completion of the memory cycle.

The second timing diagram in figure 2.2.6 shows the clocks during the execution of a microcode word with a WAIT field equal to 3. This diagram shows the clocks in single-step mode; the timing in this diagram is of interest when the Zephyr is being debugged in single-step mode (see §2.1.5).

KCLK is a 10 ns positive pulse. The leading edge of this pulse indicates the beginning of execution of a new microcode word. This signal is used to load:

- a new microcode word in the MICROCODE LATCH,
- a new WAIT field in the MICROCODE COUNTER,
- the Compass PC LATCH (see §2.4.1), and
- the next Compass instruction.

In single-step mode, KCLK remains low after the execution of the microcode word. The KCLK pulse occurs when the LOADER indicates that the next microcode word should be executed.

KECLK is a 20 ns positive pulse. The leading edge of this pulse is 10 ns before the leading edge of KCLK. KECLK is called "early clock" and is used on latch controls throughout the Zephyr. Since this clock occurs 10 ns before the end of the execution of a microcode word, the control signals from the MICROCODE LATCH will be valid while data is being stored. In single-step mode, KECLK goes high before the Zephyr waits for a signal from the LOADER (see §2.1.5).

Most KECLK signals pass through fan-out chips, but KECLK is used directly to latch the output of the CONDITION MUX so that it is stable during the end of the execution of the microcode word. This clock must be earlier than the usual KECLK signals, so it does not go through a fan-out chip (see §2.1.7 and §2.1.9).

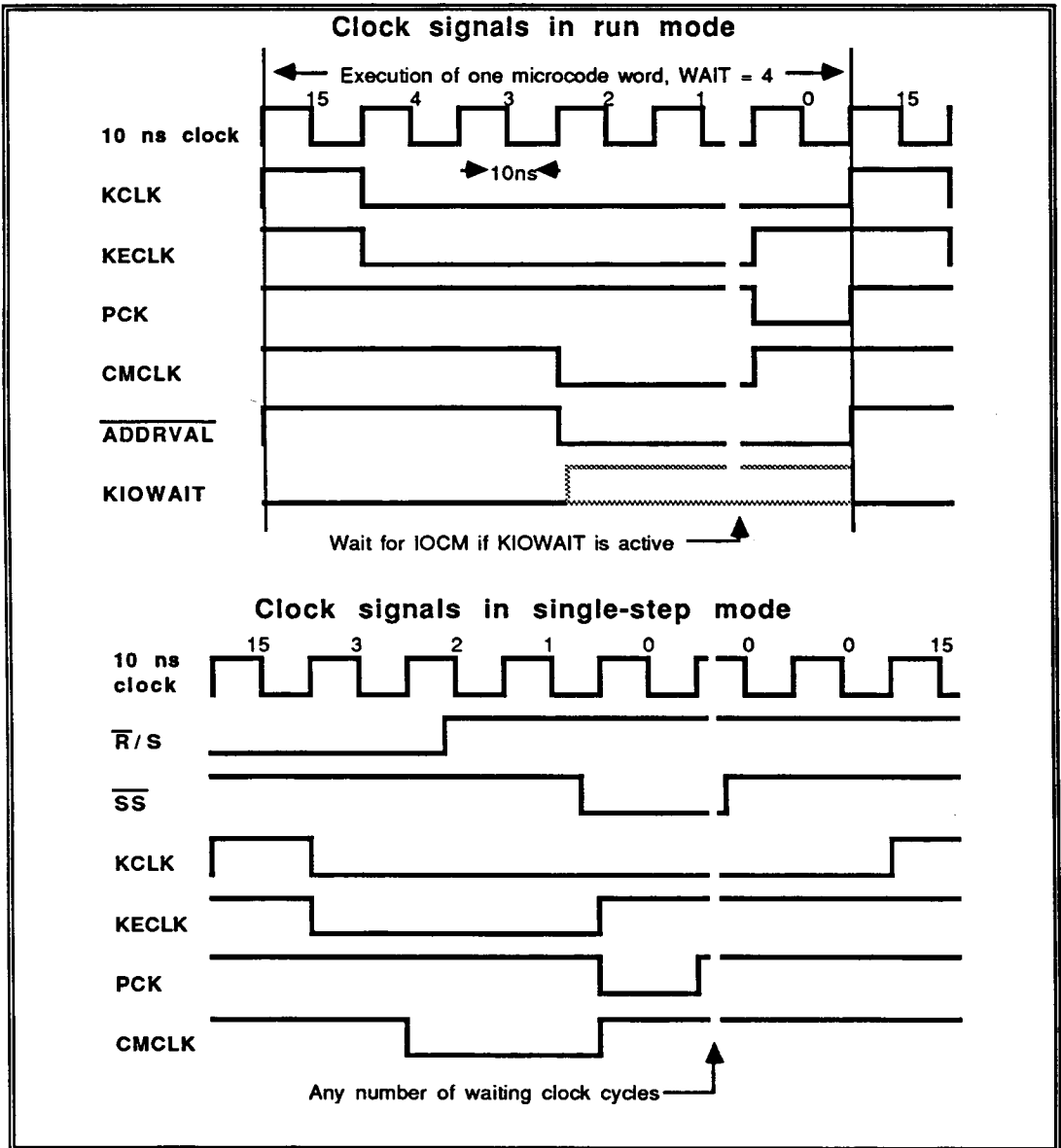


figure 2.2.6

The PCK signal is a 10 ns low pulse before the leading edge of KCLK. It is active just before execution starts on the next microcode word, when the microcode addresses have settled. It is used in latching the PARCEL REGISTER (see §2.3.1) if a new Compass instruction is being started.

CMCLK has a 20 ns wide low-going pulse which immediately precedes the rising edge of KECLK. CMCLK is used for a write pulse for CM, IOCM and registers. If the WAIT value is 2, CMCLK is only 10 ns.

ADDRVAL is active low. It is a 30 ns wide low-going pulse which immediately precedes KCLK. ADDRVAL stays high if the WAIT field is less than 4. ADDRVAL is used to indicate that the CM ADDRESS circuit has a valid address.

2.2.7. CLOCK CIRCUIT Hardware Details

Detailed B Board drawings, page 2

The SINGLE-STEP FLIP-FLOP is on A17. On the positive edge of the single-step signal from the LOADER Board, this flip-flop is cleared. It remains clear only until the next KCLK pulse, which occurs when the next microcode word begins to execute.

The IO FLIP-FLOP is on M24. When the IO Board acknowledges that it is available for CPU usage, the UGOTIT signal has a positive edge. This sets the IO FLIP-FLOP. The IO FLIP-FLOP will get cleared at the start of the execution of the microcode word (at the next KCLK pulse).

The KIOWAIT signal is used to stop the COUNTER (P22) while IOCM is being requested. The CPU waits for IOCM during the low part of CMCLK (see figure 2.2.6). The KIOWAIT signal must settle before CMCLK goes low, so a minimum WAIT field value of 4 must be used in microcode words which complete a memory access (see §2.2.1).

The LOAD signal on the COUNTER (P22) goes active during KCLK. This causes the COUNTER to go through the value 15 before the new WAIT field is loaded.

The COUNTER is loaded directly from MICROCODE MEMORY (bits 40-37 of the microcode word). This is the WAIT field before the MICROCODE LATCH.

The WLCLK signal computes the next value of ADDRVAL. Note that WLCLK is inhibited if WAIT field bits 2 and 3 are both 0 (see §2.2.1).

2.3. COMPASS INSTRUCTION PIPELINE

The assembly language for the Zephyr is called Compass. Most Compass instructions are either 15 bits (1 parcel) or 30 bits long (2 parcels). Four 15-bit Compass instructions may be put into one 60-bit Compass Instruction Word (CIW).

FORMAT FOR 15-BIT COMPASS INSTRUCTION (ONE PARCEL)

OP	i	j	k
6'	3'	3'	3'

The OP field holds the Compass instruction number.

The i field usually specifies a destination register for an operation.

The j and k fields usually specify the 2 source registers for an operation.

FORMAT FOR 30-BIT COMPASS INSTRUCTION (TWO PARCELS)

OP	i	j	K
6'	3'	3'	18'

Thirty-bit Compass instructions are much like 15-bit Compass instructions:

In 30-bit Compass instructions there are 18 bits for a K field (not 3 bits for a k field).

The K field is a signed constant.

Since a CIW is 60 bits long and is broken into 15-bit parcels, a 30-bit Compass instruction uses 2 consecutive parcels. A 30-bit Compass instruction is never split over two CIWs.

figure 2.3

2.3.1. PARCEL REGISTER

The PARCEL REGISTER holds the Compass instruction being executed, much like the MICROCODE LATCH holds the microcode word being executed.

The PARCEL REGISTER holds 30 bits (or 2 parcels). The first 6 bits of the PARCEL REGISTER are the OP field of the Compass instruction. The next 3 bits are the i field. The next 3 bits are the j field. If it is a 30-bit instruction, the next 18 bits are the K field; if

it is a 15-bit instruction, then the next 3 bits are the k field and the last 15 bits of the PARCEL REGISTER are ignored.

The PARCEL REGISTER is loaded on the trailing (positive) edge of PCK, synchronous with the MICROCODE LATCH. Unlike the MICROCODE LATCH, the PARCEL REGISTER is not loaded at every new microcode word; it is only loaded when the most significant two microcode address bits are high (1); this only occurs on NEXTOP jumps. A NEXTOP jump is always used to start a new Compass instruction (see §2.1.7).

2.3.2. PARCEL SELECT MUX

The PARCEL SELECT MUX chooses one of 4 possible instructions (parcels) contained in a CIW (Compass Instruction Word), and sends them to the PARCEL REGISTER. The NPRC (NEXT PARCEL) signals control the PARCEL SELECT MUX. NPRC tells in which parcel the next Compass instruction starts. The parcels in a CIW are numbered 0-3, with parcel 0 in the most significant bits.

When NPRC=0, parcels 0 and 1 are selected by the PARCEL SELECT MUX. Two parcels are selected in case the instruction is 30 bits long. If parcel 0 contains a 15-bit instruction, parcel 1 will not be used; after parcel 0 starts execution, parcel 1 will be selected through the PARCEL SELECT MUX with NPRC=1.

When NPRC=1, parcels 1 and 2 are selected.

When NPRC=2, parcels 2 and 3 are selected.

When NPRC=3, parcel 3 is selected.

Note that when NPRC=3 the next instruction must be a 15-bit Compass instruction.

2.3.3. CIW LATCH

The CIW LATCH (Compass Instruction Word LATCH) holds one CIW which contains up to 4 Compass instructions. The CIW LATCH is loaded from CM (Central Memory). The output of the CIW LATCH goes to the PARCEL SELECT MUX (see the block diagram).

The CIW LATCH is loaded when KPRCEN goes active. KPRCEN is activated by the ENABLE field (bits 17-14 of the microcode word). Loading the CIW LATCH is called the "CIW fetch." This normally occurs once all the instructions in the CIW LATCH have been executed.

2.3.4. Hidden CIW Fetches

CIW fetches (Compass Instruction Word fetches) may be done in parallel with other operations on the Zephyr, as long as the other operations do not require a CM access. Whenever possible, the Zephyr fetches the CIW containing the next Compass instruction from CM while the current instruction is executing. This operation is called a "hidden instruction fetch."

Hidden CIW fetches from private CM may occur completely during one microcode word execution or they may begin in one microcode word and continue for several more.

If the next Compass instruction is in the same CIW as the current Compass instruction, the microcode may perform an unnecessary hidden CIW fetch. Since hidden CIW fetches do not add to the execution time of the microcode, this is of no concern.

2.3.5. Partially Hidden CIW Fetches

Many Compass instructions are faster than a CM access. If the last Compass instruction in the current CIW (Compass Instruction Word) does not access CM, but if it is too short to do a hidden CIW fetch, then a partially hidden CIW fetch is performed.

In a partially hidden CIW fetch, the fetch of the next CIW is started during the execution of the last instruction of the current CIW. Then a special microcode word is executed to provide extra time for the CIW fetch to complete.

2.3.6. Details of Partially Hidden CIW Fetches

In writing the sequence of microcode words which implement a Compass instruction, the microcode programmer sets the WAIT field on each microcode word and knows if the microcode word does a CM access. If a microcode word does not require CM, the microcode specifies that the COMPASS PROGRAM COUNTER (the address of the next CIW) be used as the CM address. This begins the hidden fetch of the next CIW.

At this point the programmer can see how long the WAIT field is. If the WAIT field is long, there will be enough time for the CIW fetch to complete (a hidden fetch). If the WAIT field is not long, there will not be enough time to complete the fetch (partially hidden fetch).

If it is a hidden fetch *or* if it is a partially hidden fetch but there are more instructions in the current CIW, then the next Compass instruction is ready. A NEXTOP JUMP (see §2.1.7(0)) may be performed to start the sequence of microcode words for the next Compass instruction.

If it is a partially hidden instruction fetch and it is the last instruction of the CIW, then an extra microcode word is executed. This extra microcode word allows the completion of the partially hidden CIW fetch. This microcode word in turn will perform the NEXTOP JUMP to execute the first Compass instruction in the new CIW.

2.3.7. COMPASS INSTRUCTION PIPELINE Hardware Details

Detailed B Board drawings, page 7

At the top left of the drawing, KPRCEN is generated by ORing CMCLK and ENPRC. ENPRC comes from the ENABLE field decoder on page 1(B) of the detailed B Board drawings.

The CIW LATCH is transparent. It may be loaded at the same time as the PARCEL REGISTER for the first Compass instruction in the CIW.

The KPRCD signal (called VALIDCM in microcode) is only valid when the PCI field (bits 01-00 of the microcode word) is non-zero. This signal indicates if the next Compass instruction is in the current CIW, or if a new CIW must be fetched.

Detailed B Board drawings, page 8

The NEXTOP field is called I29-I24 and is the only field from the Compass instruction which does not always pass through the PARCEL REGISTER.

The PARCEL REGISTER is loaded only when NEWINST is low. NEWINST goes low only when a new Compass instruction is to be started (if the microcode address is C0 or above).

2.4. COMPASS PC COMPUTATION

The output of this circuit is the 20-bit Compass PC (Program Counter). The most significant 18 bits of the Compass PC specify the address of the CIW (Compass Instruction Word) in CM, and the least significant 2 bits specify a parcel within the CIW (see figure 2.4). The block diagram will be helpful in understanding this description.

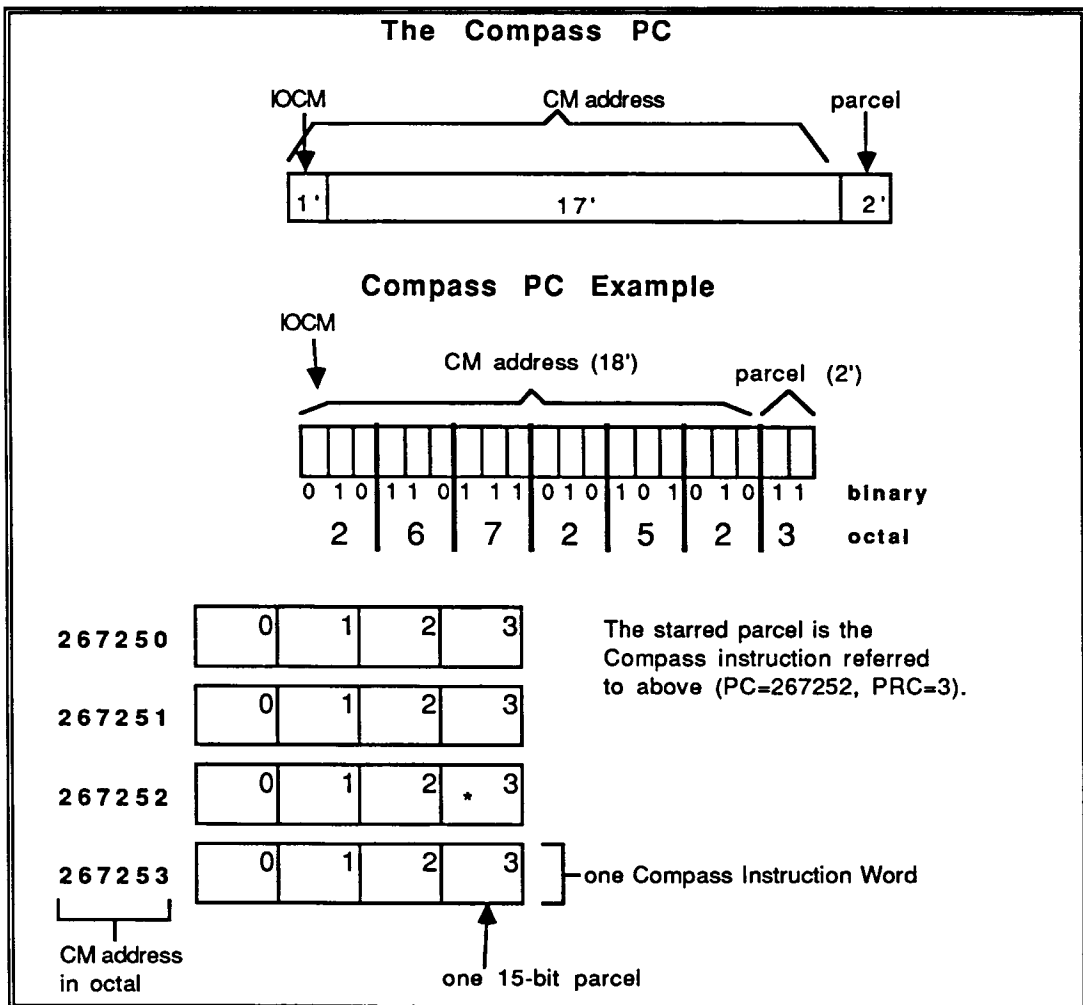


figure 2.4

2.4.1. PC LATCH

The PC LATCH holds the Compass PC and is loaded on the positive edge of KCLK. This address is displayed on the LED DISPLAY. It is displayed in octal, with the most significant 18 bits separated from the least significant 2 bits (see figure 2.4).

The sequence of microcode words for each Compass instruction updates the PC. This can be done anywhere within the sequence of microcode words, so the address on the LED DISPLAY may represent either the Compass instruction being executed or the next Compass instruction.

2.4.2. ADDER

The ADDER is used in the most common way of updating the PC. Normally the NPC (new PC) is the result of adding the current PC and the PCI (program counter increment). The input to the ADDER adds one of four possible values chosen by MUX 1:

- 0 is input to the ADDER to leave the PC unchanged,
- 1 is input to the ADDER if the Compass instruction is 15 bits long (1 parcel long),
- 2 is input to the ADDER if the Compass instruction is 30 bits long (2 parcels long),
- 4 is input to the ADDER to skip 4 parcels (one 60-bit word).

2.4.3. MUX 1

MUX 1 chooses if the value 4 or the PCI field (bits 01-00 of the microcode word) is to be passed on to the ADDER.

2.4.4. MUX 3

MUX 3 chooses between the 2 ways of computing the NPC: the normal sequential path in which MUX 1 and the ADDER are used, and the nonsequential path from the ZBUS through MUX 2. In either case, the output of MUX 3 is the 18-bit NPC and the 2-bit NPRC.

2.4.5. MUX 2

MUX 2 chooses between loading the 18-bit CM address (and placing zeroes in the extra bottom 2 bits) and loading all 20 bits (which include the 2-bit PRC).

The 18-bit CM address is used to implement Compass jumps. The PRC is not needed because jumps go to the beginning of a CIW (Compass Instruction Word).

The 20-bit address is used when returning from traps and interrupts. The Zephyr uses traps to carry out unusual floating point operations; interrupts are used to handle memory errors, the TIMER, and other functions (see §9.10, §9.11). In order to return to the correct parcel within a CIW, PRC must specify where to start executing again.

2.4.6. CONTROL

The CONTROL hardware controls the PC computation MUXes. It uses 3 input signals:

SPC (from bits 11-09 of the microcode word), and CLDPC1 & CLDPC0 for the status of registers on the A Board (see §3.6 and §3.7).

2.4.7. COMPASS PC COMPUTATION Hardware Details

Detailed B Board drawings, page 6

The CONTROL hardware is implemented with 2 MUXes: M09 and L09.

N11 is the carry look ahead for the PC ADDER. For details on its operation see the F100K ECL Databook [6].

Loading of the PC LATCH, normally done on the positive edge of KCLK, is inhibited if the microcode address is being supplied by the LOADER rather than the CPU.

NPC is sent to the CM ADDRESS circuit, the COMPASS INSTRUCTION PIPELINE, and the XJ word (see §2.9.2). NPC is the PC value before the PC LATCH. For sequences of microcode words that are one microcode word long, the use of NPC rather than PC to the CM ADDRESS circuit is important for fetching the next CIW. For longer microcode sequences the PCI is usually 0, MUX 1 usually selects the PCI, and MUX 3 usually selects the ADDER; thus, the NPC is usually equal to PC.

2.5. CM ADDRESS

The CM ADDRESS circuit has a multiplexer which chooses a CM address from one of 3 different places: CMAC, MAR, and the NPC from the COMPASS PC COMPUTATION circuit (see §2.4.4). The CM ADDRESS circuit handles both private CM and IOCM addresses.

Each Compass program has some limited area of private CM that it uses. The beginning of the area for a Compass program is called the CMRA (private CM Reference Address). The

CMFL (private CM Field Length) specifies the length of the area to which a program is allowed access (see §1.7). For accesses to private CM in problem mode, the CM ADDRESS circuit adds in the CMRA and checks that the address is within the CMFL (see the block diagram).

2.5.1. MAR LATCH

The MAR LATCH holds the MAR (Memory Address Register); this is used for memory addresses hardcoded in microcode. The MAR is used in the start-up sequence of microcode words to fetch the first CIW.

The MAR is rarely used in the sequences of microcode words which implement Compass. Its main use is for special microcode programs for debugging hardware.

The MAR is loaded from the MMAR microcode field (bits 71-55 of the microcode word). The latch control for MMAR is from the ENABLE field (bits 17-14 of the microcode word). Note that the MMAR field overlaps the FALSE, SELECT, QT, QRS and PT fields. A microcode word that uses the MMAR field can not use these overlapped fields at the same time; thus, such a microcode word is limited to loading the MAR.

2.5.2. CMAC

The CMAC (CM Address Counter) is normally used as an 18-bit latch. In Compass instructions which access BM it is used as a counter. The CMAC is loaded from the ZBUS. The value on the ZBUS may come from the ALU or from elsewhere in the Zephyr.

2.5.3. CM ADDRESS MUX

The CM ADDRESS MUX chooses from one of 3 sources of the CM address: the CMAC (see §2.5.2), the MAR (see §2.5.1), and the NPC (see §2.4.4). The CM ADDRESS MUX is controlled by the SCMA field (bits 13-12 of the microcode word).

The output of the CM ADDRESS MUX goes to the COMPARATOR, the ADDER, the SPECIAL CM ADDRESS DECODER, and the CM ERROR LATCH.

2.5.4. CMFL LATCH

The CMFL LATCH holds the CMFL (private CM Field Length). The CMFL LATCH is a 17-bit latch loaded from the ZBUS. The value on the ZBUS is usually supplied by an exchange package (see §1.7). The CMFL is loaded under the control of the BZD field (bits 41, 27, 26 of the microcode word).

2.5.5. CM ADDRESS COMPARATOR

The CM ADDRESS COMPARATOR compares the output of the CM ADDRESS MUX to the output from the CMFL LATCH. If the output from the MUX is larger than the output from the CMFL LATCH (i.e. if the address chosen exceeds the length of the program), an interrupt is generated.

The operation of the CM ADDRESS COMPARATOR is overridden by the OVERRIDE signal from the SPECIAL CM ADDRESS DECODE circuit. The OVERRIDE signal is active in monitor mode and if IOCM is being accessed.

2.5.6. CMRA LATCH

The CMRA LATCH holds the CMRA (private CM Reference Address) (see §2.5). The CMRA LATCH is a 17-bit latch loaded from the ZBUS. The value on the ZBUS is usually supplied by an exchange package (see 1.7). The CMRA LATCH is loaded under the control of the BZD field (bits 41, 27, 26 of the microcode word).

A 150 follows the CMRA; it is used to disable the CMRA whenever the OVERRIDE signal is active. The OVERRIDE signal is active in monitor mode and if IOCM is being accessed; it indicates that the CM ADDRESS (PCMA) is already an absolute address, and it should not be offset by the CMRA.

2.5.7. ADDER

The ADDER adds the CMRA to the CM ADDRESS (PCMA), making an absolute CM address (CMA) which is sent to the CM Board.

2.5.8. P registers for CM

The P registers contain absolute memory addresses; they are breakpoints for the Zephyr. The P registers are loaded under control of the BZD microcode field (bits 41, 27, 26 of the microcode word). Registers P1-P3 are for CM; these are kept separate from the P registers for BM, so that CM breakpoints may be checked simultaneously with BM breakpoints.

The most significant bit (bit 59) of registers P1-P3 is high (1) if the P register is to be active in monitor mode. Bit 58 is high if the P register is to be active in problem mode.

The least significant 18 bits in P1 are the CM read breakpoint address. If a Compass instruction reads this address, and if P1 is active, an interrupt will occur after the instruction completes. The least significant 18 bits of P2 are the CM write breakpoint address. The least significant 18 bits of P3 are the CM execution breakpoint address. If a CIW fetch occurs from this address, and if P3 is active, an interrupt will occur before any

further instructions are executed. The P registers contain absolute memory addresses, so the output of the ADDER (see §2.5.7) is checked against the P registers.

2.5.9. CM ADDRESS Hardware Details

Detailed B Board drawings, page 9

The CM ADDRESS COUNTER is loaded by the LDCMAC signal from the ZBUS destination decoder (page 1B). The count enable signals are controlled by the BM BLOCK TRANSFER CONTROL LOGIC (see §2.7.7).

Detailed B Board drawings, page 12

When a program accesses IOCM it always takes longer than an access to private CM. The time needed to access IOCM is longest if the CPU must wait for a PP microcomputer to finish using IOCM. When accessing IOCM the CLOCK CIRCUIT (see §2.2) is stopped until IOCM returns with the data.

Partially hidden CIW fetches are not effective when executing out of IOCM; IOCM is always requested and released during the execution of one microcode word, and it is never requested over a sequence of microcode words. When executing out of IOCM, the KNOPC signal on the CM ADDRESS MUX inhibits the selection of the NPC on all but the last microcode word which is part of the CIW fetch. The KNOPC signal will only allow the NPC if (1) the program is executing out of private CM, or (2) the program is executing out of IOCM and the CIW LATCH is being loaded. The CMFL INTERRUPT FLIP-FLOP is used to remember if the CMFL was exceeded anywhere in a sequence of microcode words. The interrupt occurs at the next NEXTOP JUMP (see §2.1.7(0)).

2.6. SPECIAL CM ADDRESS DECODE

CM addresses are 18 bits. If the most significant address bit is high (1), this indicates an address in IOCM or a special CM address. If the most significant bit is low (0), this indicates an address in private CM.

IOCM addresses are always absolute (i.e. no offsets are added in by the hardware to IOCM addresses). On the other hand, private CM addresses are absolute in monitor mode, but in problem mode they have the CMRA added in by the CM ADDRESS hardware (see §2.5.6).

The highest 16 addresses are special. These addresses have bits 17-04 all high (1); they are used to control special functions on the Zephyr (see §2.6.6).

2.6.1. MON FLIP-FLOP

A Zephyr CPU may run in monitor mode or in problem mode . In monitor mode, the Compass program has access to all of BM, to all of private CM, to all of IOCM, and to all special CM addresses used in communicating with the Peripheral Port microcomputers. It also is allowed to use certain privileged Compass instructions, it is immune from interrupts (see §2.1.8), and it may start programs with any exchange package (see §1.7, §9.9). In problem mode, the Compass program is more restricted in what it is allowed to do, and interrupts are allowed. Some floating point instructions are allowed only in problem mode.

In monitor mode all CM addresses are absolute addresses (the CMRA is disabled) and usually all BM addresses are absolute (the BMRA is set to 0 in the exchange package). In problem mode private CM addresses are offset by the CMRA and BM addresses are offset by the BMRA.

The Zephyr normally runs in problem mode. Monitor mode is only used at start-up and is used in parts of the operating system. The MON FLIP-FLOP indicates if the Zephyr is in monitor mode or in problem mode.

2.6.2. EXCHANGE JUMP FLIP-FLOP

The EXCHANGE JUMP FLIP-FLOP is used in the sequence of microcode words for the XJ instruction (see §9.9). It is used to temporarily give the same privileges as are given in monitor mode. This is sometimes called "temporary monitor mode."

2.6.3. HIO/LIO LATCH

The HIO/LIO LATCH holds the high IOCM address and the low IOCM address which specify the range of IOCM a user is allowed to use. It is loaded from the exchange package provided by the ZBUS (see §1.7). The output goes to the HIO/LIO COMPARATOR.

2.6.4. HIO/LIO COMPARATOR

The HIO/LIO COMPARATOR checks if the CM address is within the range of IOCM that the user is allowed to access. If bit 17 (the most significant bit) is high (1), this indicates that this is a request for IOCM. (The exception is when bits 17-04 are high (1), indicating that this is a special CM address (see §2.6.5)).

If the MON bit is high (1), then the Zephyr is in monitor mode and the address is immediately accepted as a valid address. If the MON bit is low (0), address bits 16-13 are compared to LIO and HIO. In order for the address to be accepted, address bits 16-13 must be greater than or equal to LIO and less than HIO. In addition, if this is a CIW fetch, the address is only accepted if the EXECIO bit is high (1) (see §1.7).

The HIO and LIO inputs are 4 bits each and come from the HIO/LIO LATCH.

The output of the circuit is the IOAREA signal which is high (1) when an IOCM address is accepted. The IOAREA signal is used to request IOCM and it is used to stop the clock circuit until IOCM is available. The IOAREA signal is also used to override the addition of the CMRA in the CM ADDRESS circuit and to override the function of the CMFL COMPARATOR. This is done since the CMRA and CMFL only apply to private CM accesses.

2.6.5. SPECIAL ADDRESS COMPARATOR

The SPECIAL ADDRESS COMPARATOR checks if all most significant 14 bits of the CM address are high (1). If all most significant 14 bits are high then this indicates that this is one of 16 special CM addresses. The output of this circuit does 2 things:

- (1) it enables the SPECIAL ADDRESS DECODER,
- (2) it disables the HIO/LIO COMPARATOR.

2.6.6. SPECIAL ADDRESS DECODER

The SPECIAL ADDRESS DECODER is used to decode the 16 special addresses. The SPECIAL ADDRESS DECODER allows various functions.

Writing to CM address 777776 starts the TIMER. The data is ignored. This is allowed only in monitor mode.

Writing to CM address 777775 loads the TIMER with the data written. This is allowed only in monitor mode (see §2.8).

Writing to CM address 777774 clears the TIMER interrupt. The data is ignored. This is allowed only in monitor mode (see §2.8).

Writing to CM address 777773 is used to set an interrupt for PP microcomputer A. The data is ignored. This is allowed in monitor mode and by the ABCD field (see §1.7).

Writing to CM address 777772 is used to clear an interrupt from PP microcomputer A. The data is ignored. This is used in the interrupt processing routine. This is allowed in monitor mode and by the ABCD field.

Writing to CM address 777771 is used to set the interrupt to PP microcomputer B. This is allowed in monitor mode and by the ABCD field.

Writing to CM address 777770 is used to clear the interrupt from PP microcomputer B. This is allowed in monitor mode and by the ABCD field.

Writing to CM address 777767 is used to set the interrupt to PP microcomputer C. This is allowed in monitor mode and by the ABCD field.

Writing to CM address 777766 is used to clear the interrupt from PP microcomputer C. This is allowed in monitor mode and by the ABCD field.

Writing to CM address 777765 is used to set the interrupt to PP microcomputer D. This is allowed in monitor mode and by the ABCD field.

Writing to CM address 777764 is used to clear the interrupt from PP microcomputer D. This is allowed in monitor mode and by the ABCD field.

Writing to CM address 777763 is used to initialize the COUNTER and to set memory error correction parameters. This is allowed only in monitor mode (see §5.5).

If any CM addresses between 777760 and 777777 is read, the data read is the value of the COUNTER (see §5.5). This is always allowed in monitor mode and problem mode.

Three special addresses (writing to 777762, 777761, and 777760) are not used yet.

None of the Compass Bulk Memory instructions function with special CM addresses.

2.6.7. ABCD FLIP-FLOPS

The ABCD FLIP-FLOPS hold the ABCD field from the exchange package. These control whether a program is allowed to set and clear interrupts for each of the 4 PP microcomputers.

The flip-flops are loaded from the ZBUS and are usually supplied from an exchange jump package (see §1.7).

2.6.8. SPECIAL CM ADDRESS DECODE Hardware Details

Detailed B Board drawings, page 1

H04 and H05 check for the 16 special CM addresses by comparing the most significant 14 CM address bits to high (1). The output (CMSPEC) enables the SPECIAL ADDRESS DECODERS (F04, F22) and disables the HIO/LIO COMPARATOR (L04, K17).

The HIO COMPARATOR (K17) compares HIO with the CM address (PCMA16-13). The most significant input into the comparator is the MON signal which overrides the function of the comparator in monitor mode. The second most significant input (KXIO) is used to inhibit execution into IOCM unless EXECIO is high (1).

The OR gates after F03 and F22 are used to restrict access to special CM addresses according to the ABCD field in the exchange package and according to the MON bit.

A17 is the exchange jump flip-flop. It implements temporary monitor mode by ORing its output with the output from the monitor flip-flop.

The IOREQ signal (IOCM request) does not go active until ADDRVAL goes low.

The OVERRIDE signal is used to inhibit the CMRA and CMFL. It goes active for valid IOCM addresses and for special CM addresses.

Flip-flops indicating interrupts going from the Zephyr CPU to the PP microcomputers are on the B Board (on chips F05 and H03). Flip-flops indicating interrupts going from the PP microcomputer to the Zephyr CPU are on the PP BUS INTERFACE Boards.

2.7. BULK MEMORY ADDRESS CIRCUIT

Each Compass program has some limited area of BM that it uses. The beginning of this area for a Compass program is specified by the BMRA. Whenever BM is accessed, the BMRA is added to the address. The BMFL specifies the length of the area which a program is allowed to access (see §1.7). Bulk Memory addresses are 27 bits; 26 of these bits are used by the BM Boards, and 1 is for future expansion.

The BM ADDRESS circuit adds in the BMRA (BM Reference Address) and checks that the address is within the BMFL (BM Field Length). It has a BMAR (BM Address Register and counter), a BMLR (BM Length Register and counter), an INTERLOCK FLIP-FLOP (for interlocked BM write operations), and BLOCK TRANSFER CONTROL LOGIC.

2.7.1. BMAR

The BMAR (BM Address Register) is a 24-bit counter. It holds all but the least significant 3 bits of the BM address. This is sometimes called the BM Record Address, where a record is 8 words of BM. The BMAR is loaded from the ZBUS under the control of the ENABLE field (bits 17-14 of the microcode word).

2.7.2. BMWP

The BMWP (BM Word Pointer) is a 3-bit counter which holds the least significant 3 bits of the BM address.

2.7.3. BMFL LATCH

The BMFL LATCH holds the BMFL (BM Field Length) (see §2.7). The BMFL LATCH is an 18-bit latch loaded from the ZBUS. The value on the ZBUS is usually supplied by an exchange package (see §1.7). The BMFL LATCH is loaded under the control of the BZD field (bits 41, 27, 26 of the microcode word).

2.7.4. BM ADDRESS COMPARATOR

The BM ADDRESS COMPARATOR compares the BMFL to the most significant 18 bits of the BMAR. If the output from the BMAR is larger than the BMFL (i.e. if the address chosen exceeds the length allowed for the program), an interrupt is generated. This is disabled by the OVERRIDE signal in monitor mode.

2.7.5. BMRA LATCH

The BMRA LATCH holds the BMRA (BM Reference Address) (see §2.7). The BMRA LATCH is an 18-bit latch loaded from the ZBUS. The value on the ZBUS is usually supplied by an exchange package (see §1.7). The BMRA LATCH is loaded under the control of the BZD field (bits 41, 27, 26 of the microcode word).

2.7.6. ADDER

The ADDER adds the BMRA to the most significant 18 bits of the 27-bit BM address, making an absolute BM address which is sent to the BMC Board (BM Controller Board). A 150 is used to generate differential signals for the BM Board.

2.7.7. BLOCK TRANSFER CONTROL LOGIC

BM is accessed using block transfers. Block transfers can be of variable length — as short as 1 word and as long as 100,000 words. Block transfers always occur between BM and CM. One example of a common block transfer occurs when a user starts a time slice and the user's information is copied from BM to CM (see §1.6).

The sequence of microcode words which implement block transfer instructions are highly optimized loops. The BLOCK TRANSFER CONTROL LOGIC is designed to provide signals needed by these loops.

BM is organized in groups of 8 words called BM records. During a block transfer, a special process must be performed by the microcode at each BM record boundary.

There is a 2-word wide path between BM and CM. Whenever possible the microcode transfers 2 words at a time; this is called a double-word transfer. The microcode may

transfer a double-word if: (1) the 2 words are in the same BM record, (2) the CM address is even, (3) it is not the end of the transfer, and (4) the transfer is not from BM to IOCM (this is the only one-word wide path between BM and CM).

2.7.8. BMLR

The BMLR (BM Length Register and counter) keeps track of the number of words left to transfer during a block transfer. The BMLR is loaded from the ZBUS under the control of the BZD field (bits 41, 27, 26 of the microcode word).

2.7.9. INTERLOCK FLIP-FLOP

The INTERLOCK FLIP-FLOP is used only with one-word writes to BM. It is a special mechanism used for synchronized communication between Zephyr CPUs. It is loaded at the same time that the BMLR is loaded.

A Compass program indicates an interlocked write by setting bit 29 of the BM address (BM addresses are normally 27 bits). During an interlocked write, a special sequence of microcode words is executed. The microcode first reads data stored at the BM address and checks if bit 59 (the most significant bit) is high (1). If the bit is high (1), no write is performed; if the bit is low (0), the write is performed as usual.

A normal (non-interlocked) write may be used to overwrite the BM location independent of the previous value of bit 59. Normal writes have BM address bit 29 low (0).

A programmer must use interlocked writes when requesting access to shared resources. By using interlocked writes, the programmer avoids simultaneous accessing of shared resources by several CPU's.

Procedure for interlocked writes:

To request access to a shared resource, a CPU performs an interlocked write to a special interlock address in BM. This special interlock address is set aside by the programmers to control access to the shared resource.

During the interlocked write, address bit 29 is high (1). The data in the interlocked write must have bit 59 high (1). If the interlocked write is successful, the data will be stored in BM, and execution will resume in the next CIW. If the write was not successful, the data will not be stored in BM, and execution will resume in the 3rd parcel of the CIW.

If the interlocked write was not successful, then some other CPU has already set bit 59 in the special interlock address. The CPU waits and tries again later. If the interlocked write was successful then no other CPU had set bit 59 in the special interlock address.

After a successful interlocked write, bit 59 is set, indicating that this CPU has exclusive access to the resource. When the CPU is finished, it performs a non-interlocked write to the special interlock address, clearing bit 59 to indicate that the resource is now available for other CPUs.

2.7.10. P registers for BM

The P registers contain absolute memory addresses; they are breakpoints for the Zephyr. The P registers are loaded under control of the BZD microcode field (bits 41, 27, 26 of the microcode word). Registers P4-P5 are for BM; these are kept separate from the P registers for CM, so that CM breakpoints may be checked simultaneously with BM breakpoints.

The most significant bit (bit 59) of registers P4-P5 is high (1) if the P register is to be active in monitor mode. Bit 58 is high if the P register is to be active in problem mode.

The least significant 27 bits of P4 are the BM read breakpoint address. If a BM read occurs at this address, and if P4 is active, an interrupt will occur. The least significant 27 bits of P5 are the BM write breakpoint address. If a BM write occurs at this address, and if P5 is active, an interrupt will occur. The P registers contain absolute memory addresses, so the output of the ADDER (see §2.7.6) is compared to the address in the P registers.

2.7.11. BULK MEMORY ADDRESS Hardware Details

Detailed B Board drawings, page 9

The BMAR and the BMWP are loaded with the LDBMAR signal.

The INCBMAR signal (from the ENABLE field decoder) enables incrementing the BMAR.

The BMWP is incremented independently from the BMAR. The BLOCK TRANSFER CONTROL LOGIC can simultaneously increment the BMWP, increment the CMAC, and decrement the BMLR. The DBCLK signal pulses once for single-word transfers and twice for double-word transfers; this way the counters are incremented or decremented twice for double-word transfers. The microcode uses the ENABLE field and the FFPU field to select which counters get incremented or decremented (see §9.12).

The double pulse in DBCLK is generated using delayed versions of KECLK. One wire loop controls the pulse width while a longer wire loop controls the pulse separation.

The DOUB signal indicates if a double-word transfer is possible (see §2.7.7).

The DOUBLE signal is a latched version of DOUB, and stays stable while the BMWP and CMAC are being incremented.

The FASTRD signal is used by the BM read sequence of microcode words. FASTRD indicates if the next record to be read from BM can be read by the optimized read sequence of microcode words. If it cannot be read by the optimized sequence of microcode words, a slower sequence of microcode words is used (see §9.12).

FASTRD goes low only if: (1) a full record is to be transferred, (2) this is the beginning of the record, (3) IOCM is not the destination of the transfer, (4) the CM address is even, (5) the BM address is even, (6) the BM is ready, and (7) there have been no errors. For optimal block transfer speeds, Compass programmers should arrange block transfers so they begin at even CM and even BM addresses.

Since the BLOCK TRANSFER CONTROL LOGIC automatically decides when to do single-word transfers and when to do double-word transfers, the microcode cannot predict how many transfers a record will require. The BMXFR signal is used by the microcode to determine if this is the last transfer in the BM record.

2.8. TIMER

The TIMER is used to generate an interrupt after a given amount of time. For example, at the beginning of a time slice the TUTOR executor (see §1.5.2, §1.6) may set up the TIMER to interrupt when the time slice is over.

The TIMER is loaded by writing a 12-bit value to a special CM address (see §2.6.6). This value is in units of 1/32,000 of a second. The TIMER is started by writing to another special CM address. The TIMER should not be loaded with 0.

2.8.1. TIMER Hardware Details

Detailed B Board drawings, page 3

Four flip-flops are used with the TIMER: (1) an INTERRUPT FLIP-FLOP, (2) a RUN FLIP-FLOP to indicate that the TIMER is running, (3) a TIMER LOAD FLIP-FLOP used to pulse the counter clock pins when the TIMER is being loaded, (4) a TIMER CLOCK SYNCHRONIZING FLIP-FLOP to synchronize the 32 kHz clock with KECLK.

2.9. ZBUS MUX

The ZBUS MUX is used to multiplex data about the Zephyr status on to the ZBUS. This is used for XJ instructions (see §9.9), RJ instructions (see §9.8), memory errors (see §9.10.1), and debugging microcode. The ZBUS MUX is controlled by the BZBSA

microcode field (bits 31-28 of the microcode word). The top left of the block diagram shows the bit fields available through the ZBUS MUX.

2.9.1. RJ

The RJ word is used with return jump Compass instructions. It contains the 0400 Compass instruction in the most significant 12 bits and the NPC value in the next 18 bits (see §9.8 for details on the RJ instruction).

2.9.2. XJ, CMRAFL, BMRAFL

The XJ, CMRAFL, BMRAFL words are used to generate an exchange package for the currently executing Compass program (see §1.7 for details on exchange packages, and §9.9 for XJ instruction details).

2.9.3. MMAR

The MMAR word has the value of the MAR LATCH in the least significant 18 bits (see §2.5.1).

2.9.4. ERROR ADDR/K, BMAR/SYN

The ERROR ADDR/K, BMAR/SYN words are used to report single-bit or double-bit memory error information as well as to read the K field of the current instruction, and the BMAR and the BMWP. The CM ERROR PCMAF field contains the value of the PCMAF when a memory error occurred. The RELERR field is high (1) if the error occurred at an address where the CMRA was added in.

The CMERR SYN field indicates the syndrome bits at the time of the memory error (see §5.9).

The error information is read from the CM ERROR LATCH. The CM ERROR LATCH stores information for the first error which occurred since it was cleared (see §5.10).

2.9.5. BMLR/CMAC

The BMLR/CMAC word reads the value of the BMLR and CMAC (see §2.7.8 and §2.5.2).

2.9.6. PC

The PC word has the value of the PC in the least significant 18 bits.

2.9.7. BREAKPOINT (P registers)

The BREAKPOINT word gives the value of a P register. P registers contain breakpoints for the Zephyr; see §9.23 for the function of each P register. The k field of the Compass instruction selects which P register is to be read onto the ZBUS.

2.9.8. ZBUS MUX Hardware Details

Detailed B Board drawings, page 10

The least significant 36 bits of the ZBUS MUX are implemented with type 163 multiplexer chips. The exchange jump and return jump require values set in the most significant 24 bits. These are enabled under the ZBUS using separate 150s and 155s.

Detailed B Board drawings, page 13

The P registers are read through the ZBUS MUX. 155 multiplexers are used to choose between the CM and the BM P registers. A 150 is used to enable P0; P0 provides information about which P register caused an interrupt.

3 . A BOARD

3.1. Registers

The Zephyr has 5 sets of general purpose registers which are used by Compass instructions. There are 8 registers in each set.

The Compass instruction number determines which sets of registers the Compass instruction will use. The i, j, and k fields of the Compass instruction specify the register numbers within each set (see §2.3).

Some of the register sets are 18 bits and some are 60. The 18-bit registers are primarily used to hold memory addresses and loop counters, while the 60-bit registers are primarily used to hold data.

The input to the registers is from the BZB BUS (Buffered ZBUS). The BZB BUS may get its value from the output of the ALU, the BARREL SHIFTER, from the D Board, from the B Board, or from CM (Central Memory) via the B Board. The output from the registers is to the QBUS and PBUS. These busses go to the ALU, the BARREL SHIFTER, the D Board, and to CM via the ALU and the B Board.

The next sections describe the X, T, A, B, and C register sets. P registers are for breakpoints, not for general purpose registers. Their function is described in §9.23.

3.1.1. X Registers

The X registers are 60-bit general purpose registers. X registers may be used to hold floating point values and large integer values.

In hardware there are two identical copies of the X-register set. When a Compass instruction reads an X register, the microcode may use either copy of the X register set. Both sets of X registers are used for Compass instructions which read 2 different X registers simultaneously. Whenever a write to an X register occurs, both copies of the X register are written.

In addition to the usual X register uses, register X0 is used in memory transfers to BM (Bulk Memory) in order to specify the BM start address (see §9.12).

3.1.2. T Registers

The T registers are an alternate set of X registers. They are 60-bit general purpose registers.

There are two identical copies in hardware of the T register set.

When a Compass instruction specifies an X register, it may actually access a T register. This depends on the state of the X/T FLIP-FLOP (see §3.2.3).

3.1.3. A Registers

The A registers are 18-bit registers which hold CM addresses. The hardware has only one copy of the A registers; two copies are not required because Compass instructions never read two A registers at a time.

Compass instructions use the A registers to read and write CM (Central Memory). When a Compass instruction writes an address to registers A1-A5, the corresponding X register receives the data at that address. For example: if the instruction writes 057243 to register A2, then the data at memory address CMRA plus 057243 is read into register X2 (see §2.5).

When a Compass instruction writes an address to registers A6 or A7, the data in X6 or X7 is written to CM at that address. To write a value to CM, a Compass program must first put the value in X6 or X7, and then put the address into A6 or A7.

When a Compass instruction writes A0, this does not cause a memory access. A0 is used in memory transfers to and from Bulk Memory; A0 specifies the starting address in CM (see §9.12).

When Compass instructions read A registers, this does not cause any memory accesses. When A registers are read into X registers, the values are sign extended.

3.1.4. B Registers

The B registers are 18-bit general purpose registers which hold signed values. Register B0 is special; it always has the value 0 when it is read. Writing to register B0 has no effect.

In Compass, B registers are often used for address computation, for loop counters, and to hold small integers.

There are two identical copies in hardware of the B registers. This is for Compass instructions which read two different B registers.

When B registers are read into X registers, the values are sign extended.

3.1.5. C Registers

The C registers are 18-bit general purpose registers which hold signed values. These registers are generally not used by Compass. They are used in interrupt processing and to hold information useful for debugging hardware (see §9.24).

3.1.6. Registers Hardware Details

Detailed A Board drawings, pages 5 and 6

The two copies of the X and T register sets are kept in two sets of 145s. One set of 145s outputs to the QBUS (one input to the ALU), and the other set outputs to the PBUS (the other input to the ALU). B registers have a similar configuration.

Since no A registers are required on the PBUS, that space is used to hold the C registers.

The X and T registers are kept together. The 145 chips have 4 address bits. One address bit selects between X and T registers; the other 3 select the register number. Similarly, the A and B registers are kept together on the QBUS, and the C and B registers are kept together on the PBUS.

3.2. REGISTER ADDRESS MUX

The registers output to the QBUS and PBUS. The REGISTER ADDRESS MUX selects which register to read onto the PBUS, which register to read onto the QBUS, and which register to write onto the the BZB BUS.

3.2.1. Read Register Controls

The QT field (bits 59-57 of the microcode word) and PT field (bits 54-52 of the microcode word) select which register sets are to be read onto the QBUS and PBUS. If the QT field or the PT field specify X registers, then the X/T FLIP-FLOP determines if X registers or T registers are read (see §3.2.3).

When reading registers, the PRS field (bits 51-50 of the microcode word) and the QRS field (bits 56-55 of the microcode word) both control how registers are chosen from within a register set. The PRS and QRS fields may specify that the register number come from the i field of the Compass instruction (see §2.3), the j field, the k field, or the RMFLD field (bits 23-21 of the microcode word).

The QREG0 signal causes register 0 to be selected on the QBUS, overriding the QRS field. This allows the microcode to perform an operation using register number 0 on the QBUS and another register specified by the RMFLD field on the PBUS. This is only used by the

sequences of microcode words which implement floating point instructions. The QREG0 signal is controlled by the ENABLE field (bits 17-14 of the microcode word).

3.2.2. Write Register Controls

The AZD field (bits 25-24 of the microcode word) selects which register set is to be written from the BZB BUS. If the AZD field selects the X register set as the destination, then the X/T FLIP-FLOP determines if X registers or T registers are written.

The WRS field (bits 36-35 of the microcode word) is used to select a register from the destination register set. The WRS field may specify that the register number comes from the i field of the Compass instruction (see §2.3), the j field, the k field, or the WMFLD field (bits 20-18 of the microcode word).

3.2.3. X/T FLIP-FLOP

The X/T FLIP-FLOP determines if X registers or the alternate T registers are to be used when X registers are specified.

The microcode will automatically set the X/T FLIP-FLOP to use T registers when switching to monitor mode. This occurs at start-up and during interrupt processing and traps (see §9.9-§9.11).

The microcode will automatically set the X/T FLIP-FLOP to use X registers when switching to problem mode.

The microcode will also set the X/T FLIP-FLOP for T registers during some floating point instructions. This is done because the microcode needs "scratch" registers for intermediate results. These floating point instructions are not available in monitor mode, since T registers are already being used in monitor mode.

The X/T FLIP-FLOP is controlled by the SELECT field (bits 63-60 of the microcode word), and the ENXT signal derived from the ENABLE field (bits 17-14 of the microcode word).

3.2.4. REGISTER ADDRESS MUX Hardware Details

Detailed A Board drawings, page 1

The PTS and QTS fields may select X registers in two ways: all 60 bits of the X register, or only the least significant 18 bits of the X register. The latter is used for Compass instructions which perform 18-bit math (numbers 50-77 in table 9.1). These 18-bit values will be sign extended before entering the ALU (see §3.3.3).

When copying X registers to T registers, the SELECT field is equal to 2. This corresponds to the Q OR P operation of the ALU (see §3.4.1). When copying T registers to X registers, the SELECT field is equal to 1. This corresponds to the Q AND P operation of the ALU. When copying between X and T registers, the microcode specifies the source register on both the QBUS and PBUS (see §9.22 and block diagram).

3.3. ALU and BARREL SHIFTER Inputs

The inputs to the ALU and the BARREL SHIFTER are the registers (see §3.1), the D Board, the B Board, and the constants 0 and 1.

All the inputs pass through the COMPLEMENT AND SIGN EXTEND circuit. One input is called the QBUS *before* the COMPLEMENT AND SIGN EXTEND, and is called the ABUS *after* the COMPLEMENT AND SIGN EXTEND. The other input is called the PBUS *before* the COMPLEMENT AND SIGN EXTEND, and is called the BBUS *after* the COMPLEMENT AND SIGN EXTEND.

The inputs to the QBUS are selected by the QT field (bits 59-57 of the microcode word), and the inputs to the PBUS are selected by the PT field (bits 54-52 of the microcode word).

3.3.1. D Board Inputs

The D Board supplies QBUS and PBUS values which are required for the sequences of microcode words used in floating point add and normalize instructions (see §4.1.1). The D Board also supplies ZBUS values for other operations; these do not go through the ALU.

3.3.2. B Board Inputs

The PBUS may receive the K field from the B Board's PARCEL REGISTER. This is the 18-bit K field from the Compass instruction being executed. This is only valid if a 30-bit Compass instruction is being executed (see §2.3).

The QBUS may receive the j and k fields as a 6-bit number from the B Board. The j and k fields combined are called the jk field. Few Compass instructions use the jk field. In these Compass instructions, the jk field is used to specify a shift count or mask count for the BARREL SHIFTER (see §3.4).

3.3.3. P and Q BUS COMPLEMENT AND SIGN EXTEND

The COMPLEMENT AND SIGN EXTEND circuits are under the control of the SELECT field (bits 63-60 of the microcode word) (see §3.4.1).

Sign extension is used when the source is an 18-bit register. This converts the value from 18 bits to 60 bits, so that it can be used in the 60-bit ALU.

In Compass instructions which perform 18-bit math and have an X register as a source, only the least significant 18 bits of the 60-bit X register are read. These 18 bits are sign extended to 60 bits by the COMPLEMENT AND SIGN EXTEND, using bit 17 as the sign bit.

3.3.4. ALU and BARREL SHIFTER Inputs Hardware Details

Detailed A Board drawings, page 7

The inputs to the COMPLEMENT AND SIGN EXTEND circuit are the 60-bit P and Q BUSSES. When an 18-bit value is on the P or Q BUS, the most significant 42 bits are all 0. Sign extension of the 18-bit values is accomplished by inverting the complement controls for the most significant 42 bits. For more details on the COMPLEMENT AND SIGN EXTEND, see §3.5.1.

3.4. ALU and BARREL SHIFTER

The ALU (Arithmetic Logic Unit) performs 60-bit integer add, subtract, and logical operations. The BARREL SHIFTER performs 60-bit shifting.

3.4.1. Functions

ALU and BARREL SHIFTER functions are controlled by the SELECT field (bits 63-60 of the microcode word). The SELECT field controls the COMPLEMENT AND SIGN EXTEND circuits as well. The SELECT field allows the following functions listed by SELECT field value (SELECT values 0-10 are implemented by the ALU and values 11-15 are implemented by the BARREL SHIFTER):

- | | | |
|-----|---------------------|---|
| (0) | Transfer Q | The QBUS passes through unchanged. |
| (1) | Q AND P | Bit-wise AND of QBUS and PBUS. |
| (2) | Q OR P | Bit-wise OR of QBUS and PBUS. |
| (3) | QEXCLUSIVEOR P | Bit-wise EXCLUSIVE OR of QBUS and PBUS. |
| (4) | Transfer P inverted | The PBUS is inverted. |
| (5) | Q AND P inverted | Bit-wise AND of QBUS and PBUS inverted. |
| (6) | Q OR P inverted | Bit-wise OR of QBUS and PBUS inverted. |

- (7) Q EXCLUSIVE OR P inverted
Bit-wise EXCLUSIVE OR of QBUS and PBUS inverted.
- (8) Q plus P plus end-around-carry
The QBUS and PBUS values are added with end-around-carry (see §1.9). The values added are inverted operands, resulting in $0 + 0 = 0$, $-0 + 0 = 0$, $0 + -0 = 0$, and $-0 + -0 = -0$. The QBUS, PBUS, and ZBUS are inverted by the COMPLEMENT AND SIGN EXTEND circuits.
- (9) Q minus P plus end-around-carry
The PBUS is subtracted from the QBUS, and that value is added with end-around-carry. The subtract is done with inverted operands, so that negative zero is generated only in the case of $-0 - 0 = -0$.
- (10) Q plus P plus saved end-around-carry
The QBUS and PBUS values are added with the saved end-around-carry. This is used in the floating add sequences of microcode words (see §9.18). The add is done with inverted operands, so that negative zero is generated at the proper times.
- (11) MASK
The least significant 6 bits of the QBUS specify the mask count (how many of the most significant bits in the result are to be set high (1)). For example, if the PBUS has the value 31, then the output will have the 31 most significant bits set high (1), and the 29 least significant bits low (0).
- (12) Left shift jk
The value on the PBUS is shifted left by the number of bits specified on the QBUS. This is a circular shift. Only the least significant 6 bits of the QBUS are used.
- (13) Right shift jk
The value on the PBUS is shifted right by the number of bits specified on the QBUS. This is an arithmetic shift; the sign-bit of the PBUS will appear in the most significant bits of the output. Only the least significant 6 bits of the QBUS are used.
- (14) Left shift by Q
The value on the PBUS is circular left shifted by the number of bits specified on the QBUS. This is the same as (12) above, except that if the QBUS specifies a negative number (QBUS bit 17 high), then an arithmetic right shift is performed.

- (15) Right shift by Q The value on the PBUS is arithmetic right shifted by the number of bits specified on the QBUS. This is the same as (13) above, except that if the QBUS specifies a negative number (QBUS bit 17 high), then a circular left shift is performed.

The preceding operations only occur when the ZBUS source specified by AZBSA field (bits 34-32 of the microcode word) is either the BARREL SHIFTER or the ALU. The SELECT field is also used for unrelated operations on other boards (see §1.8.5).

3.4.2. ALU and BARREL SHIFTER Hardware Details

Detailed A Board drawings, page 2

B02, A02, B05 and A05 are multiplexers used to derive the ALU control signals from the SELECT field, as shown in the ALU Function Table on the bottom left of the drawing.

K02, K03 and K04 are multiplexers used to derive the COMPLEMENT AND SIGN EXTEND control signals from the SELECT field, as shown in the ALU Function Table on the bottom left of the drawing.

H01 is used to derive the KRIGHT signal from the SELECT field. This signal is used to select right shifts for the BARREL SHIFTER. It is also used in de-selecting the BARREL SHIFTER, activating the BARREL SHIFTER's Byte Mask.

The ZPL signal (bottom right of drawing) is used to avoid -0 results when right shifting more than 63 places. This is a feature of Compass right shifting instructions. This feature is overridden when performing pre-shifts for floating add instructions by the FADDFHS signal.

The QBUS COMPLEMENT inverts the QBUS in order to negate right shift counts.

S02 and P02 are used to add 60 to the shift count when right shifts are performed. This is because right shifts are indicated by negative numbers, and the BARREL SHIFTER always expects a circular left shift counts.

If the PBUS value is negative, then the PBUS input and output of the BARREL SHIFTER will be complemented. This allows the BARREL SHIFTER to shift only positive numbers.

The shift mask generator computes the controls for both the Bit Mask and the Byte Mask of the BARREL SHIFTER. It uses the least significant 3 bits of the shift count to compute the values for the Bit Mask, and the most significant 3 bits of the shift count to compute the values for the Byte Mask. Both masks are inhibited for left shifts.

ALU functions 1 and 2 are used when transferring between X and T registers (see §3.2.4).

ALU function 9 (Q minus P plus end-around-carry) has a special use in the sequence of microcode words for floating point divide. When the DIVFIX signal is low (0) and if the QBUS value is positive, then the ALU will perform an add in place of a subtract. This is used to add or subtract from a floating point value (see §9.20.2, §3.5.1).

When neither the ALU nor the BARREL SHIFTER are selected, then B03 forces the BARREL SHIFTER's and ALU's copies of the SELECT field value to 0. In addition, the ZSALU signal is used to disable the ALU (B05).

When neither the ALU nor the BARREL SHIFTER are selected, then the QBUS COMPLEMENT AND SIGN EXTEND takes the absolute value of the QBUS. This is used for X register tests (see §3.6). This is the purpose of the BSALU AND QBUS59 notation in the first line of the ALU FUNCTION table. This is implemented by the KXTES signal connected to K03.

Detailed A Board drawings, page 8

P08 and L08 are carry-look-aheads used to compute the end-around-carry. P10, L10, P09 and L09 are used to compute internal carries for the adder. In ALU function 10 (Q plus P plus saved end-around-carry), P08 and L08 are not used. Instead, the SAVED CARRY FLIP-FLOP provides end-around-carry. The SAVED CARRY FLIP-FLOP also saves the carry-out from P10 after the add is complete.

Detailed A Board drawings, page 9

The BARREL SHIFTER has 4 stages: the Bit Mask, the Bit Shift, the Byte Shift and the Byte Mask. The Bit Shift performs shifts of 0-7 places; these are specified by the least significant 3 bits of the shift count. The Byte Shift performs shifts of 8, 16, 24, 32, ... places (shifts in multiples of 8). Similarly, the Bit Mask operates on 0-7 bits and the Byte Mask operates on multiples of 8 bits at a time. The Bit Mask and Byte Mask are used both for generating masks, and for sign extension in arithmetic right shifts.

These are the stages in generating a mask (see MASK function in §3.4.1):

- (1) The value on the PBUS is 0. The value on the QBUS is the mask count.
- (2) The PBUS COMPLEMENT AND SIGN EXTEND complements all the bits on the PBUS so that they are all high (1).
- (3) The Bit Mask zeroes the least significant N bits from step (2), where N is the least significant 3 bits of the QBUS.

- (4) The Bit and Byte Shifters left shifts the result from (3) by 60 minus the value on the QBUS. After this step, the N consecutive zeroes have a multiple of 8 bits of 1's to the left.
- (5) The Byte Mask zeroes out all the bits of 1's to the left of the N bits of 0's; now there are as many most significant bits set to 0 as specified on the QBUS.
- (6) The ZBUS COMPLEMENT AND SIGN EXTEND complements the result. Now there are as many most significant bits set to 1 as specified on the QBUS.

This is an example of values at each stage when generating a mask with a 21 (octal 25) on the QBUS: (these are 60-bit octal values)

- | | | |
|-----|--------------------------|---------------------------------------|
| (1) | 0000 0000 0000 0000 0000 | Value on PBUS |
| (2) | 7777 7777 7777 7777 7777 | After PBUS COMPLEMENT AND SIGN EXTEND |
| (3) | 7777 7777 7777 7777 7740 | After Bit Mask |
| (4) | 7777 7407 7777 7777 7777 | After Bit and Byte Shifters |
| (5) | 0000 0007 7777 7777 7777 | After Byte Mask |
| (6) | 7777 7770 0000 0000 0000 | After ZBUS COMPLEMENT AND SIGN EXTEND |

The procedure for sign extension during arithmetic right shifts is similar to the procedure above except that the PBUS starts with a non-zero value. The bits that are 1's in steps 3-5 above are the data bits for right shifts. Negative values on the PBUS are complemented before entering the BARREL SHIFTER, and complemented again after the BARREL SHIFTER.

3.5. ALU and BARREL SHIFTER Output

The input to the ZBUS COMPLEMENT AND SIGN EXTEND is the output of the ALU and BARREL SHIFTER (called the FBUS). The output of the ZBUS COMPLEMENT AND SIGN EXTEND is the ZBUS. The COMPLEMENT AND SIGN EXTEND is controlled by the SELECT field (bits 63-60 of the microcode word) (see §3.4.1).

When values are read from 18-bit registers and written to 60-bit registers, the values must be sign extended. Since 18-bit values are sign extended before entering the ALU, normally sign extension is not needed after the ALU. Sign extension is only needed if the ALU underflowed or overflowed by 18 bits.

3.5.1. ALU and BARREL SHIFTER Output Hardware Details

Detailed A Board drawings, page 11

The DIVFIX signal is used in the sequence of microcode words for divide (see §9.20.2, §3.4.2). Bit 47 of the Z BUS COMPLEMENT AND SIGN EXTEND is inverted if DIVFIX is active, and if the value of bit 48 changed in the ALU. This is used to correct the value of a positive floating point number if, after adding 1 to the mantissa, the mantissa overflows. For example:

	1720 7777 7777 7777 7777	(initial floating value)
	$\begin{array}{r} + \\ \hline 1 \end{array}$	
	1721 0000 0000 0000 0000	(after ALU)
then	1721 4000 0000 0000 0000	(after bit 47 is inverted)

This is also used to correct the value of a negative floating point number if, after subtracting 1 from the mantissa, the mantissa underflows. For example:

	6057 0000 0000 0000 0000	(initial floating value)
	$\begin{array}{r} - \\ \hline 1 \end{array}$	
	6056 7777 7777 7777 7777	(after ALU)
then	6056 3777 7777 7777 7777	(after bit 47 is inverted)

A special situation arises involving 2^{47} and indefinite exponents (0/0). This is not handled automatically by the hardware; the sequence of microcode words for divide detects this case and traps to a software simulation (see §9.20.2):

	1776 7777 7777 7777 7777	(initial floating value, just under 2^{47})
	$\begin{array}{r} + \\ \hline 1 \end{array}$	
	1777 0000 0000 0000 0000	(after ALU)
then	1777 4000 0000 0000 0000	(after bit 47 is inverted -- indefinite value!)
correct value	2000 4000 0000 0000 0000	(2^{47})

3.6. X Register Test

The X Register Test circuit generates the CLDPC0 signal which is used in the B Board PC COMPUTATION circuit (see §2.4.6). The *i* field of the Compass instruction (see §2.3) is used to select CLDPC0 from one of 8 different conditions. This is used by the conditional jumps dependent on the X registers in Compass (Compass instruction number 03, see table 9.1).

The X Register Test circuit tests the value on the QBUS. The conditions detected by the X Register Test circuit are: integer zero, integer non-zero, negative, positive, floating point

indefinite (see §1.10.2), floating point definite, floating point overflow, and floating point in range.

X register tests are performed on absolute values when the ALU is not being used (see §3.4.1); if the ALU is in use, then the X register tests may only detect positive or negative values, depending on the controls to the QBUS COMPLEMENT AND SIGN EXTEND (see §3.3.3).

The X Register Test circuitry also computes values for the floating point unit. This includes detection of zero in all the exponent bits (floating point underflow), zero in all mantissa bits, floating point indefinite, and floating point overflow. In addition, the Normalize Shift Count is computed; this is the amount an un-normalized mantissa must be shifted for normalization (see §1.10.1).

3.7. B Register Test

The B Register Test circuit generates the CLDPC1 signal which is used in the B Board PC COMPUTATION circuit (see §2.4.6). The least significant 2 bits of the Compass instruction number are used to select CLDPC1 from one of 4 different conditions. This is used by the conditional jumps dependent on the B registers in Compass (Compass instruction numbers 04-07, see table 9.1).

The B Register Test circuit tests the value on the ZBUS. The conditions detected by the B Register Test circuit are: the ALU result was integer zero, the ALU result was integer non-zero, the ALU result was negative, and the ALU result was positive.

4. D and E BOARDS

4.1. D and E BOARD CONTROL

The D and E Boards perform floating point functions. Most floating point instructions are implemented with long sequences of microcode words. It will be helpful to study the sequences of microcode words which implement the floating point Compass instructions when learning about the hardware (see §9.14 - §9.20).

The D and E Boards perform integer multiply, floating multiply, floating divide, floating add, floating subtract, normalize, population count, pack and unpack.

When two floating point numbers are multiplied together, the two exponents are added and the two 48-bit mantissas are multiplied together to give a 96-bit result. The D Board can return a floating point number which contains the most significant 48 bits of the result with the exponent, or the least significant 48 bits of the result with the exponent decreased by 48. If the inputs to the multiplier were normalized floating point numbers, then the result will be a normalized floating point number. This may require a post-normalize: a one-bit left shift and a corresponding decrement of the exponent.

Integer multiplies are restricted to 48-bit integers. Two 48-bit numbers are multiplied together to give a 96-bit result. The D Board can provide either the most significant or least significant 48 bits of the result.

Integer multiplies are performed with the same Compass instructions as the floating point multiply. The D Board decides whether the multiply will be integer or floating point. If both inputs have all exponent bits equal to the sign bit (usually interpreted as an underflow exponent) and if at least one input is not normalized, then an integer multiply is performed (the result will have all exponent bits equal to the sign bit).

When two numbers are divided, the exponents are subtracted and the two 48-bit mantissas are divided to give a 48-bit result. No integer divide capability is provided; integers must be converted to floating point format, divided, and then converted back to integer form.

Divides are implemented using an iterative algorithm. This algorithm uses several multiplies to converge on the proper divide result. A few special bells and whistles are added on the A Board (see DIVFIX signal: §3.4.2, §4.5.1) and on the D Board to help efficiently implement the iterative algorithm in microcode.

In floating add and subtract, the two source registers from the Compass instructions are read on the A Board and sent to the D Board. The D Board computes the exponent value of the result, and provides the A Board with sign extended mantissas so that the A Board may compute the result's mantissa value. The A Board uses the Q plus P plus saved end-

around-carry ALU function (see §3.4.1), and the A Board performs shifts and masks, with shift counts and mask count provided by the D Board. At the end of the floating add and subtract, the mantissa from the A Board and the exponent from the D Board are packed together on the D Board and then stored in the A Board destination registers.

4.1.1. FPU CONTROL DECODER

The floating point function is selected by the FFPU field (bits 6-2 of the microcode word). The FPU CONTROL DECODER decodes the FFPU field into the following signals:

LDMUL2	This loads the second multiply operand's exponent into the EXPONENT LATCH B, and the mantissa into the Y input of the MULTIPLY ARRAY. It also initializes the MULTIPLIER ARRAY (see §4.8.7) and the DISBLNK FLIP-FLOP (see DISBLNK below). The SELECT field (bits 63-60 of the microcode word) indicates a rounding or non-rounding multiply. This always precedes LDMUL1.
LDMUL1	This loads the first multiply operand's exponent into the EXPONENT LATCH A, and the mantissa into the X input of the MULTIPLY ARRAY. The SPECIAL signal goes active if either multiply operand has overflow or indefinite exponents (see §1.10.2). The SPECIAL signal goes to the CONDITION MUX on the B Board (see §2.1.7), and the microcode traps to a Compass multiply simulation if SPECIAL is active. LDMUL2 and LDMUL1 are used for both integer and floating multiplies.
MPYGO	This starts the MULTIPLY SEQUENCER (see §4.1.2). The SELECT field controls doubling of the the multiply result.
OMULSP	This outputs the most significant 48 bits of the result mantissa of a multiply packed with the exponent. This is normally used for floating point multiplies, but it may also be used to get the most significant 48 bits resulting from an integer multiply. The MULOVL signal indicates if there was an overflow or underflow in the exponent computations. If MULOVL goes active, the microcode traps to a Compass routine which stores underflow ($\pm 1/\infty$) or overflow ($\pm \infty$) into the result register (see §1.10.2). A one-bit post-normalize is performed if both inputs were normalized and the output is not normalized; the SELECT field may suppress this one-bit post-normalize (see §4.1).

OMULDP	This outputs the least significant 48 bits of the result mantissa for a multiply packed with the exponent decremented by 48. This is normally used for integer multiplies, but it may also be used to get the low significance result of a floating point multiply. Both OMULSP and OMULDP return all exponent bits equal to the sign bit if the inputs to the multiply were integers (see §4.1). A one-bit post-normalize is performed if both inputs were normalized and the most significant 48 bits of the output is not normalized; the SELECT field may suppress this one-bit post-normalize.
LDDIV2	This loads the second divide operand's exponent into the EXPONENT LATCH B, and the mantissa into the Y input of the MULTIPLY ARRAY. This precedes LDDIV1.
LDDIV1	This loads the first divide operand's exponent into the EXPONENT LATCH A, and the mantissa into the Y input of the MULTIPLY ARRAY. The SPECIAL signal indicates if this divide must be simulated in Compass (see §4.1.3 for details).
DIVLOOK	This performs the DIVIDE ROM look up and loads the result into the X input of the MULTIPLY ARRAY. This is used with the second operand (denominator) of the divide.
LDITER1	This loads the complement of the value (2^{48} minus the value) into the X input of the MULTIPLY ARRAY. This is used in each iteration of the divide algorithm.
LDDIV2	This loads the value into the Y input of the MULTIPLY ARRAY. This is used during each iteration of the divide algorithm.
OITER	This outputs the divide iteration result. The SELECT field may be used to suppress one-bit post-normalize.
ODIV	This outputs the packed divide result. The MULOVL signal goes active if the divide exponent overflowed or underflowed. If MULOVL goes active, the microcode traps to a Compass routine which stores underflow ($\pm 1/\infty$) or overflow ($\pm \infty$) into the result register (see §1.10.2).

DISBLNK	This sets the DISBLNK FLIP-FLOP; the DISBLNK FLIP-FLOP is cleared by the LDMUL2 signal (see above). The NOBLANK signal from the DISBLNK FLIP-FLOP disables the BLANKME and BLANKMM signals during the iterations of the divide algorithm (see §4.1.3). This is necessary to permit intermediate divide results with an underflow exponent.
LDADD2	This loads the second floating add operand's exponent into the EXPONENT LATCH B, and the mantissa into the ADD/SUBTRACT/NORMALIZE MANTISSA LATCH B. The SELECT field indicates a rounding or non-rounding add. LDADD2 always precedes LDADD1.
LDSUB2	This loads the second floating subtract operand's exponent into the EXPONENT LATCH B, and the mantissa into the ADD/SUBTRACT/NORMALIZE MANTISSA LATCH B. This operation is the same as LDADD2 except that the value is complemented. The SELECT field indicates a rounding or non-rounding subtract. LDSUB2 always precedes LDADD1.
LDADD1	This loads the first add or subtract operand's exponent into the EXPONENT LATCH A, and the mantissa into the ADD/SUBTRACT/NORMALIZE MANTISSA LATCH A. The SPECIAL signal indicates whether this add must be simulated in Compass (see §4.1.3 for details).
ADDSHF	This provides the A Board with the sign extended mantissa of the smaller floating add or subtract operand. The sign extended mantissa is shifted left one and includes a round bit inserted in the least significant bit position (see §4.10). ADDSHF also provides a right shift count which is required to make the significance of the smaller mantissa equal to the significance of the larger mantissa. This is used together with ADDMAN to perform the high significance add in the add algorithm (see §9.18).

ADDMAN	This outputs the sign extended mantissa of the larger floating add or subtract operand to the A Board. The sign extended mantissa is shifted left one and includes a round bit inserted in the least significant bit position (see §4.10). ADDMAN is also used to latch the POST-NORMALIZE FLIP-FLOP in the final add of the add algorithm (see §4.9.2).
ADDSHFL	This provides the A Board with the sign extended mantissa of the smaller floating add or subtract operand. The sign extended mantissa is shifted left 1 and includes a round bit inserted in the least significant bit position (see §4.10). ADDSHFL also provides a left shift count, equal to 60 minus the shift count used in ADDSHF. ADDSHFL is used together with ADDMASK to perform the low significance add in the add algorithm.
ADDMASK	This outputs the mask count for the mantissa shifted with ADDSHFL. ADDSHFL and ADDMASK are used for the low significance add in the add algorithm.
ADDEXP	This outputs the final packed floating add or subtract result. The mantissa is provided by the A Board and the exponent by the D Board. The SELECT field must be set as it would be for a PACKUN operation. If the POST-NORMALIZE FLIP-FLOP is set, then the exponent is incremented by 1 (see §4.9.2).
LDNORM	This loads the normalize operand and the normalize shift count (from the X REGISTER TEST on the A Board, see §3.6) into the D Board. The operand's exponent is stored in EXPONENT LATCH B, and the mantissa is stored in ADD/SUBTRACT/NORMALIZE MANTISSA LATCH B. The SELECT field indicates a rounding or non-rounding normalize. The SPECIAL signal indicates if this normalize must be simulated in Compass (see §4.1.3 for details).

ONMS	This outputs the sign extended normalize mantissa and the normalize shift count to the A Board. The sign extended mantissa is shifted left 1 and includes a round bit inserted into the least significant bit (see §4.10).
ONE	This outputs the final packed normalize result. The mantissa is provided by the A Board and the exponent by the D Board. The SELECT field must be set as it would be for a PACKUN operation. The NORMOVL signal goes active if the normalize exponent underflowed. If NORMOVL goes active, the microcode traps to a Compass routine which stores underflow ($\pm 1/\infty$) into the result register (see §1.10.2).
ENPOP	This outputs a count of bits high (1) in the 60-bit operand supplied by the A Board.
PACKUN	This is used for pack and unpack operations. The SELECT field is used to control sign extension, exponent enabling and mantissa enabling. The ONE and ADDEXP signals also use the PACK/UNPACK hardware.

4.1.2. MULTIPLY SEQUENCER

The MULTIPLY SEQUENCER generates control signals for multiplies. The sequencer is started by the MPYGO signal (see §4.1.1). It first generates the MPY signal; this clears a PARTIAL PRODUCTS ACCUMULATOR and gives the MULTIPLIER ARRAY time to multiply. The PP1, PP2, PP3, PP4, PP5 and PP6 signals are generated in sequence. Each of these accumulates a partial product from the MULTIPLY ARRAY (see §4.8.2). During the PP6 signal, the MULDONE signal goes active, indicating to the microcode that the partial products are all accumulated.

4.1.3. SPECIAL INPUT DETECT

This circuit is used to detect special values for floating point operations. These include overflow ($\pm\infty$), underflow ($\pm 1/\infty$), indefinite ($\pm 0/0$) and integer values. The A Board's X REGISTER TEST circuit supplies the signals which indicate the special exponents and zero mantissas (see §3.6).

The SPECIAL INPUT DETECT circuit generates the following signals:

- SPECIAL** This is valid during LDNORM, LDMUL1, LDDIV1, and LDADD1 (see §4.1.1). If the SPECIAL signal goes active in problem mode, the microcode traps to a Compass simulation of the floating point operation. If the SPECIAL signal goes active in monitor mode, the operation is not legal in monitor mode, and the microcode will hang at location "SOSEXEC" (see Appendix 1).
- MANZERO** This is used during ONE and ONMS; it indicates a ± 0 mantissa for a non-rounding normalize. If a value with a ± 0 mantissa is normalized, or if the exponent underflows during the normalize, the normalize hardware always returns +0.
- BLANKME** This is used during OMULDP and OMULSP; it indicates that the exponent resulting from the multiply will have all bits equal to the sign bit of the result. This is valid during integer multiplies, and during floating multiplies which involve underflow exponents (see §4.1 for a definition of integer multiplies). BLANKME is also used to override the MULOVL signal during integer multiplies; the MULOVL signal indicates that the exponent overflowed or underflowed during a floating point multiply and divide (see §4.7.1). BLANKME is disabled during the iterations of the divide algorithm by the DISBLNK signal (see §4.1.1).
- BLANKMM** This is used during OMULDP and OMULSP; it indicates that the result of the multiply will be positive and have all bits zero. This signal is only active for floating point multiplies which involve underflow exponents. If both input exponents are underflow and if both are not normalized, then it is an integer multiply and BLANKMM will not go active. Floating multiplies always give positive zero results, but integer multiplies can give negative results (this is consistent with Control Data Corporation CYBERs). BLANKMM is disabled during the iterations of the divide algorithm by the DISBLNK signal (see §4.1.1).

This is an overview of special values involved in each of the D Board operations:

Normalize: Normalize is simulated in Compass if either input has an overflow or an indefinite exponent. It results in all bits zero (positive underflow exponent with a zero mantissa) if the exponent underflows during normalization (NORMOVL signal), if the input had an un-normalized mantissa and an underflow exponent (NORMOVL signal), or if it is a non-rounding normalize and the input has a ± 0 mantissa (MANZERO signal). Normalize results in a value equal to the input if the input is an underflow exponent with a normalized mantissa value. The normalize shift count is 0 if the input was already normalized; it is 48 if the input mantissa was zero. The normalize shift count is valid even if the input had an underflow exponent.

Floating add and subtract:

Floating add and subtract are simulated in Compass if either input has an overflow or an indefinite exponent. Inputs with underflow exponents are handled in the hardware; this is important for the common case of all bits zero (underflow exponent with a zero mantissa) plus a floating point number. If the result mantissa overflows 48 bits, the mantissa is shifted right by 1 and the exponent is incremented by 1; if normalized numbers of the same sign are added, the result will always be normalized. Add and subtract mantissa computations are done on inverted mantissas, so that -0 will only result if both input mantissas were -0 (see §1.9).

Multiplies: Multiplies are simulated in Compass if either input has an overflow or an indefinite exponent. Integer multiply is performed if both operands have an underflow exponent (all exponent bits equal the sign bit) and both mantissas are not normalized; otherwise a floating point multiply is performed. In integer multiplies, the result will have an exponent with all bits equal to the result sign bit; integer multiplies are restricted to 48-bit integer inputs and 96-bit integer results. A one-bit normalize is performed on the multiply result if both inputs were normalized floating point numbers; when normalized floating point numbers are multiplied, the result will always be a normalized number. Floating point multiplies involving underflow

exponents are handled in hardware; this is important for the common case of all bits zero (underflow exponent with a zero mantissa) times a floating point number. Integer multiply by zero will result in all bits zero if the inputs have the same sign, and all bits one if the inputs differ in sign; floating multiply by underflow will always result in all bits zero. If the exponent underflows or overflows during a floating multiply operation, the MUOVFL signal will go active, and a trap to Compass software will be executed; the Compass software decides if the result is $\pm$ overflow or $\pm$ underflow.

Divide: Divides are simulated in Compass if either input has an overflow or an indefinite exponent. Divides are also simulated in Compass if the denominator is not normalized (including a zero mantissa), or if the denominator has an underflow exponent. The numerator is normalized by the sequence of microcode words for divide (see §9.20). If the numerator has an underflow exponent or a zero mantissa, the result of the divide will be all bits zero (positive underflow with zero mantissa), independent of the sign of the inputs.

4.1.4. D and E BOARD CONTROL Hardware Details

Detailed D Board drawings, page 2

The FFPU field (bits 6-2 of the microcode word) selects PACKUN when the D Board is not in use; the A Board ignores input coming from the D Board ZBUS.

The FFPU field is used for more than floating point operations; it is used in the B Board's BULK MEMORY ADDRESS CIRCUIT (see §2.7.11).

The ROUND FLIP-FLOP stores the least significant bit of the SELECT field during LDMUL2, DIVLOOK, LDADD2, LDSUB2, and LDNORM. The output of the ROUND FLIP-FLOP is the ROUND signal.

Zero mantissas are detected for divide by the ZERO signal from the A Board's X REGISTER TEST circuit.

4.2. MBUS COMPLEMENT

The input to the MBUS COMPLEMENT is the D Board's QBUS and the output is the MBUS. Multiply, divide, floating add, floating subtract and normalize all use the MBUS COMPLEMENT.

The exponent value on the MBUS is unpacked; bits 58-48 of the MBUS are the signed one's complement exponent value. If bit 59 (the sign bit) of the QBUS is high (1), then the MBUS COMPLEMENT inverts bits 58-48. Bit 58 is inverted an extra time (see §1.10). During LDDIV2, the exponent on the MBUS is inverted again since the exponents are subtracted during divides (see §4.1.1).

The MBUS COMPLEMENT negates the mantissa during LDSUB2; subtracts are the same as adds except that LDSUB2 complements where LDADD2 does not.

The MBUS COMPLEMENT takes the absolute value of the mantissa during LDDIV1, LDDIV2, LDMUL1, LDMUL2 and DIVLOOK (see §4.1.1). This is because the multiply and divide mantissa hardware works with positive numbers only. If the result of a multiply or divide is negative, the final output of the multiply or divide is complemented.

When LDITER1 is active, the input to the multipliers is a positive number. It is complemented to generate 2^{48} minus the number; the result is still interpreted as a positive number (see §9.20.2).

4.2.1. MBUS COMPLEMENT Hardware Details

Detailed D Board drawings, page 6

The complement control for bit 58 (KEXPC01 signal) is the inverse of the complement controls for bits 57-48 (KEXPC01, KEXPC02). In this way bit 58 is complemented an extra time to unpack the exponent.

The LDDIV2 signal complements the EXPC controls, causing another inversion of the exponent.

4.3. EXPONENT LATCHES

The EXPONENT LATCHES hold exponents for multiply, divide, add, subtract and normalize. The input to the EXPONENT LATCHES is the unpacked exponent on the MBUS plus the sign bit from the QBUS.

EXPONENT LATCH A is loaded during LDMUL1, LDDIV1, and LDADD1 (see §4.1.1). EXPONENT LATCH B is loaded during LDMUL2, LDDIV2, LDADD2, LDSUB2 and LDNORM.

4.4. NORMALIZE SHIFT COUNT LATCH

The NORMALIZE SHIFT COUNT LATCH holds the normalize shift count for normalize operations. The input to the NORMALIZE SHIFT COUNT LATCH is NSC05-00 from the A Board X REGISTER TEST circuit (see §3.6). The output of the NORMALIZE SHIFT COUNT LATCH is connected to the output of the EXPONENT LATCH A (see §4.3). The NORMALIZE SHIFT COUNT LATCH indicates the number of positions a mantissa must be shifted left for normalization. If the number is already normalized, then the normalize shift count is 0. If the number has a zero mantissa, then the normalize shift count is 48.

During a normalize operation, the normalize shift count is subtracted from the exponent of the number to be normalized (see § 1.10.1).

4.5. EXPONENT COMPARATOR

The input to the EXPONENT COMPARATOR is from the EXPONENT LATCHES (see §4.3). It generates the following signals:

AGTB This signal is high if EXPONENT A > EXPONENT B

AEQB This signal is high if EXPONENT A = EXPONENT B

4.5.1. EXPONENT COMPARATOR Hardware Details

Detailed D Board drawings, page 7

The most significant input bit of comparator chip E02 is complemented; this is done so that the unsigned comparator chips will work properly.

4.6. ADD/SUBTRACT SELECTION

This circuit is used with the ADDEXP, ADDSHF, ADDSHFL, ADDMAN and ADDMASK signal (see §4.1.1). It selects which add/subtract operand should be used in each of these operations, based on the AGTB signal from the EXPONENT COMPARATOR circuit (see §4.5).

4.7. MULTIPLY/DIVIDE EXPONENT PROCESSING

The MULTIPLY and DIVIDE EXPONENT PROCESSING consists of 4 adders. The first adder adds the two exponents; in divides, the denominator exponent is inverted (see §4.2), so a subtract is performed. The second adder adds 48 to the exponent if the most significant 48 bits of the multipliers are to be read. This adder subtracts 47 if a divide is being performed. The third adder subtracts 1 if a one-bit post-normalizing left shift was performed to normalize the multiply result (see §4.8.6). The last adder is used to add the end-around-carry for the exponents.

If an integer multiply was performed (see BLANKME signal, §4.1.3), the exponent is blanked out (all bits low (0)).

The final step is to pack the exponent. This requires inverting the exponent sign bit (unless the BLANKME signal is active) and then, if the result is negative, inverting all the exponent bits. These exponent bits are then combined with the output of the MULTIPLY PARTIAL PRODUCTS ACCUMULATOR and sent to the A Board (see §4.11).

4.7.1. MULTIPLY/DIVIDE EXPONENT PROCESSING Hardware Details

Detailed D Board drawings, page 8

The 11-bit exponents are sign extended to 12 bits. This aids in the overflow detection for multiply and divide needed to generate the MULOVL signal. If the BLANKME signal is active, an integer multiply is being performed and all the packed exponent result bits are equal to the sign bit.

The BLANKME signal is used to override the MULOVL signal, it is used to master reset K06 and H07 (blanking exponent bits 09-00), and in F20 it is used to blank the exponent sign bit.

K06 and H07 are also used to convert -0 exponents to +0 exponents. Negative zero exponents would erroneously indicate an indefinite (0/0) result (see §1.10.2).

The MULTIPLY/DIVIDE EXPONENT adders are ripple-carry adders. The carry-look-ahead chips are used to generate the ripple-carry for each adder chip. If the result of the multiply or divide is negative, then all the exponent bits are complemented to give a packed result. The result will be negative if the signs of the input operands differ.

4.8. MULTIPLY/DIVIDE MANTISSA PROCESSING

The MULTIPLY ARRAY used for computing partial products during multiplies is a TTL circuit on the E Board. The DIVIDE ROM which is used in the divide algorithm, is also a TTL circuit on the E Board. Multiply/divide mantissa values are converted from ECL to TTL, then partial products are generated on the E Board, then converted from TTL to ECL, and then accumulated on the D Board.

4.8.1. DIVIDE ROM

The iterative divide algorithm requires a table look-up which approximates the reciprocal of the denominator's mantissa (see §9.20). The DIVIDE ROM on the E Board supplies this table.

The output of the DIVIDE ROM is sent directly to the MULTIPLIER ARRAY. This is convenient for the iterative divide algorithm since the table look-up value must be multiplied by the numerator's and denominator's mantissas.

4.8.2. MULTIPLY ARRAY

The E Board provides an array of 9 multipliers. They are used in computing the product of the two mantissas. The two mantissas are broken down into 16-bit chunks, labeled A, B, C and D, E, F (see figure 4.8.2a). Each of the 9 multipliers multiply two of the 16-bit chunks together to give: A•D, A•E, A•F, B•D, B•E, B•F, C•D, C•E and C•F.

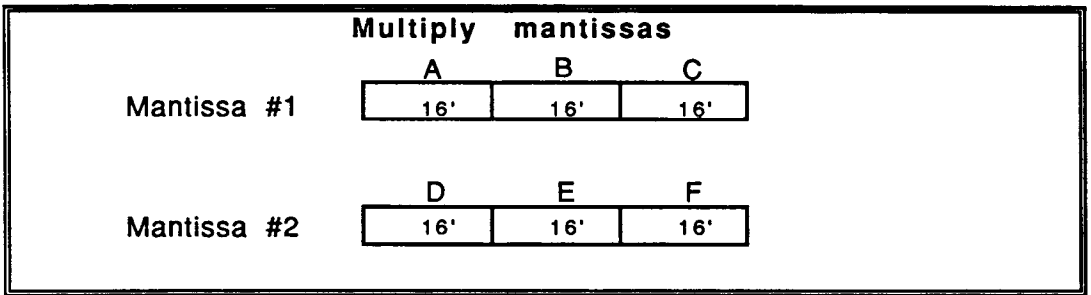


figure 4.8.2a

Some of the multipliers give results which are of low significance in the output, while others have results of higher significance. Figure 4.8.2b shows the significance of each of the multiplier outputs.

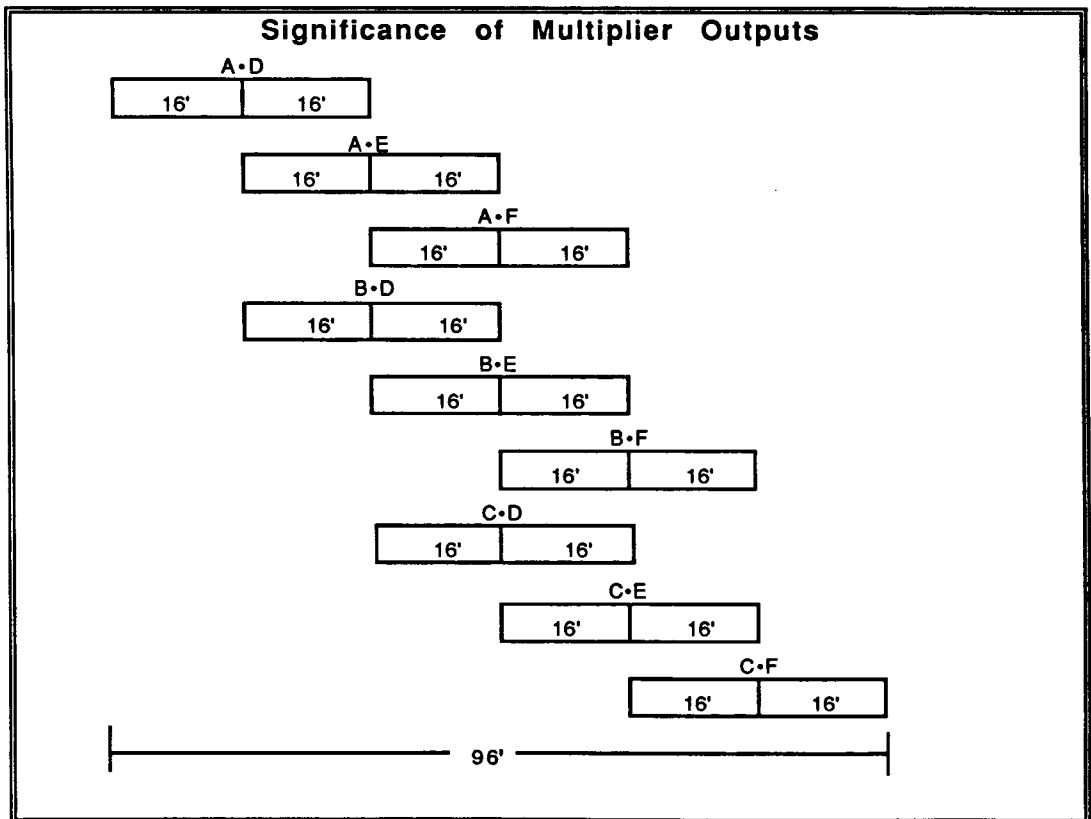


figure 4.8.2b

4.8.3. PARTIAL PRODUCTS

The outputs of the multipliers in the multiplier array are combined to give 6 different partial products. Figure 4.8.3 shows which multiplier outputs are included in each partial product. If a rounded multiply was requested, a round bit is included in bit 46 of partial product 5.

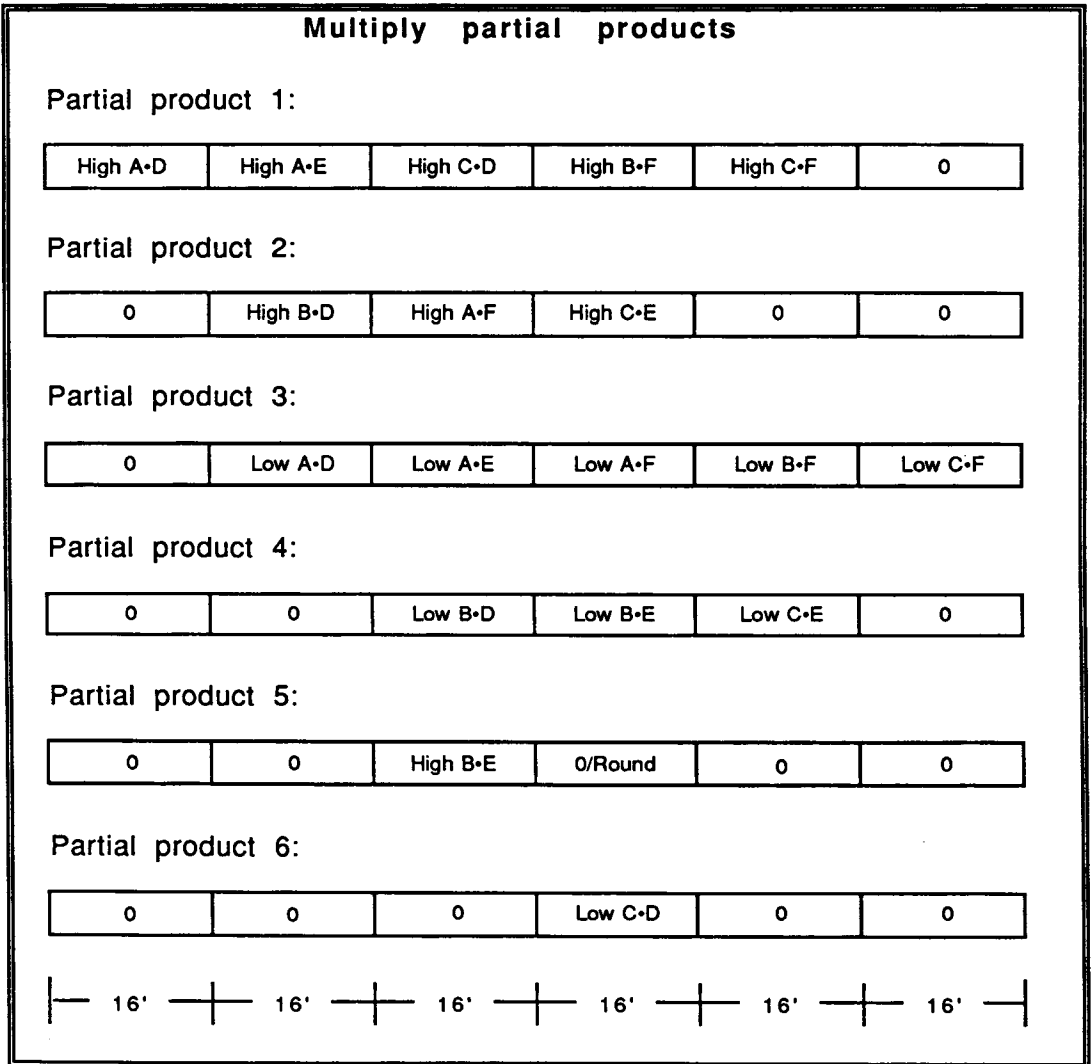


figure 4.8.3

4.8.4. PARTIAL PRODUCTS ACCUMULATOR

In a series of 6 steps, the 6 partial products are added together to give the final 96-bit result. One microcode word jumps to itself while these 6 steps are being performed (see §9.17); the control signals for these 6 steps are controlled by the MULTIPLY SEQUENCER (see §4.1.2).

4.8.5. MULTIPLY COMPLEMENT

The 96-bit complementor at the output of the PARTIAL PRODUCTS ACCUMULATOR inverts the 96-bit result if necessary. Multiplies are always done with the absolute values of the mantissa (see §4.2), so the output must be complemented if the signs of the input operands differed.

4.8.6. MULTIPLY POST-NORMALIZE

If the input operands to the multiply were both normalized and if the 96-bit result is not normalized, a left shift by one bit is performed to normalize the result. If the shift is performed, then the exponent is decremented by 1 (see §4.7). The SELECT field (bits 63-60 of the microcode word) may be used to suppress this one-bit normalize.

4.8.7. MULTIPLY/DIVIDE MANTISSA PROCESSING Hardware Details

Detailed D Board drawings, page 10

ECL to TTL and TTL to ECL converter chips require a 5 volt power supply. Five volts is not available on the back plane of the wire-wrap board, so a normal signal connector is used to supply 5 volts. Each pin on the signal connector has a wire to one of the converter chips. There is a decoupling capacitor between 5 volts and any other supply at each converter chip.

The enables on the TTL to ECL converters generate the 16 bits of 0 in each partial product (see §4.8.3). The CLOCKX and CLOCKY control signals for the MULTIPLIER ARRAY are generated in the top right corner of the drawing. LDMUL2 clocks both the X and Y inputs to the MULTIPLIER ARRAY; this is needed to latch the ROUND, FORMAT, and the other control signals on the multiplier.

Detailed D Board drawings, page 11

The DOUBLE ACCUMULATE FLIP-FLOP is used in generating a double accumulate pulse for the PARTIAL PRODUCTS ACCUMULATOR (see MPYGO §4.1.1). The KDC signal is a 10 ns wide pulse occurring at the leading edge of the FPU DC signal (CMCLK on the B Board, see §2.2). CLOCK1 is ORed with KDC; the rising edge of CLOCK1 is

20 ns after the rising edge of KDC. A wire loop is provided to further delay CLOCK1 to allow adequate time for the accumulator to settle.

The ACCUMULATOR LATCH is master reset by the MPY signal from the MULTIPLY SEQUENCER.

Detailed D Board drawings, page 13

The latch chips, the tri-state controls on the multipliers, and the significance controls on the multipliers are used to control which multiplier outputs are in each partial product (see figure 4.8.3).

4.9. ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING

The ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING consists of 2 ALUs and the POST-NORMALIZE FLIP-FLOP. The ALU control circuitry uses signals from the FPU CONTROL DECODER (see §4.1.1) and the ADD/SUBTRACT SELECTION (§4.6). The output of the ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING is an unpacked exponent, a shift count, or a mask count onto the OUTQBUS.

4.9.1. ALU 1

ALU 1 performs the following functions:

EXPONENT B minus NORMALIZE SHIFT COUNT is used in computing the normalize exponent during ONE (see §4.1.1).

ABSOLUTE VALUE OF (exponent A - exponent B) is used to compute the shift count for the mantissa of the smaller operand during ADDSHF and ADDSHFL in adds and subtracts.

LARGER OF (exponent A and exponent B) is used during ADDEXP to provide the exponent for the final exponent in add/subtract.

NORMALIZE SHIFT COUNT is used during ONMS to provide the shift count for the mantissa to be normalized.

4.9.2. POST-NORMALIZE FLIP-FLOP

The POST-NORMALIZE FLIP-FLOP is used for floating adds and subtracts. It provides a 1 if adding the mantissas overflowed 48 bits and had to be shifted right by one. The POST-NORMALIZE FLIP-FLOP is latched during ADDMAN and its output is used during ADDEXP (see §4.1.1).

4.9.3. ALU 2

The input to ALU 2 is the output of ALU 1. ALU 2 performs the following functions:

ADD POST-NORMALIZE is used during ADDEXP (see §4.1.1). This adds 1 to the exponent if a post-normalize was performed.

PASS SHIFT COUNT is used during ONMS, ADDSHF, ADDSHFL and ADDMASK. This outputs the value from ALU1 unchanged.

60 MINUS SHIFT COUNT is used during ADDSHFL to compute the left shift count. ADDSHFL outputs 60 minus the shift count output by ADDSHF.

SMALLER OF (SHIFT COUNT, 47) is used during ADDMASK. The mask count provided by ADDMASK is normally equal to the shift count provided by ADDSHF, but the mask count never exceeds 47. This limits the precision to the precision of Control Data Corporation's CYBER floating point add.

4.9.4. ADD/SUB/NORMALIZE EXPONENT PROCESSING Hardware Details

Detailed D Board drawings, page 7

The output of the ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING circuit is an unpacked exponent, as opposed to the output from the MULTIPLY/DIVIDE EXPONENT PROCESSING circuit which is a packed exponent.

The 11-bit exponents are sign extended to 12 bits. This aids in the overflow detection needed to generate the NORMOVL signal.

H09 is used during ADDMASK to compare to 47. The AEQB signal is used to avoid confusion resulting from -0.

F05 and F06 are used to convert -0 exponents to +0 exponents. A -0 exponent would erroneously indicate an indefinite (0/0) result (see §1.10.1). F05 and F06 are also used to blank out the exponent if a ± 0 mantissa (MANZERO signal) was used in a non-rounding normalize (see §4.1.3).

4.10. ADD/SUBTRACT/NORMALIZE MANTISSA LATCHES

The ADD/SUBTRACT/NORMALIZE MANTISSA LATCHES hold mantissas for adds, subtracts and normalizes. The inputs to the ADD/SUBTRACT/NORMALIZE MANTISSA LATCHES are signed mantissas from the MBUS. The outputs are the sign extended mantissas shifted left 1 and include a round bit inserted in the least significant bit position.

MANTISSA LATCH A is loaded during LDADD1 and MANTISSA LATCH B is loaded during LDADD2, LDSUB2 and LDMORM (see §4.1.1). The MANTISSA LATCHES are loaded but not used during multiplies and divides.

During rounding adds and subtracts a round bit is inserted in the larger mantissa. A second round bit is inserted in the smaller mantissa if both mantissas are normalized or have unlike signs. The round bits have a value equal to the complement of the sign bit and are inserted immediately to the right of the least significant bit in each mantissa.

During a non-rounding add or subtract a bit equal to the sign bit is inserted immediately to the right of the least significant bit of each mantissa.

During ADDSHF, ADDMAN, ADDSHFL and ADDMASK, the values of the MANTISSA LATCHES are output onto the OUTPBUS. The ADD/SUBTRACT SELECTION circuit supplies control signals which determine whether MANTISSA A or MANTISSA B is read.

During ONMS, the MANZERO signal is used to disable the mantissa latches for non-rounding normalize operations on zero mantissas. This forces all bits to zero if a -0 mantissa is being normalized.

4.11. PACK/UNPACK

The PACK/UNPACK circuit is used for pack and unpack operations. During unpack operations, the input comes from the D Board QBUS, which is connected to the A Board QBUS.

During pack operations, the mantissa comes from the QBUS on the A Board. The exponent comes from the ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING circuit during ONE and ADDEXP; it comes from the A Board's PBUS during PACKUN (see §4.1.1).

The SELECT field (bits 63-60 of the microcode word) controls packing and unpacking operations. SELECT bit 0, used for pack/unpack, enables the mantissa from the QBUS onto the ZBUS. SELECT bit 1, used for unpack, enables sign extension of this mantissa. SELECT bit 2, used for pack, enables the value from the ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING circuit or from the A Board PBUS onto the most significant bits of the ZBUS. SELECT bit 3, used for unpacking the exponent, enables the value from the most significant bits of QBUS onto the least significant bits of ZBUS.

4.12. POPULATION COUNT

The POPULATION COUNT counts the number of high (1) bits in the 60-bit value on the QBUS. The result is output on the PBUS during ENPOP (see §4.1.1).

5. CM BOARD

5.1. EVEN BANK AND ODD BANK

There are 256K words of CM (Central Memory). Half of CM is private memory and half is shared memory. The CM Board contains the private CM and the IOCM Board contains the shared CM.

The CM Board is organized in 2 banks: one for odd addresses and one for even addresses. The address to the ODD BANK is always one more than the address to the EVEN BANK. Both banks may be read or written simultaneously when transferring data between BM (Bulk Memory) and CM. This is called a double-word transfer (see §2.7.7).

The A and B Boards always read or write one CM word at a time. Data from the A Board registers passes through the ALU on the A Board, and through the ZBUS on the B Board before arriving at the CM Board (see the block diagram).

On the CM Board 72 bits are kept for each data word. Sixty of these bits are the data bits actually used by the CPU, eight are for error correction, and four bits are not used by the A and B Boards. The memory chips store inverted values for the error correction bits and they store non-inverted values for the data bits.

5.2. ADDRESS FAN-OUT

The ADDRESS FAN-OUT provides non-inverted address bits to the EVEN BANK of private CM. The address bits to the ODD BANK are the inverted address; this is done to make use of the inverted output pins on the 112 chips used for address fan-out. The ADDRESS FAN-OUT circuit also provides the memory address for the IOCM Board.

5.3. BM DRIVER AND BM RECEIVER

The BM DRIVERS send 2 words of CM at a time (2 times 72 bits) to BM. BM may store both of these words (double-word transfer) or it may store only one of these words (single-word transfer). The BM RECEIVERS receive 2 words at a time from BM. The CM may store both of these words (double-word transfer) or it may store only one of these words (single-word transfers). The data to and from CM is differential data (see §1.11.3).

The address for BM goes directly from the B Board BM ADDRESS circuit to BM; it does not pass through the CM Board (see §2.7).

5.4. IOCM RECEIVER

The IOCM RECEIVER receives 2 words at a time from IOCM. The error correction bits on these two words are in the same form as the error correction bits for private CM.

Both of the words from the IOCM RECEIVER may be sent to BM through the BM DRIVERS. This allows single- or double-word transfers from IOCM to BM (see the block diagram).

The A and B Boards may read one of the words from the IOCM RECEIVER just as the A and B Boards read one of the words from the EVEN or ODD BANK of private CM.

When the A and B Boards write to IOCM, the data does not pass through the CM Board; rather, it goes directly from the B Board to the IOCM Board.

When BM writes to IOCM, only single-word transfers are allowed. The data passes through the CM Board to the B Board and then to the IOCM Board.

5.5. COUNTER

The COUNTER increments at a rate of 32 kHz and has a total of 40 bits. The COUNTER is master reset by writing to one of the CM SPECIAL ADDRESSES (see §2.6.6).

The data written to CM special address 777763 is not used to load the COUNTER: the COUNTER is always set to 0 (see figure 5.5). The least significant 2 bits of the data are high (1) for debugging memory (see §5.6), and the next higher bit is used to disable error correction (§5.9). To read the COUNTER value, read CM special address 777763.

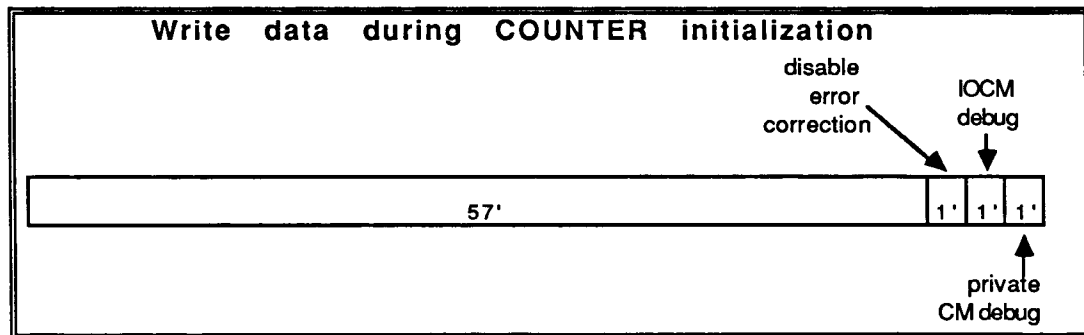


figure 5.5

5.6. TOP 12 BITS MUX

When the A and B Boards read CM or BM, only 60 bits are used of the 72 bits stored for each word (see §5.1). The TOP 12 BITS MUX allows the A and B Boards to read the other 12 bits instead of the 60 bits; this is only used during hardware debugging and memory testing (see figure 5.6).

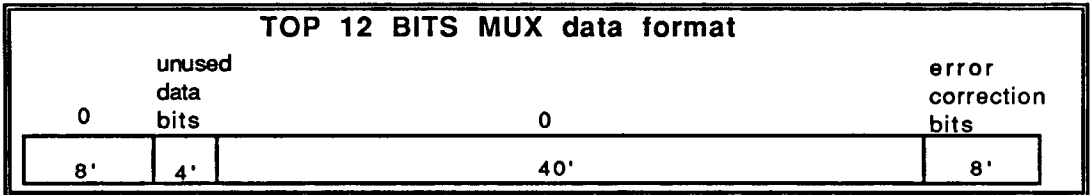


figure 5.6

The TOP 12 BITS MUX has 4 inputs: one input from the EVEN BANK of CM, one input from the ODD BANK of CM, and two inputs from the BM RECEIVERS. At this time the inputs from the BM RECEIVERS are not used.

The TOP 12 BITS MUX is controlled by two bits set during COUNTER initialization (see §5.5). One bit enables the TOP 12 BITS MUX for private CM and the other enables the TOP 12 BITS MUX for IOCM. When the TOP 12 BITS MUX is enabled for private CM, the Compass program must run in IOCM. When TOP 12 BITS MUX is enabled for IOCM, the Compass program must run in private CM. The TOP 12 BITS MUX should never be enabled for both IOCM and private CM.

5.7. CM MUX

The CM MUX chooses between data from CM (even or odd bank), data from BM (one of two words from the BM RECEIVER), the COUNTER (§5.5), and data from the TOP 12 BITS MUX (see §5.6). The output of the CM MUX goes to the ERROR CORRECTION BIT GENERATION circuit and to the ERROR CORRECTION circuit.

Data passes through the CM MUX when the A and B Boards read private CM or IOCM, when the A and B Boards read BM directly (not yet implemented in microcode), when the A and B Boards read the COUNTER, when the A and B Boards write to private CM, and when data is written from BM to IOCM (see the block diagram).

The data passes through the CM MUX during writes from the A and B Boards in order to generate the error correction bits.

Data passes through the CM MUX when data is written from BM to IOCM so that the data may be error corrected before it is sent to the IOCM Board via the B Board. Data which is

transferred from BM to private CM, data transferred from private CM to BM, and data transferred from IOCM to CM is not error corrected. The error correction bits are simply copied along with the data, allowing fast data transfers between the memories. Error correction is delayed until the A and B Boards read the data or the data is written to IOCM.

5.8. ERROR CORRECTION BITS GENERATION

Each of the 8 error correction bits is the parity of 26 data bits. An error correction bit is high (1) if an odd number of its 26 data bits are high.

Table 5.8 shows which error correction bits include each data bit. These are the error correction bit values (in octal) which correspond to data words having only one bit high (1).

Each of the 64 data bits is included in 3 or 5 error correction bits; so a data word with one bit high (1) results in 3 or 5 error correction bits high (1).

Error correction bits for 64 bits of data

DATA BIT #	ERROR CORR. BITS	DATA BIT #	ERROR CORR. BITS	DATA BIT #	ERROR CORR. BITS	DATA BIT #	ERROR CORR. BITS
00	051	16	112	32	222	48	244
01	224	17	045	33	111	49	122
02	062	18	214	34	043	50	310
03	031	19	106	35	221	51	144
04	212	20	242	36	250	52	052
05	105	21	121	37	124	53	025
06	142	22	230	38	046	54	211
07	061	23	114	39	023	55	304
08	150	24	032	40	206	56	241
09	064	25	015	41	103	57	320
10	026	26	205	42	141	58	130
11	013	27	302	43	260	59	054
12	203	28	340	44	070	60	016
13	301	29	160	45	034	61	007
14	334	30	067	46	315	62	163
15	354	31	073	47	316	63	263

table 5.8

5.9. ERROR CORRECTION

When a word is read from memory, 8 error correction bits are included. These 8 error correction bits are EXCLUSIVE-ORed with 8 newly generated error correction bits. If the result, called the syndrome bits, is all zeroes, then no error is detected. If the syndrome bits have an odd number of bits high (1), a single-bit error is detected. If the syndrome bits have an even number of high bits, a double-bit error is detected.

If a single-bit error is detected, then the syndrome bits indicate which bit is in error. If only one bit is high (1) in the syndrome bits, then the corresponding error correction bit is in error. If 3 or 5 error correction bits are high, then table 5.8 may be used to find which bit is in error.

If a double-bit error is detected the error cannot be corrected. If more than two bits are in error, the error may either go undetected or look like a single- or double-bit error.

ERROR CORRECTION may be disabled for testing (see §5.5).

5.10. ERROR FLIP-FLOP

If a single- or double-bit error occurs, then the syndrome bits and the CM address are stored in the B Board's ERROR LATCH. The ERROR LATCH may be read through the ZBUS MUX (see §2.9.4). The ERROR FLIP-FLOP stores whether there is a single-bit error or a double-bit error. The outputs of the ERROR FLIP-FLOP are interrupts for the B Board.

5.11. B BOARD RECEIVER

The input to the B BOARD RECEIVER is inverted data from the B Board and non-inverted error correction bits from the ERROR CORRECTION BITS GENERATION circuit. When the B BOARD RECEIVER is enabled, it outputs non-inverted data and inverted error correction bits to both banks of private CM. The CM stores 64 data bits; 60 of these come from the B Board and 4 are always zero.

Data for IOCM is sent directly from the B Board to the IOCM Board.

6 . IOCM BOARD

6.1. EVEN BANK AND ODD BANK

There are 256K words of CM (Central Memory). Half of CM is for private memory and half is shared memory. The CM BOARD contains the private CM and the IOCM BOARD contains the shared CM.

The IOCM Board is organized in 2 banks: one for odd addresses and one for even addresses. An address to the ODD BANK is always one more than an address to the EVEN BANK. Both banks may be read simultaneously when transferring data from IOCM to BM (Bulk Memory); this is called a double-word transfer (see §2.7.7). Data from IOCM passes through the CM Board's IOCM RECEIVER and the CM Board's BM DRIVER before arriving at the BM Boards. Only single-word transfers may be performed when transferring from BM to IOCM. The data from BM passes through the CM Board's ERROR CORRECTION circuit to the B Board and then to the IOCM Board (see the block diagram and §5.7).

The A and B Boards always read or write one IOCM word at a time. Data from the A Board registers passes through the ALU on the A Board and through the ZBUS on the B Board before arriving at the IOCM Board. Data from IOCM passes through the CM Board's ERROR CORRECTION circuit and the B Board ZBUS before arriving at the A Board registers.

The IOCM Board provides 4 PPs (Peripheral Ports). These provide access to IOCM for PP microcomputers. Each may read or write one quarter word (16 of 64 data bits) of IOCM at a time.

On the IOCM Board 96 bits are kept for each word: 64 of these are data bits, and 32 are error correction bits. More error correction bits are needed on the IOCM Board than on the private CM Board because error correction must be done on 64 data bits at a time for the A and B Boards, and on 16 data bits at a time for the PPs. All 64 of the data bits are used by the PPs, but only 60 of the data bits are used by the A and B Boards.

6.2. IO REQUEST PROCESSOR

The IO REQUEST PROCESSOR determines whether the A and B Boards or one of the 4 PPs will be the next to be allowed access to IOCM. The input to the IO REQUEST PROCESSOR is 5 request lines: one from each of the 4 PPs, and one from the A and B Boards. Access to IOCM is granted with equal priority given to each of the request lines.

6.2.1. IO REQUEST PROCESSOR Hardware Details

Detailed IOCM Board drawings, page 2

B27 and A27 are flip-flops holding the 5 requests. These are active whenever there is a new request which has not been processed. C29 and C28 hold the acknowledge signals for each of the 5 requests. The acknowledge signal goes active when access is granted and is deactivated when access is no longer requested.

B26 is a shift matrix which gives lowest priority to the request line which was most recently granted access.

B24 is an adder whose output selects the next access request to be granted. If there are no pending requests, B24 adds 0.

B23 is a latch holding the access request being granted. This latch is only clocked when a new request is granted.

B21 is a decoder used to activate the acknowledge flip-flops and deactivate the new request flip-flops. This decoder is only activated at the end of a memory cycle (during the WRITE signal, see §6.3). The acknowledge signal going to the A and B Boards, called UGOTIT on the B Board, is delayed so that the data has time to make it to the A Board registers before the microcode word completes execution (see §2.2.6).

6.3. MEMORY TIMING GENERATOR

Once access to IOCM has been granted to one of the 4 PPs or to the A and B Board, this circuit generates the WRITE signal which controls the timing for the memory access.

The WRITE signal is used as a write pulse to the memory chips during memory write cycles. It is also used to signal the completion of the memory cycle (see §6.2.1) and to latch the data read during read cycles (see §6.10 and §6.13).

6.4. ADDRESS MUX

The ADDRESS MUX multiplexes the addresses from the 4 PPs and from the B Board. The PP addresses have two more address bits than the addresses from the B Board, since the PPs access IOCM 16 data bits at a time, and the B Board accesses 64 data bits at a time (with 4 bits unused). The least significant 2 bits of the PP address are both low (0) to select the most significant 16 bits of a 64-bit word. The multiplexer is controlled by the output from the IO REQUEST PROCESSOR (see §6.2).

6.5. ADDRESS FAN-OUT

The address bits to the EVEN BANK are the non-inverted address. The address bits to the ODD BANK are the inverted address; this is done to make use of the inverted output pins on the 112 chips used for address fan-out.

The 2 least significant bits from the ADDRESS FAN-OUT circuit select which quarter word (16 of 64 data bits) is to be accessed; these are always 0 for A and B Board accesses. The next higher significant bit selects between EVEN and ODD BANK.

6.6. PP ADDRESS RECEIVERS and PP OFFSET ADDERS

The PP ADDRESS RECEIVERS are differential receivers for addresses from each PP. The PP OFFSET ADDERS add a 4-bit offset to the most significant bits of the PP address.

Switches on the PP BUS INTERFACE Boards specify a low and high address in the PP microcomputer address space. All addresses in this range access IOCM. The PP OFFSETs on the IOCM Board (a set of switches) are added to the PP's addresses. The address range in IOCM depends on a combination of the switch settings on the PP BUS INTERFACE Boards and on the switch settings on the IOCM board.

The HIO and LIO fields on the B Board (see §1.7 and §2.6.4) specify what part of IOCM a Compass program is allowed to access. This access is granted in multiples of 8K words. The LOW and HIGH ADDRESS SWITCHES on the PP BUS INTERFACE Boards, together with the PP OFFSET SWITCHES on the IOCM Board, also allocate IOCM in memory in multiples of 8K words (32K of 16-bit words).

6.7. B BOARD DATA RECEIVER

The B BOARD DATA RECEIVER receives inverted data from the B Board. When the receiver is enabled, it outputs non-inverted data to the EVEN and ODD BANKS of IOCM and to the ERROR CORRECTION BIT GENERATION circuit (see §6.8). The B BOARD DATA RECEIVER is enabled by control signals from the IO REQUEST PROCESSOR (see §6.2). The input to the B BOARD DATA RECEIVER is 60 bits of data. The output is 64 bits of data; the extra 4 bits are zero.

6.8. ERROR CORRECTION BIT GENERATION

The input to the ERROR CORRECTION BIT GENERATION circuit comes from the B BOARD DATA RECEIVER (see §6.7). Eight error correction bits are generated for every

16 data bits. Each of the 8 error correction bits is the parity of several of the 16 data bits. Each of the 16 data bits is included in 3 or 5 error correction bits.

Table 6.8 shows which error correction bits include each data bit. These are the error correction bit values (in octal) which correspond to having only one bit high in the 16 data bits.

The PP BUS INTERFACE Boards generate these error correction bits for accesses from the PPs.

Error correction bits for 16 bits of data

DATA BIT NUMBER	ERROR CORR. BITS
00	244
01	122
02	310
03	144
04	052
05	025
06	211
07	304
08	241
09	320
10	130
11	054
12	016
13	007
14	163
15	263

table 6.8

6.9. COMBINE ERROR CORRECTION BITS

The COMBINE ERROR CORRECTION BITS circuit combines the error correction bits from each of the 16 data bits to give 8 error correction bits for all 64 data bits. These final 8 error correction bits are in the same form as error correction bits on the CM Board (see §5.8).

Error correction bits may be combined simultaneously for the ODD BANK and the EVEN BANK of IOCM. This supports double-word transfers (see §2.7.7 and §6.1).

6.10. LATCH/BUFFER TO CM BOARD

The LATCH/BUFFER TO CM BOARD sends 2 IOCM words to the CM Board. Each of these words include 64 data bits from IOCM and 8 error correction bits from the COMBINE CHECK BITS circuit. The output from the LATCH/BUFFER TO CM BOARD goes to the A and B Boards via the CM Board's ERROR CORRECTION circuit, or to BM via the CM Board's IOCM RECEIVER and BM DRIVERS.

The LATCH/BUFFER TO CM BOARD latches the words from IOCM as the acknowledge signal is activated; the words are held until the B Board deactivates its IO REQUEST line. Since the data is stored in the latch, the IOCM Board may go on to grant requests from PPs before the B Board is done with the data and deactivates its IO REQUEST line.

6.11. PP DATA RECEIVER AND PP DATA SHIFTER

The PP DATA RECEIVER receives 16 inverted data bits with 8 inverted error correction bits from each PP. These are differential signals.

The PP DATA SHIFTER is a shift matrix which performs 3 functions: (1) it selects data from one of four PPs under control of signals from the IO REQUEST PROCESSOR (see §6.2), (2) it sends the data to the proper quarter word of the EVEN or ODD BANK of IOCM under the control of the least significant 3 address bits from the ADDRESS FAN-OUT circuit (see §6.5), and (3) it disables the output if no PP is selected.

6.11.1. PP DATA RECEIVER AND PP DATA SHIFTER Hardware Details

Detailed IOCM Board drawings, page 3

The 150s following the shift matrix are used when no PP is selected. The shift matrix chips are only capable of disabling 7 of their 8 outputs; the 150s are used to disable the 8th output of each chip.

6.12. PP QUARTER WORD SELECT

The PP QUARTER WORD SELECT circuit selects a quarter word (16 bits with 8 error correction bits) from the EVEN BANK or ODD BANK of IOCM; this data is sent to a PP.

The PP QUARTER WORD SELECT is controlled by the least significant 3 bits from the ADDRESS FAN-OUT circuit (see §6.5).

6.13. PP LATCH/DRIVERS

The PP LATCH/DRIVERS send 16 data bits plus 8 error correction bits to each PP BUS INTERFACE Board. This is inverted differential data. Error correction is performed on the 16 data bits on the PP BUS INTERFACE Board. This error correction is similar to the CM Board's ERROR CORRECTION (see §5.8, §5.9) except that table 6.8 is used.

The PP LATCH/DRIVERS holds the data from IOCM as soon as the acknowledge signal is activated; the data remains there until the next time the PP accesses IOCM. Since the data is stored in the latch, the IOCM Board may go on to grant other access requests before the PP deactivates its request lines.

6.14. INTERRUPT BUFFER

The INTERRUPT BUFFER forwards interrupt signals between the PPs and the B Board (see §2.6.6 and §2.6.8). The interrupt signals are differential between the IOCM Board and the PPs.

7. BULK MEMORY DATA BOARDS

7.1. TTL DYNAMIC MEMORY: BANK 0 and BANK 1

BM (Bulk Memory) data is stored on 6 boards: BM0, BM1, BM2, BM3, BM4, and BM5. BM has 64 million words. Each word in BM is 72 bits long. Sixty-four of these bits are data bits and 8 of these bits are error correction bits. The A and B Boards only use 60 of the 64 data bits.

The most significant BM address bit (bit 26) is not used; it is for future expansion. BM address bit 25 selects between BM banks: BANK 0 and BANK 1. Address bits 24-03 select a group of 8 BM words, called BM records. Address bits 02-00 select the word within a record.

Each BM Data Board holds 12 bits of each word (see figure 8.1), or, equivalently, 12 times 8 bits of each record; each data board is divided into BANK 0 and BANK1 (see figure 8.1).

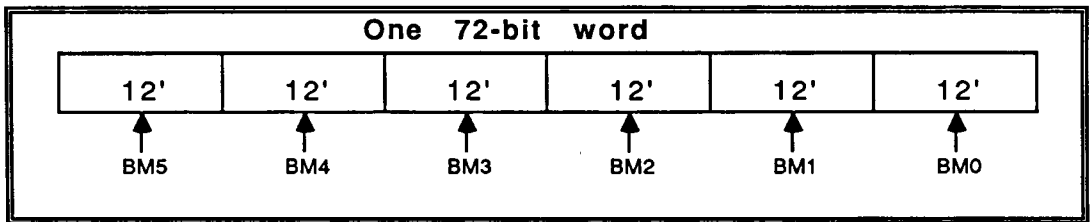


figure 8.1

The BM is implemented with 4 meg by 1 dynamic TTL RAMS. 1 meg by 1 RAM chips may be used instead, with the result that the BM will have a total of 16 million words (2 million records), and address bit 23 will be used to select between BANK 0 and BANK 1.

For details on error correction on BM data, see §5.7.

7.2. TTL TO ECL CONVERTERS

The TTL TO ECL CONVERTERS convert output from the TTL dynamic RAMS to ECL. The TTL TO ECL CONVERTERS are enabled when BM is being read and they are disabled when BM is being written.

7.3. ECL TO TTL CONVERTERS and RESISTORS

The ECL TO TTL CONVERTERS convert input data, addresses, and controls of the TTL dynamic RAMS from ECL to TTL. They are followed by series RESISTORS to minimize reflections on the TTL signals. The addresses and control signals come from the Bulk Memory Controller Board (see Chapter 8). On the BM Data Boards, the addresses and control signals are fanned-out in ECL before being converted to TTL.

7.4. SCRATCH PAD LATCHES

The SCRATCH PAD LATCHES hold one BM record. During BM writes, the SCRATCH PAD LATCHES are loaded one or two words at a time from the CPU (see §2.7.7). When the CPU has completed loading the SCRATCH PAD LATCHES, all of the SCRATCH PAD LATCHES are written to the dynamic RAMs simultaneously. The CPU may load only some of the SCRATCH PAD LATCHES; in this case, only the SCRATCH PAD LATCHES which have been loaded are written to the dynamic RAMs.

During BM reads, all of the SCRATCH PAD LATCHES are loaded simultaneously from the dynamic RAMs. The CPU then reads the SCRATCH PAD LATCHES one or two words at a time. The CPU need not read all of the SCRATCH PAD LATCHES.

7.5. DATA IN RECEIVER and DATA IN SHIFTER

The DATA IN RECEIVER receives 12 bits from 2 words of data from each of 4 CPUs (see §7.1). This data comes over differential lines from the BM DRIVERS on each of the CPU's CM Boards (§5.3).

If a double-word transfer is performed, then the 12 bits of both words from the DATA IN RECEIVER will be loaded into the SCRATCH PAD LATCHES. If a single-word transfer is performed, then the 12 bits of only one word from the DATA IN RECEIVER is loaded into the SCRATCH PAD LATCHES.

The DATA IN SHIFTER is a shift matrix which performs 3 functions: (1) it selects data from 1 of 4 CPUs under control of signals from the BM REQUEST PROCESSOR on the BMC Board (see §8.1) and the least significant 3 address bits, (2) it sends the data to the proper scratch pads, and (3) its outputs are disabled when a BM read is being performed.

7.5.1. DATA IN RECEIVER and DATA IN SHIFTER Hardware Details

Detailed BM Data Board drawings, page 1

The 150s following the DATA IN SHIFTER are used during BM reads. The shift matrix chips are only capable of disabling 7 of their 8 outputs. The 150s are used to disable the output of each shift matrix chip.

7.6. DATA OUT SHIFTER

The DATA OUT SHIFTER selects 12 bits of 2 words from the SCRATCH PAD LATCHES to be sent to a CPU. The DATA OUT SHIFTER is controlled by signals from the BMC Board's BM REQUEST PROCESSOR and the least significant 3 address bits.

7.7. DATA OUT DRIVERS

The DATA OUT DRIVERS send 12 bits of 2 words to the 4 CPUs. The data goes over differential lines to the BM RECEIVERS on each of the CPU's CM Boards (see §5.3).

When a CPU is not reading BM, the DATA OUT DRIVER for that CPU is disabled.

If a double-word transfer is performed, then the 12 bits of both words from the DATA OUT DRIVER are used by the CPU. If a single-word transfer is performed, then the 12 bits of only one word is used by the CPU.

8 . BULK MEMORY CONTROLLER BOARD

8.1. BM REQUEST PROCESSOR

The BMC Board (Bulk Memory Controller Board) provides address and control signals for the Bulk Memory Data Boards (BM0-BM5).

The BM REQUEST PROCESSOR determines which of the 4 CPUs will be the next to access BM. The inputs to the BM REQUEST PROCESSOR are 9 request signals: 4 high priority requests from each CPU, 4 low priority requests from each CPU, and a refresh request from the REFRESH REQUESTS COUNTER circuit (see §8.2). There are also 5 end request signals: 4 from each CPU and 1 from the REFRESH REQUESTS COUNTER circuit.

The high priority requests indicate that a CPU requests a transfer of 8 words or less. The low priority requests indicate that a CPU requests a longer transfer.

The refresh request indicates that the TTL dynamic RAMS on the BM Data Boards must be refreshed. This request has the same priority as the CPU's high priority requests.

8.1.1. BM REQUEST PROCESSOR Hardware Details

Detailed BMC Board drawings, page 2

K19, L19, and M19 are flip-flops holding the 9 refresh requests. These are activated whenever there is a new request, and deactivated on the end request signal.

L18 and M18 are shift matrix chips which give the lowest priority to the CPU which was most recently granted access.

L17 is an adder whose output selects the next access request to be granted. If there are no pending requests, L17 adds zero.

M16 is a latch holding the access request being granted. This latch is clocked only when a new request is granted.

K16 is a decoder used to generate reply signals to the CPUs. This decoder is de-activated if there are no pending requests.

8.2. REFRESH REQUEST COUNTER

The REFRESH REQUEST COUNTER guarantees a refresh for the BM Data Board's dynamic RAMs at an average rate of 32,000 per second. The positive edge of the 32 kHz

clock is used to increment the REFRESH REQUEST COUNTER. Whenever the REFRESH REQUEST COUNTER holds a non-zero value, it generates a refresh request which is sent to the BM REQUEST PROCESSOR. Whenever the BM REQUEST PROCESSOR grants a refresh request, the REFRESH REQUEST COUNTER is decremented.

8.3. REFRESH ADDRESS COUNTER

The REFRESH ADDRESS COUNTER increments each time a refresh request is granted by the BM REQUEST PROCESSOR (see §8.1). This provides the address for the dynamic RAMS during refresh. It is multiplexed with the address from the CPUs.

8.4. RECEIVERS and ADDRESS/CONTROL MUX

The RECEIVERS receive differential signals with address and control information from each of the 4 CPUs (see §2.7).

The ADDRESS/CONTROL MUX selects the address and control signals from the proper CPUs. The ADDRESS/CONTROL MUX is controlled by signals from the BM REQUEST PROCESSOR.

The ADDRESS/CONTROL MUX provides a second function for addresses. It multiplexes the row address bits and the column address bits for the BM Data Board's dynamic RAM chips. This is done with control signals from the DYNAMIC RAM TIMING circuit (see §8.8). There are 11 row address bits and 11 column address bits.

8.4.1. RECEIVERS and ADDRESS/CONTROL MUX Hardware Details

Detailed BMC Board drawings, page 1

R22 is plugged in if 4 meg by 1 dynamic RAM chips are used on the BM Data Boards.

P22 is plugged in if 1 meg by 1 dynamic RAM chips are used on the BM Data Boards.

R22 and P22 determine which address bit is used to select between BANK0 and BANK 1 (see §7.1).

The CONTROL MUX is disabled during refreshes.

8.5. SHIFTER CONTROLS

The SHIFTER CONTROLS circuit generates control signals for the shifters on the BM Data Boards (see §7.5 and §7.6). The SHIFTER CONTROLS must take into account which CPU has BM and which SCRATCH PAD is to be accessed (least significant 3 address bits). The inputs to this circuit come from the BM REQUEST PROCESSOR.

8.6. SCRATCH PAD CLOCKS

The SCRATCH PAD CLOCKS circuit generates the clock signals for the SCRATCH PADS on the BM Data Boards (see §7.4). The least significant 3 address bits specify which SCRATCH PAD is to be loaded.

During BM writes, if a double-word transfer is being performed, the least significant 3 address bits plus 1 specify the second SCRATCH PAD to be loaded.

During BM reads, all the SCRATCH PADS are loaded at one time since a whole record of BM is read simultaneously from the dynamic RAMS. The CPU need not read all of the SCRATCH PADS.

8.7. ACTIVE SCRATCH PAD FLIP-FLOPS

The ACTIVE SCRATCH PAD FLIP-FLOPS keep track of which SCRATCH PADS have been loaded by the CPU during BM writes. When the CPU has completed loading SCRATCH PADS, the data from the SCRATCH PADS is written into the dynamic RAMS. The ACTIVE SCRATCH PAD FLIP-FLOPS allow writes only to those dynamic RAMS which correspond to SCRATCH PADS loaded by the CPU.

8.7.1. ACTIVE SCRATCH PAD FLIP-FLOPS Hardware Details

Detailed BMC Board drawings, page 2

The ACTIVE SCRATCH PAD FLIP-FLOPS are high (1) to indicate that a SCRATCH PAD has not been loaded by the CPU. When a CPU's request for BM is first granted, the ACTIVE SCRATCH PAD FLIP-FLOPS are all set high (1) by flip-flop F19. As the CPU loads the SCRATCH PADS, the corresponding ACTIVE SCRATCH PAD FLIP-FLOPS are individually set low (0).

When the microcode specifies that the values in the SCRATCH PADS be written to the dynamic RAMS, the values from the ACTIVE SCRATCH PAD FLIP-FLOPS are stored in a latch (E15 and D15), and the ACTIVE SCRATCH PAD FLIP-FLOPS are all set high (1).

The output of the latch is used to control the writes to the dynamic RAMs, and the ACTIVE SCRATCH PAD FLIP-FLOPS are ready to keep track of which SCRATCH PADS are loaded in the next record.

8.8. DYNAMIC RAM TIMING

The DYNAMIC RAM TIMING circuit generates 4 control signals: RAS (Row Address Strobe), CAS (Column Address Strobe), Write Enable, and Address Multiplex (see figures 8.8a, 8.8b, and 8.8c).

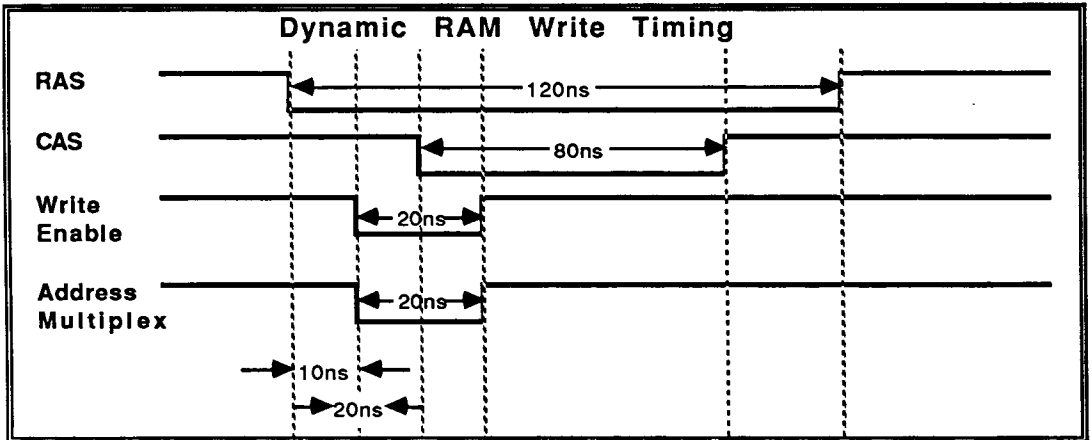


figure 8.8a

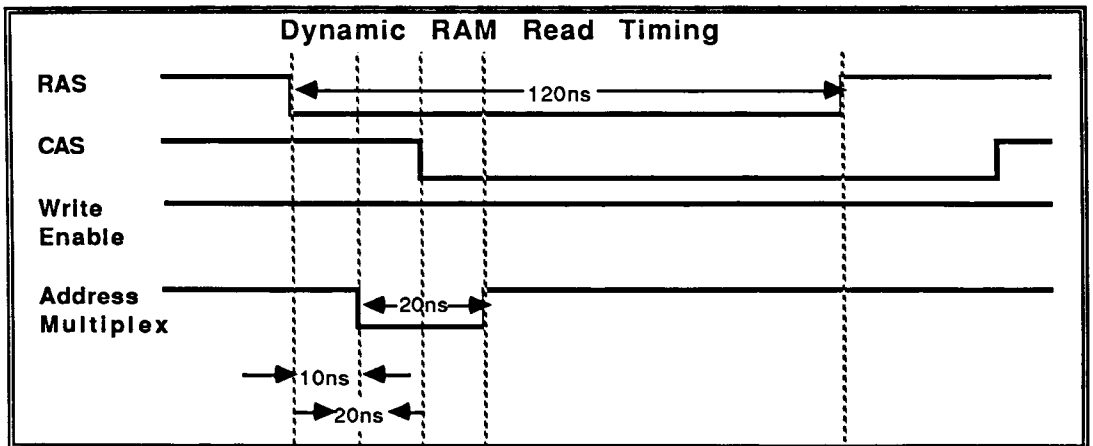


figure 8.8b

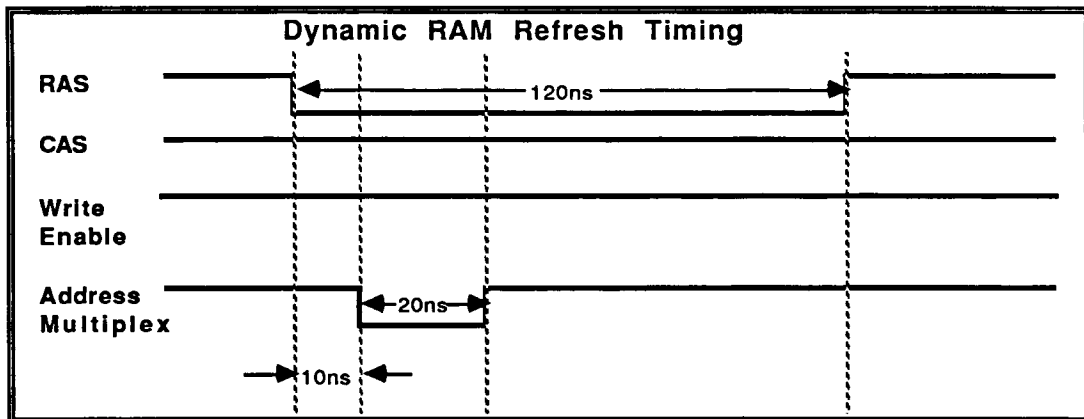


figure 8.8c

8.8.1. DYNAMIC RAM TIMING Hardware Details

Detailed BMC Board drawings, page 2

The dynamic RAM timing is generated by use of 2 shift registers (G18 and G19) and 3 flip-flops (on H19). A 10 ns pulse ripples through the shift register. By connecting the proper outputs of the shift registers to the set and clear pins on the flip-flops, timing may be controlled to 10 ns accuracy.

The shift register is started by a CPU memory cycle (STBM signal) or by a refresh memory cycle (KREFRSH signal).

During reads, the CAS signal waits to go high until the microcode specifies that the SCRATCH PADS be loaded; in this way the BM waits until the CPU has completed reading the previous data from the SCRATCH PADS.

During refreshes, CAS does not go low; this is the purpose of the KREFRSH signal on H19.

8.9. FAN-OUT

The FAN-OUT circuit generates identical copies of the address and control signals to each of the 6 BM Data Boards.

9. MICROCODE TO IMPLEMENT COMPASS

9.1. Compass instructions

Each Compass instruction is implemented by a sequence of microcode words. The formats for Compass instructions are given in §2.3. Table 9.1 gives a brief description of all the Compass instructions implemented on the Zephyr. The Compass Reference Manual [7] gives a detailed explanation of all the instructions except XC, XP, XT and XJ. Each microcode sequence starts at the microcode memory location specified by the first column in the table; this microcode memory location is C0 hex plus the first six bits of the Compass instruction (see NEXTOP jump, §9.1.7). Many of the microcode sequences are only one word long, but some are several words long. Some microcode sequences use the i field of the Compass instruction to further decode the instruction (see i field jumps, §9.1.7).

Appendix 1 lists the microcode which implements Compass. Each microcode word is specified by a series of microcode field names and their values (see §1.8 for an overview of the microcode). The values are specified by named constants which are defined in Appendix 2.

When reading a description of each of the Compass instructions, the reader should follow along in the microcode listing in Appendix 1 and follow the path of the data on the block diagram.

9.2. Start-up

When the Zephyr is started-up, the microcode begins execution at address 0. In Appendix 1, the 3 start-up microcode words follow the macro definitions at the beginning of the listing. The hardware automatically starts-up in monitor mode (see §1.7). The microcode begins to execute a Compass program at address 400100 octal.

9.3. Set Instructions

The set instructions perform 18-bit adds and subtracts. The A, B and X registers and 18-bit constants (the K field) from the Compass instruction are used in the set instructions. X registers are 60-bit registers, but only the least significant 18 bits are read during set instructions. Two distinct values are used for the QT and PT fields when reading X registers. X60 is used to read all 60 bits and X18 is used to read only the least significant 18 bits.

Zephyr Compass Instructions

MICRO-CODE ADDRESS (HEX)	INSTRUCTION (OCTAL)	OPERATION	DESCRIPTION
C0 A8	00000	PS	Program stop †
C0 A9	001jk	XCj Xk	(Xk) to Cj *
C0 AA	002jk	XXj Ck	(Ck) to Xj *
C0 AB	003jk	XPj Xk	Set breakpoint: (Xk) to Pj *
C0 AC	004jk	XXj Pk	(Pk) to Xj *
C0 AD	005jk	XTj Xk	(Xk) to Tj *
C0 AE	006jk	XXj Tk	(Tk) to Xj *
C1 B0	0100K	RJ K	Return jump to K
C1 B1	011jK	RE Bj±K	Read (Bj) plus K words from BM at (X0) to CM at (A0)
C1 B2	012jK	WE Bj±K	Write (Bj) plus K words to BM at (X0) from CM at (A0)
C1 B3	013jK	XJ Bj±K	Exchange jump to Bj+K
C1 B6	016jk	XXj Xj+Tk	(Xj) plus (Tk) to Xj †
C1 B7	017jK	XXj Xj, K	(Xj) plus CM at K to Xj †
C2	02iiK	JP Bi+K	Jump to K plus (Bi)
C3	030jK	ZR Xj, K	Jump to K if (Xj)=0
C3	031jK	NZ Xj, K	Jump to K if (Xj)≠0
C3	032jK	PL Xj, K	Jump to K if (Xj) sign is plus
C3	033jK	MI Xj, K	Jump to K if (Xj) sign is minus
C3	033jK	NG Xj, K	Jump to K if (Xj) sign is minus
C3	034jK	IR Xj, K	Jump to K if (Xj) in range
C3	035jK	OR Xj, K	Jump to K if (Xj) not in range
C3	036jK	DF Xj, K	Jump to K if (Xj) definite
C3	037jK	ID Xj, K	Jump to K if (Xj) indefinite
C4	04ijK	EQ Bi, Bj, K	Jump to K if (Bi)=(Bj)
C5	05ijK	NE Bi, Bj, K	Jump to K if (Bi)≠(Bj)
C6	06ijK	GE Bi, Bj, K	Jump to K if (Bi)≥(Bj)
C6	06ijK	LE Bi, Bj, K	Jump to K if (Bi)≤(Bj)
C7	07ijK	LT Bi, Bj, K	Jump to K if (Bi)<(Bj)
C7	07ijK	GT Bj, Bi, K	Jump to K if (Bj)>(Bi)
C8	10ijj	BXi Xj	Copy (Xj) to Xi
C9	11ijk	BXi Xj*Xk	Logical AND of (Xj), (Xk) to Xi
CA	12ijk	BXi Xj+Xk	Logical OR of (Xj), (Xk) to Xi
CB	13ijk	BXi Xj-Xk	Logical XOR of (Xj), (Xk) to Xi
CC	14ikk	BXi -Xk	Copy complement of (Xk) to Xi
CD	15ijk	BXi -Xk*Xj	Logical AND of (Xj), complement of (Xk) to Xi
CE	16ijk	BXi -Xk+Xj	Logical OR of (Xj), complement of (Xk) to Xi
CF	17ijk	BXi -Xk-Xj	Logical XOR of (Xj), complement of (Xk) to Xi
D0	20ijk	LXi jk	Left shift (Xi) by ±jk
D1	21ijk	AXi jk	Right shift (Xi) by ±jk
D2	22ijk	LXi Bj, Xk	Left shift (Xk) by (Bj) to Xi
D3	23ijk	AXi Bj, Xk	Right shift (Xk) by (Bj) to Xi
D4	24ijk	NXi, Bj Xk	Normalize (Xk) to Xi and Bj
D5	25ijk	ZXi, Bj Xk	Round and normalize (Xk) to Xi and Bj

* only available in monitor mode

† only available in problem mode

table 9.1

Zephyr Compass Instructions

MICRO-CODE ADDRESS (H E X)	INSTRUCTION (OCTAL)	OPERATION	DESCRIPTION
D6	26ijk	UXi, Bj Xk	Unpack (Xk) to Xi and Bj
D7	27ijk	PXi Bj, Xk	Pack (Xk) and (Bj) to Xi
D8	30ijk	FXi Xj+Xk	Sum of (Xj) plus (Xk) to Xi †
D9	31ijk	FXi Xj-Xk	Difference of (Xj) minus (Xk) to Xi †
DA	32ijk	DXi Xj+Xk	Double-precision sum of (Xj) plus (Xk) to Xi †
DB	33ijk	DXi Xj-Xk	Double-precision difference of (Xj) minus (Xk) to Xi †
DC	34ijk	RXi Xj+Xk	Rounded sum of (Xj) plus (Xk) to Xi †
DD	35ijk	RXi Xj-Xk	Rounded difference of (Xj) minus (Xk) to Xi †
DE	36ijk	IXi Xj+Xk	Integer sum of (Xj) plus (Xk) to Xi
DF	37ijk	IXi Xj-Xk	Integer difference of (Xj) minus (Xk) to Xi
E0	40ijk	FXi Xj*Xk	Product of (Xj) to (Xk) to Xi
E1	41ijk	RXi Xj*Xk	Rounded product of (Xj) times (Xk) to Xi
E2	42ijk	IXi Xj*Xk	Integer product of (Xj) times (Xk) to Xi
E2	42ijk	DXi Xj*Xk	Double-precision product of (Xj) times (Xk) to Xi
E3	43ijk	MXi $\pm jk$	Form mask of $\pm jk$ bits in Xi
E4	44ijk	FXi Xj/Xk	Divide (Xj) by (Xk) to Xi †
E5	45ijk	RXi Xj/Xk	Rounded divide (Xj) by (Xk) to Xi †
E6	46000	NO	Pass (do nothing)
E7	47ikk	CXi Xk	Population count of (Xk) to Xi
E8	50ijK	SAi Aj $\pm$ K	(Aj) plus K to Ai
E9	51ijK	SAi Bj $\pm$ K	(Bj) plus K to Ai
EA	52ijK	SAi Xj $\pm$ K	(Xj) plus K to Ai
EB	53ijk	SAi Xj+Bk	(Xj) plus (Bk) to Ai
EC	54ijk	SAi Aj+Bk	(Aj) plus (Bk) to Ai
ED	55ijk	SAi Aj-Bk	(Aj) minus (Bk) to Ai
EE	56ijk	SAi Bj+Bk	(Bj) plus (Bk) to Ai
EF	57ijk	SAi Bj-Bk	(Bj) minus (Bk) to Ai
F0	60ijK	SBi Aj $\pm$ K	(Aj) plus K to Bi
F1	61ijK	SBi Bj $\pm$ K	(Bj) plus K to Bi
F2	62ijK	SBi Xj $\pm$ K	(Xj) plus K to Bi
F3	63ijk	SBi Bk+Xj	(Bk) plus (Xj) to Bi
F4	64ijk	SBi Aj+Bk	(Aj) plus (Bk) to Bi
F5	65ijk	SBi Aj-Bk	(Aj) minus (Bk) to Bi
F6	66ijk	SBi Bj+Bk	(Bj) plus (Bk) to Bi
F7	67ijk	SBi Bj-Bk	(Bj) minus (Bk) to Bi
F8	70ijK	SXi Aj $\pm$ K	(Aj) plus K to Xi
F9	71ijK	SXi Bj $\pm$ K	(Bj) plus K to Xi
FA	72ijK	SXi Xj+K	(Xi) plus K to Xi
FB	73ijk	SXi Xj+Bk	(Xj) plus (Bk) to Xi
FC	74ijk	SXi Aj+Bk	(Aj) plus (Bk) to Xi
FD	75ijk	SXi Aj-Bk	(Aj) minus (Bk) to Xi
FE	76ijk	SXi Bj+Bk	(Bj) plus (Bk) to Xi
FF	77ijk	SXi Bj-Bk	(Bj) minus (Bk) to Xi

SAi CM functions:
i=0 none
1 $\leq$ i $\leq$ 5 CM to Xi
6 $\leq$ i $\leq$ 7 Xi to CM

* only available in monitor mode

† only available in problem mode

table 9.1

9.3.1. Set B Instructions

The SB instructions (set B register instructions) are each implemented in one microcode word. The first operand is a source register specified by the Compass instruction j field; this value is on the QBUS. The second operand can come from the Compass instruction K field or from a source register specified by the Compass instruction k field; this value is on the PBUS. The values on the QBUS and PBUS pass through the ALU to the ZBUS. The value on the ZBUS is stored into a destination register specified by the Compass instruction i field (see the A Board on the block diagram).

The SB microcode words are at microcode addresses F0-F7 (see table 9.1). In Appendix 1, a macro (called SETB) is used in defining the SB instructions since most microcode fields are identical for all SB instructions.

The QT microcode field specifies whether the first operand is an A, B or X register. The PT field specifies if the second operand is an X or B register, or if it is the Compass instruction K field. The QRS and PRS microcode fields specify that the register numbers for the first and second operands come from the j and k field of the Compass instruction.

The PRS field is ignored if the PT field selects the K field from the Compass instruction.

The SELECT field selects add or subtract, and the AZBSA field sends the output from the ALU to the ZBUS. The AZD field specifies that a B register will be the destination, and the WRS field selects the Compass instruction i field for the destination register number.

The PCI field is 1 or 2, and determines how much to increment the Compass program counter. If the set B instruction uses the K field, PCI=2; otherwise, PCI=1. The hardware immediately starts fetching the CIW (Compass Instruction Word) for the next Compass instruction; the SB instructions do not use CM, so the CM is available for starting a CIW fetch.

The COND, JMP, and FALSE fields are used to check if the next Compass instruction is ready for immediate execution. The VALIDCM signal is specified with the COND field; it indicates if there is another instruction in the current CIW. If so, the next instruction is immediately executed; if not, the partially hidden CIW fetch is completed by executing an extra microcode word (see §2.3.5).

B0 is a constant 0, so an SB instruction which sets B0 has no effect.

9.3.2. Set X Instructions

The SX instructions (set X register instructions) are the same as the SB instructions except that their destinations are X registers. Since the set instructions perform 18-bit math, the hardware automatically sign extends the result to 60 bits (see §3.3.3 and §3.5).

9.3.3. Set A Instruction

The SA instructions are similar to the SB instructions except that they also perform CM operations. The SA instructions (set A register instructions) are each implemented in a sequence of 3 microcode words.

If the destination register is A1-A5, then the corresponding X register (X1-X5) is set to a value read from CM. The value in the A register is used as the CM address. If the destination register is A6 or A7, then the corresponding X register (X6 or X7) is written to CM. The value in the A register is used as the CM address.

Table 9.1 shows that the sequences of microcode words for the SA instructions begin at addresses E8 through EF. The first microcode word in the SA instructions is similar to the SB instructions, except that the value from the ALU is stored in an A register, and it is sent to the B Board (under control of the BZBSA field) and stored in the CMAC (under control of the BZD field). Use the block diagram to follow this path.

The second microcode word in the SA instruction is only executed if the instruction was the last instruction in the CIW (see §2.3.3). The conditional jump in the SETA macro using the VALIDCM signal goes directly to the third microcode word of the SA instruction unless a CIW fetch must be performed. The second microcode word completes the partially hidden CIW fetch started in the first microcode word (see §2.3.4).

The third microcode word is specified by a JPIFD1 JMP field value for the JP field in the first and second microcode word (see §2.1.7(1)). The address of the third microcode word is B8 hex plus the Compass instruction i field value:

- B8 The microcode word at B8 hex is executed if A0 is the destination register. Setting A0 does not cause a CM operation so this microcode word goes on to the next Compass instruction.
- B9-BD One of the microcode words at B9-BD hex is executed if the destination register is between A1 and A5. These microcode words are identical, so an echo is used to generate the 5 instructions in Appendix 1. The BZBSA field specifies that the B Board ZBUS value come from the CM Board and the SCMAAD field specifies that the CM address comes from the CMAC (set in the first microcode word). The AZBSA field specifies that the A Board ZBUS value come from the B Board and the AZD field specifies that an X register be written from the A Board ZBUS. Use the block diagram to follow the path of the data from the CM Board through the B Board and to the register on the A Board.
- BE-BF One of the microcode words at BE or BF hex (see Appendix 1) is executed if the destination register is A6 or A7. The SELECT field

specifies that the X6 or X7 register value passes through the ALU unchanged. The AZBSA field specifies that the A Board ZBUS value is from the ALU; the BZBSA field specifies that the B Board ZBUS value comes from the A Board ZBUS; the BZD field specifies that the B Board ZBUS value is sent to CM; and the SCMAAD specifies that the CM address come from the CMAC (set in the first microcode word). Use the block diagram to follow the path of the data from the A Board X register through the ALU and through the B Board to the CM.

9.4. 60-bit Integer and Boolean Instructions

The IX and BX instructions perform 60-bit integer operations. The IX instructions perform adds and subtracts and the BX instructions perform logical AND, logical OR and logical XOR (see figure 1.9c). Each of these instructions is implemented using one microcode word. In Appendix 1 the ALU60 macro is used in defining these microcode words.

9.5. Shift and Mask Instructions

The LX, AX and MX instructions are handled by the 60-bit BARREL SHIFTER rather than by the ALU (see A Board on the block diagram). Each of these instructions is implemented in one microcode word and is defined in the SHIFT macro in Appendix 1.

The LX instruction is used for circular left shifts and the AX instruction is used for arithmetic (sign-extended) right shifts. The MX instruction is used to generate a series of ones in the most significant bits of an X register (see §3.4.1).

The MX instruction and one form of the LX and AX instruction have an unusual use of the Compass instruction j and k fields. The j and k fields do not specify registers. Instead, these bits specify a count for the BARREL SHIFTER. Details of the function of these instructions may be found in §3.4.2.

9.6. Jump Instruction

The JP instruction is an unconditional jump which is implemented in 2 microcode words. The first microcode word computes the address to jump to by adding a B register to the K field. This address is sent to the Compass PC on the B Board under the control of the SPC field (see block diagram). The parcel bits in the Compass PC are loaded with zeroes so that execution begins at the first parcel of the CIW (Compass Instruction Word) (see §2.4).

The second microcode word for a JP instruction is used to fetch the CIW at the address computed in the first microcode word.

9.7. Conditional Jump Instructions

The conditional jump instructions are implemented in 2 microcode words. During the first microcode word a test is performed in the A Board. This test determines how the Compass PC is loaded on the B Board. The second microcode word is used to fetch the CIW.

The ZR, NZ, PL, MI, IR, OR, DF and ID instructions test the value of X registers. The i field of the instruction is used to tell the X REGISTER TEST hardware which test is to be performed (see the A Board on the block diagram). The SPC field specifies that the PC is loaded depending on the result of the X register test. If the test result is true, the K field is available from the ZBUS MUX (see the B Board on the block diagram).

The EQ, NE, GE and LT instructions are used to compare 2 B registers. The ALU subtracts the 2 B registers and the B REGISTER TEST hardware evaluates the result (see the A Board on the block diagram). The Compass instruction number is used to specify which test is to be performed by the B REGISTER TEST hardware. The SPC field specifies that the PC is loaded depending on the result of the B register test.

9.8. Return Jump Instruction

The RJ instruction (return jump instruction) is used to call subroutines. Two operations are performed: A JP instruction to the current Compass program counter plus one is stored at the location specified by the K field, and the Compass program counter is set to the K field plus one (see arrow labeled A in figure 9.8). The subroutine ends with a jump to its first word (see arrow B); this contains the JP instruction which was stored during the RJ instruction, so a JP back to the Compass instruction following the RJ is performed (see arrow C).

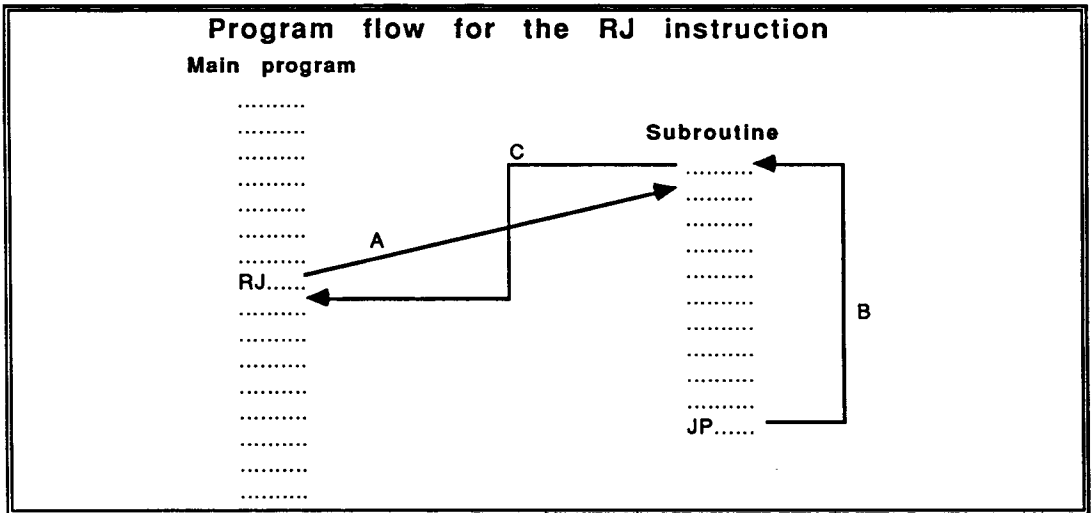


figure 9.8

A sequence of three microcode words is used to perform the RJ instruction. The first microcode word increments the Compass PC and loads the RJ instruction's K field into the CMAC. The second microcode word stores the JP instruction at the CMAC; this instruction is assembled by the B Board's ZBUS MUX using the incremented PC value. At the end of the second microcode word the CMAC is incremented using the INCCMBM value for the ENABLE field. (The INCCMBM value performs other functions in addition to incrementing the CMAC; these are used in BM instructions.) The third microcode word loads the PC from the CMAC and fetches the CIW at the new PC value.

9.9. Exchange Jump Instruction

The XJ instruction (exchange jump instruction) activates exchange packages. Exchange packages specify a Compass program counter value, restrictions on memory usage, restrictions on executing out of IOCM, restrictions on interrupting peripheral port microcomputers, and a monitor mode flag. Exchange packages are described in §1.7.

The XJ instruction is used to call operating system functions from problem mode, and it is used to start problem mode programs from monitor mode.

The XJ instruction operates differently in problem mode than in monitor mode. In problem mode, the current exchange package is stored at a fixed location in memory, and then a pre-set exchange package is activated. In monitor mode, the current exchange package is not stored, and the exchange package which is activated immediately follows the XJ instruction: it is not a pre-set exchange package.

Figure 9.9a shows the program flow for the XJ instruction. In this example, a problem mode XJ instruction is being used to call an operating system function. The operating system function executes in monitor mode; it returns to the problem mode program after it completes. Arrow A in figure 9.9a is the problem mode XJ jumping to the monitor mode program; the address of the monitor mode program is stored in a pre-set exchange package, stored at memory location XV.PMODE plus 3. Before the problem mode XJ instruction performs the jump indicated by arrow A, it stores the current exchange package, complete with the incremented PC value, at memory location XV.PMODE. When the operating system function completes, it jumps to XV.PMODE (see arrow B); this is the start of the exchange package stored during arrow A. The exchange package at XV.PMODE starts with 0130, which is an XJ instruction (see §1.7 and table 9.1); this XJ instruction, executed in monitor mode, results in a return to the problem mode program (see arrow C).

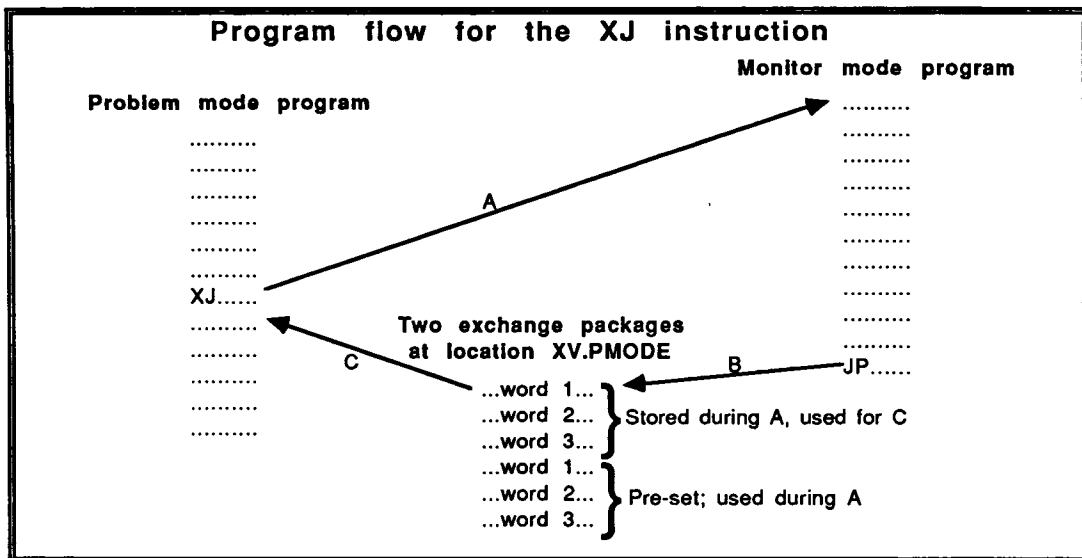


figure 9.9a

In problem mode, XJ instructions are used to suspend the program and call operating system functions. To suspend the program, the A, B and X registers must be saved so that the suspended program can be restarted later. The XJ instruction automatically saves registers A6 and A7 in T4 and T5 when entering monitor mode. It also automatically switches to the T register set; this means that the operating system program will use the T registers when it references an X register (see §3.1.2). This is done to retain the values in the X registers. The XJ instruction automatically restores registers A6 and A7 from T4 and T5 when leaving monitor mode, and switches back to the X register set. Registers A0-A5 and B1-B7 are not automatically saved by XJ; this must be done by the operating system.

When an XJ instruction is executed in monitor mode, the instruction is 3 60-bit words long, since it includes the exchange package. In problem mode, the XJ instruction is one 60-bit word long; only the instruction number from that word is used. Figure 9.9b outlines the sequence of microcode words which implement XJ.

Microcode sequence for the XJ instruction	
	1) Set temporary monitor mode; if in monitor mode go to XJMON, else go to XJPROB
XJPROB	2) MAR = XV.PMODE; advance PC to next CIW
INTRAP2	3) CMAC = MAR; select T registers
	4) Write exchange package word 1 to CM at CMAC; increment CMAC
	5) Write exchange package word 2 to CM at CMAC; increment CMAC; T4 = A6
	6) Write exchange package word 3 to CM at CMAC; increment CMAC; T5 = A7; go to XJCONT
XJMON	7) CMAC = PC
XJCONT	8) Load exchange package word 1 from CM at CMAC; increment CMAC
	9) Load exchange package word 2 from CM at CMAC; increment CMAC; A6 = T4
	10) Load exchange package word 3 from CM at CMAC; A7 = T5; if in monitor mode go to FETCH
	11) Select X registers; go to FETCH

figure 9.9b

In problem mode the current exchange package is stored to a fixed location in memory, and a pre-set exchange package is loaded from memory. In the first microcode word temporary monitor mode is set by the ENABLE field (bits 17-14 of the microcode word). This allows the exchange jump microcode access to all memory without actually entering monitor mode. Temporary monitor mode is automatically cleared when the next CIW is loaded.

Microcode words 2-6 are executed only in problem mode. These store the current exchange package at a fixed location in memory called XV.PMODE. The current exchange package is available from the ZBUS MUX (see the B Board on the block diagram). Microcode words 2-6 also save A6 and A7 in T4 and T5, and select the T register set.

Microcode word 7 is executed only in monitor mode. The CMAC (CM Address Counter) is set to the Compass PC; this is the address of the exchange package.

Microcode words 8-10 are used to load the new exchange package. They are executed in both monitor and problem mode. These instructions also copy A6 and A7 from T4 and T5; in monitor mode this restores A6 and A7.

Microcode word 11 is executed only if the newly loaded exchange package specified problem mode. This microcode word selects the X register set.

FETCH is a microcode word which reads the next CIW from CM.

9.10. Interrupts

Interrupts may be generated by the PP microcomputers (Peripheral Port microcomputers), by the TIMER, by P register breakpoints, by single-bit and double-bit memory errors, and by attempting to access memory outside of the ranges specified in the exchange package (see §1.7). Interrupts are ignored in monitor mode; they are processed whenever a new Compass instruction is started in problem mode.

Each type of interrupt has a pre-set exchange package at a fixed location in memory. When an interrupt occurs the equivalent of a problem mode exchange jump is executed using the pre-set exchange package. When the interrupt has been processed, a monitor mode exchange jump is used to continue the interrupted program (see §9.9).

The sequence of microcode words to process an interrupt begins at microcode memory location 80 hex plus the interrupt number (see §2.1.7).

The TRAP macro is used to define interrupts in Appendix 1; the argument of the TRAP macro is a fixed memory address for the exchange package. The TRAP macro loads this address into the MAR (Memory Address Register), and goes to INTRAP2 (see figure 9.9b).

9.10.1. Single-Bit and Double-Bit Error Interrupts

When single-bit and double-bit memory errors occur, information is stored in the B Board's ERROR LATCH (see §2.9.4). A special sequence of microcode words is executed when a single-bit error interrupt or a double-bit error interrupt occurs.

The sequence of microcode words for single-bit errors increments a counter of errors at memory location IO.SBE. If the counter was zero, additional operations are performed: the BMAR/SYN word (see the B Board's ZBUS MUX on the block diagram) is stored at IO.SBE+1, the ERROR ADDR/K word is stored at IO.SBE+2, the CMRAFL word is stored at IO.SBE+3, and the exchange package at XV.SBE is activated using the TRAP macro (see §9.9). The Compass program specified by XV.SBE may read the words of information stored at IO.SBE. The Compass program specified by XV.SBE stores zero

into the counter at IO.SBE; if it does not store zero into the counter, the XV.SBE exchange package will not be activated again until the operating system stores zero into the counter.

The sequence of microcode words for double-bit errors is similar to the sequence for single-bit errors, except that it uses memory location IO.DBE to store information, and it always activates the exchange package at XV.DBE, regardless of the value of the counter at IO.DBE.

9.11. Traps

Traps are taken when the hardware cannot handle a floating point operation (see §4.1.3). When a trap is taken, the equivalent of a problem mode XJ instruction is executed using a pre-set exchange package. This is implemented using the TRAP macro in Appendix 1; this is the same macro used in interrupts (see §9.10). The OP and the ijk fields in the exchange package are provided specifically for floating point traps; these tell the operating system which floating point instruction could not be handled by the hardware and must be simulated by Compass software.

Traps can only be taken in problem mode, so operands which cause traps in multiplies and normalizes are illegal in monitor mode. If such a trap is attempted, the microcode will hang at microcode word SOSEXEC (see Appendix 1).

9.12. Bulk Memory Instructions

The RE instruction transfers words from BM to CM. The WE instruction transfers words from CM to BM. The number of words to transfer is the K field plus the B register specified by the j field. The starting address in CM is specified by register A0, and the starting address in BM is specified by X0. In WE instructions, bit 29 of X0 is used to specify an interlocked write operation (see §2.7.9).

The RE and WE instructions are 30 bits long, and must occupy the first two parcels of a CIW (Compass Instruction Word). If an interlocked write operation fails, execution resumes in the third parcel of the CIW; in all other cases, execution resumes in the first parcel of the next CIW.

BM is organized in groups of 8 words called BM records. Data is transferred one or two words at a time (single- or double-word transfers) between CM and the Bulk Memory Data Boards' SCRATCH PAD LATCHES; data is transferred one record at a time between the SCRATCH PAD LATCHES and the Bulk Memory dynamic RAMs (see §7.4). During a block transfer, a special process must be performed by the microcode at each record boundary. The B Board's BM ADDRESS hardware is used by the RE and WE microcode

to generate addresses and BM control signals (see §2.7). The least significant 3 bits of the 27-bit BM address are held in the BMWP; the other 24 bits are held in the BMAR. The length of the transfer is held in the BMLR.

The microcode controls BM by specifying BMSTUFF for the ENABLE field (bits 17-14 in the microcode word), and using the SELECT field (bits 63-60 of the microcode word). Each of the four bits in the SELECT field is used to control one of four BM functions: (1) latch the scratch pad; (2) start a dynamic RAM memory cycle; (3) increment the BMAR; (4) add 1 (for single-word transfers) or 2 (for double-word transfers) to the CMAC and the BMWP, and subtract 1 or 2 from the BMLR. The FFPU field controls the fourth function: it may individually inhibit the incrementing or decrementing of any of the three counters.

The BMWRITE field (bit 07 of the microcode word) is used to specify a read from BM or a write to BM. In Appendix 1, the BMREAD and the BMWRITE macros are used to set this microcode field.

Figure 9.12a outlines the sequence of microcode words which implement the RE instruction. Microcode words 1-5 initialize the BMLR (Bulk Memory Length Register), the BMAR (Bulk Memory Address Register), and the CMAC (Central Memory Address Counter); a check is made for zero-length transfers; the PC is incremented by 2 parcels (this is done again in microcode word 20); the BM is requested; and a dynamic RAM read cycle of the first BM record is initiated.

Microcode words 6-11 are the slow BM record read loop; microcode words 14-18 are the fast BM record read loop. Each of these loops transfers one record from the BM's SCRATCH PAD LATCHES to CM, one or two words at a time. The fast transfer loop is used whenever possible; the B Board's BM ADDRESS circuit generates the FASTRD signal (called FASTREAD in Appendix 1) to tell the microcode when conditions allow the use of the fast transfer loop (see §2.7.11). For example, the fast loop cannot be used if the BMLR is less than 8, since the fast transfer loop assumes a complete record is to be transferred. In the fast loop, transfers from the SCRATCH PAD LATCHES to CM are pipelined: they are started in one microcode word, and completed in the next. In both loops, a dynamic RAM read cycle of the next BM record is done simultaneously with transferring the current BM record from the SCRATCH PAD LATCHES to CM (see Bulk Memory Data Boards on the block diagram, and §7.4). A dynamic RAM read cycle can complete more quickly than the time it takes to execute the fast loop; the slow loop needs checks to make sure the dynamic RAM read cycle is complete, in case the slow loop is transferring only one word in the BM record.

Microcode sequence for the RE instruction

INITIALIZATIONS

- 1) $BMLR = (Bj) + K$
- 2) $BMAR$ and $BMWP = (X0)$
- 3) $CMAC = (A0)$; $PC = PC + 2$; if $BMLR=0$ go to OKGO
- 4) Request BM; wait until BM acknowledges
- 5) Initiate a BM dynamic RAM read cycle

SLOW BM RECORD READ LOOP

- | | |
|-------|---|
| RLOOP | 6) Wait until BM dynamic RAM read cycle completes |
| | 7) Latch record from BM dynamic RAMs into SCRATCH PAD LATCHES;
if a fast record transfer is possible, go to FASTREAD |

- | | |
|----------|--|
| FASTEXIT | 8) Increment $BMAR$; if $BMLR=0$ go to RDEXIT |
| | 9) Initiate a BM dynamic RAM read cycle |
| | 10) Transfer one or two words from the SCRATCH PAD LATCHES to CM;
increment or double increment $CMAC$ and $BMWP$;
decrement or double decrement $BMLR$;
repeat this until end of record or $BMLR=0$ or an error has occurred |

- 11) If end of record go to RLOOP

- | | |
|--------|--|
| RDEXIT | 12) Wait until BM dynamic RAM read cycle completes |
| | 13) Latch record from dynamic RAMs into SCRATCH PAD LATCHES; go to ENDBM |

FAST BM RECORD READ LOOP

- | | |
|----------|---|
| FASTREAD | 14) Read two words from the SCRATCH PAD LATCHES (to be stored in 15);
double increment $BMWP$; double decrement $BMLR$;
increment $BMAR$; if an error has occurred go to RDEXIT |
| | 15) Store two words in CM; read two new words from SCRATCH PAD LATCHES;
double increment $CMAC$ and $BMWP$; double decrement $BMLR$;
initiate a BM dynamic RAM read cycle; if an error has occurred go to RDEXIT |
| | 16) Store two words in CM; read two new words from SCRATCH PAD LATCHES;
double increment $CMAC$ and $BMWP$; double decrement $BMLR$; if error, RDEXIT |
| | 17) Store two words in CM; read two new words from SCRATCH PAD LATCHES;
double increment $CMAC$ and $BMWP$; double decrement $BMLR$;
latch record from BM dynamic RAMs into SCRATCH PAD LATCHES;
if an error has occurred go to RDEXIT |
| | 18) Store two words in CM; double increment $CMAC$;
if a fast record transfer is possible, go to FASTREAD, else go to FASTEXIT. |

END OF RE INSTRUCTION

- | | |
|-------|--|
| ENDBM | 19) Release the BM |
| OKGO | 20) $PC = PC + 2$; load the CIW LATCH;
go to next Compass instruction unless the $BMFL$ was exceeded |
| | 21) Trap to operating system for $BMFL$ error |

figure 9.12a

Microcode words 12 and 13 ensure that the last dynamic RAM read cycle has completed; data from this last (superfluous) read cycle is latched into the SCRATCH PAD LATCHES, but is not transferred to CM (see §8.8.1 for details). Microcode words 19 and 20 release the BM so that other CPUs may use it, and execution is continued in the first parcel of the next CIW. Microcode word 21 traps to the operating system if the BMFL was exceeded (see §1.7); other error conditions, such as single-bit or double-bit memory errors, or CMFL exceeded are handled by interrupts which will occur during microcode word 20's NEXTOP jump (see §2.1.7 for interrupt details). If a BM read breakpoint (register P4) or a CM write breakpoint (register P2) occurred during the RE instruction, the instruction will complete and a breakpoint interrupt will occur on microcode word 20 (see §9.23 for breakpoint operation).

Figure 9.12b outlines the sequence of microcode words which implement the WE instruction. Microcode words 1-5 initialize the BMLR, the BMAR, and the CMAC; a check is made for zero-length transfers; the PC is incremented by 2 parcels (this is done again in microcode word 20); the BM is requested; and a check is made if bit 29 was set in BM address, indicating an interlocked write.

Microcode words 6-11 are the BM record write loop. This loop transfers one BM record from CM to the BM's SCRATCH PAD LATCHES, one or two words at a time. Microcode word 6 is only executed for the first transfer (one or two words) of the first record; microcode word 7 is used for the first transfer in all subsequent records. Microcode word 7 performs a dynamic RAM write cycle to write the previous record from the SCRATCH PAD LATCHES to the dynamic RAMs; once the write cycle completes, the new data (one or two words) being transferred from CM is loaded into the SCRATCH PAD LATCHES. Microcode word 8 completes the transfers for each record. Microcode word 9 checks if there are any errors or if the transfer has completed; if not, it goes to word 7 to transfer the next record. Microcode word 10 aborts if there was an error; the error may be a double-bit memory error, or the exchange package specifications may have been exceeded (see §1.7). Microcode word 11 performs a dynamic RAM write cycle to write the last record from the SCRATCH PAD LATCHES to the dynamic RAMs.

Microcode words 7 and 11 may write the complete record from the SCRATCH PAD LATCHES to the BM dynamic RAMs, or they may write a partial record. The BMC Board's ACTIVE SCRATCH PAD FLIP-FLOPS keep track of which words in the SCRATCH PAD LATCHES received data from CM; the words which did not receive data from CM are not written to the BM dynamic RAMs (see §8.7).

Microcode sequence for the WE instruction

INITIALIZATIONS

- 1) $BMLR = (Bj) + K$
- 2) $BMAR$ and $BMWP = (X0)$
- 3) $CMAC = (A0)$; $PC = PC + 2$; if $BMLR=0$ go to OKGO
- 4) Request BM; wait until BM acknowledges
- 5) If interlocked write was requested, go to INTLOK

BM RECORD WRITE LOOP

- | | |
|-----------------------|---|
| WRITE | <ol style="list-style-type: none"> 6) Transfer one or two words from CM to SCRATCH PAD LATCHES; increment or double increment CMAC and BMWP; decrement or double decrement BMLR; if an error has occurred go to ERRCHK else go to WRITE2 7) Initiate and complete a BM dynamic RAM write cycle; transfer one or two words from CM to SCRATCH PAD LATCHES; increment or double increment CMAC and BMWP; decrement or double decrement BMLR; if $BMLR=0$ or an error has occurred go to ERRCHK |
| WRITE2 | <ol style="list-style-type: none"> 8) Transfer one or two words from CM to SCRATCH PAD LATCHES; increment or double increment CMAC and BMWP; decrement or double decrement BMLR; repeat until end of record or $BMLR=0$ or an error has occurred |
| ERRCHK | <ol style="list-style-type: none"> 9) If $BMLR \neq 0$ and there was no error, go to WRITE 10) If an error occurred, go to ENDBM 11) Initiate and complete the final BM dynamic RAM write cycle; go to ENDBM |
| END OF WE INSTRUCTION | |
| ENDBM | <ol style="list-style-type: none"> 12) Release the BM |
| OKGO | <ol style="list-style-type: none"> 13) $PC = PC + 2$; load the CIW LATCH; go to next Compass instruction unless the BMFL was exceeded 14) Trap to operating system for BMFL error |
| INTERLOCK OPERATION | |
| INTLOK | <ol style="list-style-type: none"> 15) Initiate a BM dynamic RAM read cycle 16) Wait until BM dynamic RAM read cycle completes 17) Latch record from BM dynamic RAMs into SCRATCH PAD LATCHES; if bit 59=0 go to WRITE2 18) Release the BM; go to next instruction |

figure 9.12b

Microcode words 12-14 are shared between RE and WE (see ENDBM in figure 9.12a). Microcode word 12 releases the BM so that other CPUs may use it, and in microcode word 13 execution is continued in the first parcel of the next CIW. Microcode word 14 traps to the operating system if the BMFL was exceeded (see §1.7); other error conditions, such as single-bit or double-bit memory errors, or CMFL exceeded are handled by interrupts which will occur during microcode word 13's NEXTOP jump (see §2.1.7 for interrupt details). If a BM write breakpoint (register P5) or a CM read breakpoint (register P1) occurred during the WE instruction, the instruction will complete and a breakpoint interrupt will occur on microcode word 13 (see §9.23 for breakpoint operation).

Microcode words 14-18 are used for interlocked writes. The first word of the BM transfer is read; if it has bit 59 low (0), the WE instruction completes normally. If bit 59 is high (1), no write to BM is performed. Instead, execution resumes in the third parcel of the CIW (see §2.7.9).

9.13. Population Count Instruction

The CX instruction (population count instruction) counts the number of high (1) bits in a 60-bit word. The Xk register is read onto the A Board QBUS, and is sent to the D Board POPULATION COUNT hardware. The output of the POPULATION COUNT hardware goes to the D Board ZBUS, to the A Board ZBUS and then to the Xi register (see the A and D Boards on the block diagram). The FFPU field enables the POPULATION COUNT hardware on the D Board and the AZBSA field enables the D Board ZBUS onto the A Board ZBUS.

9.14. Pack Instruction

The PX instruction (pack instruction) is used to assemble a floating point number given the exponent and the mantissa value. The floating point format is explained in §1.10. The mantissa value (from Xk) is placed on the QBUS and the exponent value (from Bj) is placed on the PBUS. The values on the Q and P BUSES are sent to the D Board PACK/UNPACK hardware. The output of the PACK/UNPACK hardware is read onto the D Board ZBUS, sent to the A Board ZBUS, and stored in the Xi register (see the A and D Boards on the block diagram).

9.15. Unpack Instruction

The UX instruction (unpack instruction) is used to disassemble a floating point number to an exponent and a mantissa value. The unpack operation is implemented in a sequence of two microcode words. In the first microcode word the floating point value from Xk is read

onto the QBUS, and it is sent to the D Board PACK/UNPACK hardware; there the unpacked exponent is read onto the D Board ZBUS, and it is sent via the A Board ZBUS to the Bj register. In the second microcode word Xk is again read onto the QBUS, and it is sent to the D Board's PACK/UNPACK hardware; there the mantissa is sign extended and read onto the D Board ZBUS and it is sent via the A Board ZBUS to the Xi register (see the A and D Boards on the block diagram).

9.16. Normalize Instructions

The NX instruction (normalize) and the ZX instruction (rounding normalize) are used to normalize floating point numbers. They return both the normalized number and the amount the number had to be shifted during normalization. Normalization is described in §1.9.1. The ZX instruction inserts a round bit (the complement of the sign bit) to the right of the least significant bit of the floating point value before normalization.

Microcode sequence for the NX and ZX instructions

- | | |
|-----------|--|
| 1) | ADD/SUBTRACT/NORMALIZE MANTISSA LATCH B = (Xk mantissa);
EXPONENT LATCH B = (Xk exponent);
NORMALIZE SHIFT COUNT LATCH = normalize shift count;
specify rounding or non-rounding operation; PC = PC +1;
trap to a software simulation for special values of Xk |
| 2) | Bj = normalize shift count |
| 3) | Xi = mantissa shifted by normalize shift count |
| 4) | Xi = Xi shifted right by 1; if exponent underflowed go to EXPUND |
| 5) | Compute exponent; send packed result to Xi;
load the CIW LATCH; go to next instruction |
| EXPUND 6) | Xi = 0;
load the CIW LATCH; go to next instruction |

figure 9.16

The normalize instructions are implemented in a sequence of 6 microcode words. Use the block diagram to trace the paths of the data during each of these microcode words. The first of these words is separate for the ZX and the NX instructions, but the other 5 words are shared between the two. The first microcode word indicates to the hardware whether a rounding (ZX) or a non-rounding (NX) normalize is to be performed. The Xk register is read onto the A Board QBUS and is sent to the A Board X REGISTER TEST hardware. The X REGISTER TEST hardware computes a normalize shift count; this is the amount that the floating point mantissa must be shifted left to be normalized. The normalize shift count is stored in the D Board's NORMALIZE SHIFT COUNT latch, and the value on the A Board QBUS is stored in the D Board's ADD/SUBTRACT/NORMALIZE MANTISSA LATCH B and the D Board's EXPONENT LATCH B. The PC is incremented by 1 parcel

so that the fetch of the CIW for the next instruction may begin (see §2.3.4). A trap is taken if the D Board's SPECIAL INPUT DETECT circuit detects an X_k value which cannot be handled by hardware (see §4.1.3). This trap can only be taken in problem mode; values of X_k which cause this trap are not legal in monitor mode.

In the second microcode word the normalize shift count is read from the D Board's NORMALIZE SHIFT COUNT LATCH, passes through the ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING circuit, is sent to the A Board QBUS, passes through the A Board ALU, and is sent to B_j .

In the third microcode word the sign extended mantissa from the D Board's ADD/SUBTRACT/NORMALIZE MANTISSA LATCH B is read onto the A Board PBUS, and the D Board's NORMALIZE SHIFT COUNT LATCH is read onto the A Board QBUS. A shift is performed in the A Board's BARREL SHIFTER and the result is sent to X_i .

In the fourth microcode word X_i is shifted right by one. This is done because the ADD/SUBTRACT/NORMALIZE MANTISSA LATCH B on the D Board previously shifted the mantissa left by 1 in order to insert a round bit. The D Board's ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING may indicate that the result of the normalize underflowed the minimum exponent value; if so, the microcode goes to microcode word 6.

In the fifth microcode word the value is read from the D Board's NORMALIZE SHIFT COUNT LATCH and from the D Board's EXPONENT LATCH B. These values are subtracted in the D Board's ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING and the result is sent to the D Board's PACK/UNPACK hardware. A pack is performed with the exponent value from the D Board and the mantissa value from the X_i register on the A Board. This special type of pack is specified by a combination of the FFPU field and the SELECT field. The result from the pack is sent to the X_i register on the A Board. The B Board's CIW LATCH is loaded with the CIW being read from memory (see microcode word 1); this may or may not be the same as the CIW already in the latch. The microcode proceeds to the next Compass instruction.

Microcode word 6 places underflow (all bits zero) into X_i (see §1.10.2). This microcode word is executed only if microcode word 4 detected an underflow of the exponent.

9.17. Multiply Instructions

When two floating point numbers are multiplied, the two exponents are added and the 48-bit mantissas are multiplied to give a 96-bit product. The result of an FX multiply instruction consists of the most significant 48 bits of the product packed with the exponent.

The RX multiply instruction is a rounded version of the FX multiply instruction. The result of a DX multiply instruction consists of the least significant 48 bits of the product packed with the exponent decreased by 48.

The FX, RX, and DX multiply instructions are normally used with floating point inputs. The FX and DX multiply instructions may also be used with integer inputs. These integers are restricted to 48 bits; for details, see §4.1.3.

Microcode sequence for the multiply instructions

- 1) MULTIPLIER ARRAY Y INPUT and EXPONENT LATCH B = (Xk);
specify rounding or non-rounding operation
- 2) MULTIPLIER ARRAY X INPUT and EXPONENT LATCH A = (Xj);
PC = PC + 1;
trap to software simulation for special values of Xj, Xk
- 3) Multiply the mantissas;
loop until all partial products are accumulated
- 4) Compute exponent; perform post-normalize; send packed result to Xi;
load the CIW LATCH;
trap if exponent underflowed or overflowed; go to next instruction

figure 9.17

Figure 9.17 shows an outline of the sequence of microcode words which implement the FX and RX multiply instructions. All but the first microcode word of this sequence is shared for these two instructions. In the first microcode word, the mantissa of Xk is loaded into the D Board's MULTIPLIER ARRAY, and the exponent of Xk is loaded into the D Board's EXPONENT LATCH B (see the D Board on the block diagram). Rounding or non-rounding operation is specified.

In the second microcode word, the mantissa of Xj is loaded into the D Board's MULTIPLIER ARRAY, and the exponent of Xj is loaded into the D Board's EXPONENT LATCH A. The PC is incremented by 1 parcel so that the fetch of the CIW for the next instruction may begin (see §2.3.4). A trap is taken if the D Board's SPECIAL INPUT DETECT circuit detects values which cannot be handled by hardware (see §4.1.3). This trap can only be taken in problem mode; values of Xj and Xk which cause this trap are not legal in monitor mode.

The third microcode word computes the mantissa. It is implemented using the MPY macro; it loops to itself until the D Board's PARTIAL PRODUCTS ACCUMULATOR has accumulated all the outputs of the MULTIPLIER ARRAY.

If both inputs were normalized and the output is not normalized, the final microcode word performs a one-bit post-normalize on the mantissa. It also computes the exponent in the D Board's MULTIPLY/DIVIDE EXPONENT PROCESSING hardware by adding the values

from EXPONENT LATCH A and EXPONENT LATCH B; 1 is subtracted from the resulting exponent if a one-bit post-normalize was performed on the mantissa. The most significant 48 bits of the mantissa are packed with the exponent, and sent to the Xi register on the A Board. If the exponent underflowed or overflowed, trap to a Compass routine to store underflow or overflow in Xi (see §1.10.2). The B Board's CIW LATCH is loaded with the CIW being read from memory (see microcode word 2); this may or may not be the same as the CIW already in the latch. The microcode proceeds to the next Compass instruction.

The sequence of microcode words for the DX multiply instruction is the same as the sequence for the FX and RX instructions, except for the last microcode word. The least significant 48 bits of the mantissa are packed with the exponent decremented by 48, and sent to the Xi register on the A Board.

9.18. Floating Add and Subtract Instructions

The FX add and subtract instructions perform floating point addition and subtraction. The RX add and subtract instructions perform rounding floating point addition and subtraction.

The T register set is used in the sequence of microcode words which implements floating add and subtract. Since T registers are also used by programs in monitor mode (see §3.1.2), it is not possible to perform floating adds or subtracts in monitor mode.

Figure 9.18 outlines the sequence of microcode words used to implement the floating add and subtract instructions. These instructions require multiple precision math to compute the end-around-carry for the mantissa; the A Board's ALU provides the Q plus P plus saved end-around-carry function for this purpose (see §3.4.1). The saved end-around-carry is cleared in the first microcode word; it is used and updated by microcode words which have SELECT fields specifying MPADD.

The first microcode word is separate for each of the four instructions; the rest of the microcode words are shared. The first microcode word loads Xk, specifies add or subtract (the value from Xk is complemented on the D Board for subtracts), and rounding or non-rounding operation. The second microcode word loads Xj, and selects the T register set. The PC is incremented by 1 parcel so that the fetch of the CIW for the next instruction may begin (see §2.3.4). A trap is taken to a Compass simulation program if either Xj or Xk has a value which cannot be handled in hardware (see §4.1.3 for details). The third microcode word shifts the mantissa of the operand with the smaller exponent right, so that it has the same significance as the mantissa with the larger exponent; in the fourth microcode word the two are added together.

Microcode sequence for FX and RX floating add and subtract

- 1) ADD/SUBTRACT/NORMALIZE MANTISSA LATCH B = (Xk mantissa);
EXPONENT LATCH B = (Xk exponent); clear the saved carry;
specify add or subtract; specify rounding or non-rounding operation
- 2) ADD/SUBTRACT/NORMALIZE MANTISSA LATCH A = (Xj mantissa);
EXPONENT LATCH A = (Xj exponent);
select T registers; PC = PC +1;
trap to software simulation for special values of Xj, Xk
- 3) T1 = right shift of mantissa of operand with smaller exponent
- 4) T1 = MPADD(mantissa of operand with larger exponent, T1)
- 5) T0 = shift of mantissa of operand with smaller exponent
- 6) T3 = mask to apply to T0
- 7) T2 = T0 masked with T3;
if operand with larger exponent is negative go to NEGMAN
- 8) T0 = complement of T3; go to FACONT
- NEGMAN 9) T0 = all bits 1
- FACONT 10) MPADD(T0, T2) (low significance add with saved carry)
- 11) T0 = all bits 1
- 12) T1 = MPADD(T0, T1); (add saved carry to high significance result)
if T1 did not overflow 48 bits go to NONORM
- 13) T1 = T1 shifted right by 1 (post-normalize)
- NONORM 14) T1 = T1 shifted right by 1
- 15) Select X registers
- 16) Xi = T1
- 17) Compute the exponent; pack exponent and Xi; send result to Xi;
load the CIW LATCH; go to next instruction

figure 9.18

Microcode words 5-12 are used to add in the end-around-carry. Multiply precision math is needed to compute the end-around-carry. In microcode words 5 and 6, the mantissa with the smaller exponent is shifted and masked, in preparation for the low significance add (see §4.9.3). The data bits are now in the most significant bits of register T2; the less significant bits of T2 are zero. In microcode words 7-9, the mantissa with the larger exponent is sign extended to the right, in preparation for the low significance add. The sign extension is limited to the number of data bits in T2; all bits less significant than the data bits in T2 are set to one, so that they propagate the saved end-around-carry without change. Microcode word 10 performs the low significance add. Microcode words 11 and 12 perform the high significance add; all bits 1 are added in microcode word 11 because the A Board's ALU performs adds on inverted operands (see adds in §3.4.1).

Microcode word 13 performs a right shift if the A Board indicates that the mantissa overflowed 48 bits. This post-normalize guarantees that the output is normalized if both inputs were normalized and of the same sign. Microcode word 14 performs an additional right shift; the D Board's ADD/SUBTRACT/NORMALIZE MANTISSA LATCHES shifted the operands left by 1 in order to insert a round bit (see §4.10). Microcode word 15 selects the X register set; this undoes the selection of T registers in microcode word 2. In word 17 the mantissa (which is in register T1 on the A Board) and the exponent (which is computed by the D Board's ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING hardware) are packed to give the final result. The B Board's CIW LATCH is loaded with the CIW being read from memory (see microcode word 2); this may or may not be the same as the CIW already in the latch. The microcode proceeds to the next Compass instruction.

9.19. Double Precision Floating Add and Subtract Instructions

The DX add and subtract instructions are used for double precision floating point addition and subtraction, and are not implemented in microcode. A trap is always taken to a Compass program which simulates these instructions.

9.20. Divide Instructions

When two floating point numbers are divided, the two exponents are subtracted and the 48-bit mantissas are divided. The RX and FX divide instructions perform rounding and non-rounding floating point division. No integer divide instruction is provided.

The T register set is used in the sequence of microcode words which implements divide. Since T registers are also used by programs in monitor mode (see §3.1.2), it is not possible to use the divide instructions in monitor mode.

The sequence of microcode words which implement divide has three stages: (1) X_j , X_k are loaded into the D Board, checks are made for values which cannot be handled in hardware, and X_j is normalized. (2) An approximate value for X_j/X_k is computed. (3) The approximate result is made exact and stored in X_i .

9.20.1. Stage 1 of divide

Figure 9.20.1 outlines the sequence of microcode words for stage 1 of divide. Microcode words 2 and 3 copy X_j and X_k to T0 and T1; microcode words 3 and 4 use the A Board's X REGISTER TEST circuit together with the D Board's SPECIAL INPUT DETECT circuit to check for X_j and X_k values which cannot be handled by hardware (see §4.1.3 for details). Microcode words 2 and 3 are not combined with microcode words 4 and 5; this is

not possible because the X REGISTER TEST circuit is needed to check for positive and negative zero mantissas, which it cannot do while X registers are copied to T registers (see §3.6).

Microcode words 6-9 normalize the numerator (X_j) for divide. Stage 2 of divide expects both the numerator and the denominator (X_k) to be normalized; if a Compass program executes a divide with an unnormalized denominator, microcode word 5 will trap to a Compass program which simulates divide (see §4.1.3).

If the numerator has an underflow exponent, or if the numerator mantissa is ± 0 , microcode words 10-11 ensure that the result of the divide is $+0$.

Microcode sequence for stage 1 of divide

- | | |
|--------|--|
| 1) | Initialize MULTIPLIER ARRAY control signals;
select T registers; infinite loop if in monitor mode;
T7 = 0 for non-rounding divide or 1 for rounding divide |
| 2) | T1 = (X_k) |
| 3) | T0 = (X_j) |
| 4) | Test T1 |
| 5) | Test T0; PC = PC + 1;
trap to software simulation for special values of X_j , X_k |
| 6-9) | Normalize T0; trap to software simulation if the exponent underflows |
| 10-11) | Result of divide is $+0$ if numerator is ± 0 or underflow |

figure 9.20.1

9.20.2. Stage 2 of divide

Figure 9.20.2 shows the algorithm used in stage 2 of divide to compute the approximate mantissa value. This form of Newton's Method is called the Anderson-Earle-Goldschmidt-Powers method [8].

An initial table look-up approximates $\lambda = 1/\beta$. λ is used to find initial values for δ and ρ . An iteration is performed on δ and ρ , where δ converges towards 2^{47} and ρ converges to the result α/β . ρ does not exactly converge to α/β ; stage 3 of the divide algorithm is used to correct the value of ρ .

Algorithm for stage 2 of divide

Approximate α/β by repetitive multiplication

α is the mantissa of the numerator

β is the mantissa of the denominator

$\lambda = F(\beta)$ = approximate $1/\beta$ (look-up ROM)

$\delta_1 = \beta * \lambda$; $\rho_1 = \alpha * \lambda$

$\delta_2 = \delta_1 * (2^{48} - \delta_1)$; $\rho_2 = \rho_1 * (2^{48} - \delta_1)$

$\rho_3 = \rho_2 * (2^{48} - \delta_2)$

Microcode sequence for stage 2 of divide

- 1) MULTIPLIER ARRAY X INPUT = DIVIDE ROM look-up with T1 (β)
- 2) MULTIPLIER ARRAY Y INPUT = (T1 mantissa) (α);
EXPONENT LATCH B = (T1 exponent)
- 3) Multiply ($\delta_1 = \beta * \lambda$); loop until all partial products are accumulated
- 4) MULTIPLIER ARRAY X INPUT = (T0 mantissa) (α);
EXPONENT LATCH A = (T0 exponent)
- 5) T3 = most significant 48 bits of the multiply result (δ_1)
- 6) Multiply ($\rho_1 = \alpha * \lambda$); loop until all partial products are accumulated
- 7) MULTIPLIER ARRAY X INPUT = $2^{48} - T3$ ($2^{48} - \delta_1$)
- 8) MULTIPLIER ARRAY Y INPUT = T3 (δ_1)
- 9) T3 = most significant 48 bits of the multiply result (ρ_1)
- 10) Multiply ($\delta_2 = \delta_1 * (2^{48} - \delta_1)$); loop until partial products are accumulated
- 11) MULTIPLIER ARRAY Y INPUT = T3 (ρ_1)
- 12) T3 = most significant 48 bits of the multiply result (δ_2)
- 13) Multiply ($\rho_2 = \rho_1 * (2^{48} - \delta_1)$); loop until partial products are accumulated
- 14) MULTIPLIER ARRAY X INPUT = $2^{48} - T3$ ($2^{48} - \delta_2$)
- 15) T3 = most significant 48 bits of the multiply result (ρ_2)
- 16) MULTIPLIER ARRAY Y INPUT = T3 (ρ_2)
- 17) Multiply ($\rho_3 = \rho_2 * (2^{48} - \delta_2)$); loop until partial products are accumulated
- 18) Compute exponent; T3 = multiply result (ρ_3) packed with exponent;
trap to software simulation if exponent underflowed or overflowed

figure 9.20.2

The outline of the sequence of microcode words for stage 2 of divide is given in figure 9.20.2. The divide mantissa result is at most 1 bit larger than the input mantissas, since both inputs are normalized. In this microcode sequence, the microcode words which perform multiplies shift the result left by one under the control of the SELECT field (bits 63-60 of the microcode word). This shift is required since the input to each multiply is shifted right to accommodate mantissa values which may be 1 bit larger than the inputs; because of the left shift, the resultant multiply outputs are shifted right by 1 instead of 2. (The value from the table look-up is shifted right by 2, causing p1 and $\delta 1$ as well as p2 and $\delta 2$ to be shifted right by 1.)

If the final approximation, p3, requires a shift for post-normalization, 1 is added to the exponent in the D Board's MULTIPLY/DIVIDE EXPONENT PROCESSING hardware during microcode word 18.

If the exponent underflowed or overflowed, a trap is taken to a Compass program which simulates divide.

9.20.3. Stage 3 of divide

The result from stage 2 will be smaller than the correct result of the divide. Stage 3 of divide corrects the result from stage 2. Figure 9.20.3 outlines the sequence of microcode words for stage 3 of divide.

Microcode word 3 increases the magnitude of the result. For details of this operation see the DIVFIX signal in §3.5.1. A trial multiply is performed with this increased result times the denominator of the divide. If the trial multiply result is larger than the numerator of the divide, the result has been increased too much, and the un-increased result saved in microcode word 2 is the correct non-rounding result. If the trial multiply is exactly equal to the numerator, then the increased result from microcode word 3 is the correct result. If the trial multiply is less than the numerator of the divide, then the loop including microcode words 2 and 3 is repeated.

Microcode words 10-12 are used to check the least significant 48 bits of the trial multiply for equality to zero, if the most significant 48 bits of the trial multiply matched the numerator of the divide. This is necessary to check if there was an exact match with the numerator, or if the trial multiply was just slightly larger than the numerator. The unpack in microcode word 11 is needed to remove the exponent value from T5.

Microcode word 15 is used in non-rounding divides, and microcode word 16 is used in rounding divides. At the writing of this manual, rounding divides always round up. The B Board's CIW LATCH is loaded with the CIW being read from memory (see microcode word 5 in stage 1); this may or may not be the same as the CIW already in the latch. The microcode proceeds to the next Compass instruction.

Microcode words 2 and 4 handle special intermediate values. Microcode word 2 disables multiply blanking to allow intermediate divide iteration results with underflow exponents (see §4.1.3). Microcode word 4 checks for and indefinite (0/0) exponent in the intermediate result; this occurs when the final result is 2^{47} (see §3.5.1).

Microcode sequence for stage 3 of divide

MAN	1)	MULTIPLIER ARRAY Y INPUT = T1
	2)	T4 = T3; disable multiply blanking
	3)	T3 = T3 increased
	4)	MULTIPLIER ARRAY X INPUT = T3; trap to software simulation if T3 has an indefinite (0/0) exponent
	5)	Multiply the mantissas; loop until all partial products are accumulated
	6)	T2 = most significant 48 bits of multiply result with exponent
	7)	T5 = T0-T2 (subtract from the numerator)
	8)	Test T5; if T5=0, go to ZERO
	9)	T0-T2; if sign bit is the same, go to MAN, else DONE
ZERO	10)	T5 = least significant 48 bits of multiply result with exponent
	11)	T5 = unpack mantissa(T5)
	12)	Test T5; go to DONE if T5=0
	13)	T4 = T3
DONE	14)	Enable multiply blanking; test T7; if T7=0, go to FDIVX, else RDIVX
FDIVX	15)	Xi = T4, select X registers; load the CIW LATCH; go to next instruction
RDIVX	16)	Xi = T3, select X registers; load the CIW LATCH; go to next instruction

figure 9.20.3

9.21. Program Stop Instruction

The PS instruction (program stop instruction) is used to stop a Compass program running in problem mode and exit to the operating system. The instruction performs a trap using the exchange package at memory location XV.STOP. In monitor mode, the instruction stops the Zephyr; the microcode loops to itself, and execution can only resume after the machine is reset.

In problem mode, any Compass instruction with the first 6 bits equal to 0 is interpreted as a PS instruction. In monitor mode, the first 9 bits must be 0; further decoding is done on the i field (see table 9.1 and Appendix 1).

9.2.2. Transfer X to T and T to X Instructions

The XTj Xk instruction is used to transfer a value from Xk to Tj. The XXj Tk instruction is used to transfer a value from Tk to Xj. The XXj Xj+Tk instruction adds Tk to Xj. The T registers are an alternate register set which are used in place of X registers in monitor mode (see §3.1.2, §9.9). The T registers are also used in the sequences of microcode words for certain Compass instructions (see §9.18, §9.20).

The XTj Xk and XXj Tk instructions are available only in monitor mode. In problem mode, they act as PS instructions (see §9.21).

The XXj Xj+Tk instruction is used in problem mode only.

9.2.3. Transfer X to P and P to X Instructions (Breakpoints)

The XPj Xk instruction is used to transfer a value from Xk to Pj. The XXj Pk instruction is used to transfer a value from Pj to Xk. The P registers contain absolute memory addresses; they are breakpoints for the Zephyr.

The most significant bit (bit 59) of registers P1-P5 is high (1) if the P register is to be active in monitor mode. Bit 58 is high if the P register is to be active in problem mode.

The least significant 18 bits in P1 are the CM read breakpoint address. If a Compass instruction reads this address, and if P1 is active, an interrupt will occur after the next Compass instruction is executed in problem mode (see §9.10). The P registers contain absolute memory addresses, so the Compass program's memory address is added to the CMRA before it is checked against the address in P1. The least significant 18 bits of P2 are the CM write breakpoint address. The least significant 18 bits of P3 are the CM execution breakpoint address. If a CIW fetch occurs from this address, and if P3 is active, an interrupt will occur before any further instructions are executed.

The least significant 27 bits of P4 are the BM read breakpoint address. If a RE instruction reads this address from BM, and if P4 is active, an interrupt will occur. The P registers contain absolute memory addresses, so the RE instruction's BM memory address is added to the BMRA before it is checked against the address in P4. The least significant 27 bits of P5 are the BM write breakpoint address. If a WE instruction writes this address in BM, and if P5 is active, an interrupt will occur.

P6 and P7 are not used. P0 is used once an interrupt occurs; when P0 is read, the least significant 3 bits specify which P register caused the interrupt; when P0 is written, the interrupt is cleared. P0 must be written in the breakpoint interrupt processing routine.

These instructions are available only in monitor mode. In problem mode, they act as PS instructions (see §9.21).

9.24. Transfer X to C and C to X Instructions

The XCj Xk instruction is used to transfer an 18-bit value from Xk to Cj. The XXj Ck instruction is used to transfer a sign extended 18-bit value from Ck to Xj. The C registers are rarely used by Compass programs; they are used as scratch registers and for debugging by the microcode.

At the writing of this manual, C0-C3 are being used to help track down problems after the system crashes. The use of these registers may change. C0 holds the Bj+K value for the most recent JP instruction. C1 holds the K field for the most recent RJ instruction. C2 holds the address of the most recent XJ instruction.

These instructions are available only in monitor mode. In problem mode, they act as PS instructions (see §9.21).

9.25. Add CM to X Instruction

The XXj Xj,K instruction is used to add a value from CM to an X register. The CM address is specified by the K field.

The equivalent operation of XXj Xj,K can be achieved by a two instruction sequence (SA followed by IX); the XXj Xj,K instruction has the advantage that it does not require the use of an A and a X register for reading CM.

This instruction is only used in monitor mode; this is because the sequence of microcode words used to implement this instruction uses register T0 (see §3.2.3).

APPENDIX 1. MICROCODE LISTING

This appendix is a listing of the microcode described in §9; this microcode implements Compass on the Zephyr.

Each word of microcode is specified by a list of field values, and starts with BEGBUF -1.

The WAIT field values have not been minimized in this microcode listing. Many of the microcode words have maximum WAIT field values of 15; others have smaller WAIT field values, but not minimal values. The long process of minimizing WAIT field values, and the accompanying process of finely timing hardware clock signals with wire loops, was not complete at the time this manual was printed.

The index may be used to help find Compass instructions and macros in this listing. The bold page numbers listed for each instruction in the index are page numbers in this listing.

This listing uses symbolic names for the field values; the definitions of the field values may be found in Appendix 2.

```
20008      xv.bmer equ 4000100
20010      xv.cmer equ 4000200
20020      io.sbe  equ 4000400
20024      io.dbe  equ 4000440

**          wait - macro to specify *wait* field.
*
*          this macro also terminates the microcode word
*          (i.e. it calls *endbuf*).
*
*          wait    n
*
*          entry   n = value for *wait*.
*

wait      macro n
wait      set      n
**** the following line is to set all waits to maximum
* wait    set      15
**** the following 4 lines increment all waits by 1
* ifeq    n,15,1
* wait    set      15
* ifne    n,15,1
* wait    set      n+1
endbuf
endm

**          halt - instruction that hangs, looping on itself.
*
```

```

halt      macro
begbuf -1
scmad     set    zeroaddr
true      set    *
jmp        set    jp
wait      15
endm

**          alu60 - execute 60-bit alu instructions.
*
*          alu60  type,weight
*
*          entry  type = select field.
*                  weight = wait field.
*
*          used in -ix- and -bx- instructions.
*

alu60     macro  type,weight
begbuf -1
qt        set    x60
qrs       set    jfld

pt        set    x60
prs       set    kfld

select    set    type
azbsa     set    zsalu
azd       set    x
wrs       set    ifld

pci       set    1
cond      set    validcm
jmp       set    jpnop
false     set    fetch
wait      weight
endm

**          trap - initiate a trap to compass.
*
*          trap   addr
*
*          entry  addr = fwa of xp-s for exchange.
*
*          exit   to *intrap*.
*

trap      macro  addr,pcinc
begbuf -1
addrs     set    addr
enable    set    enmmar
*
z         ifc    ne,/pcinc//
pci       set    pcinc
z         endif
*
true      set    intrap

```

```

        jmp      set      jp
        wait     15
        endm

**      mpy - perform a multiply.
*
*      mpy      sel=select,next=next
*
*      entry    select = value for *select*.
*               *select* = "double" to double result.
*               *select* = "single" for normal result.
*               next = (optional) next micro-code address.
*

        mpy      macroe sel,next,wt
        begbuf   -1
        ffp      set      mpy
        select    set      sel
        cond      set      muldone
*
        nxt      ifc      ne,*next**
        true      set      next
        nxt      else
        true      set      *+1
        nxt      endif
*
        false    set      *
        jmp      set      jp
        wait     wt
*          wait     15          temporary test
        endm

**      memchk - set bit to do memory error checking
*               during this instruction.
*

        memchk    macro
        memchk    set      0
        endm

**      start-up stuff.
*
*      set mar to address to start execution.

0          org      0
0  ffffffff        start    begbuf -1
                addr      set      400100o
                enable    set      enmmar

                true      set      *+1
                jmp      set      jp
0  0180103e03ffffd6ffc    wait     15

*      clear memory error flip-flops.
*      hang if it gets here in problem mode.

```

1	ffffffffffffffff	enable	begbuf -1 set clrbe
		cond	set mon
		false	set *
		true	set **+1
		jmp	set jp
1	0201fffe67fffffaffc	wait	15
		* load pc from mar. use t-registers.	
2	ffffffffffffffff	bzbsa	begbuf -1 set zaddr
		spc	set ldpc18
		select	set sett
		enable	set enxt
		true	set fetch
		jmp	set jp
2	a7ff3ffe03ff3ffce9fc	wait	15
	3	free	setequ *
a3		org	0a3x
	a3	equ	*
		** sosovl - fix up overflowed result in software.	
		*	
		used for multiply, divide, normalize.	
		*	
		the machine is halted if in monitor mode.	
		*	
a3	ffffffffffffffff	sosovl	begbuf -1
		addrs	set xv.fpv
		enable	set enmmar
		true	set probtrap
		jmp	set jp
a3	a50015be03ffffd6ffc	wait	15
		** sosexec - execute this instruction in software.	
		*	
		used primarily for special cases of floating point instructions.	
		*	
		the machine is halted if in monitor mode.	
		*	
a4	ffffffffffffffff	sosexec	begbuf -1
		addrs	set xv.op
		enable	set enmmar
		true	set probtrap
		jmp	set jp
a4	a500143e03ffffd6ffc	wait	15

```

a5  ffffffff            probtrap begbuf -1
                                cond  set  mon
                                true  set  *
                                false set  intrap
                                jmp    set  jp
a5  a5a6fffe67fffffeffc wait    15

                                **      intrap - common entry point for interrupts
                                *              and traps.
                                *
                                *      entry  mar = cm address for exchange jump.
                                *
                                *      exit   temporary monitor mode set.
                                *              to intrap2 in -xj- sequence.
                                *

a6  ffffffff            intrap  begbuf -1
                                enable set  setxjff

                                true  set  intrap2
                                jmp    set  jp
a6  3cfffffe03fffffeaffc wait    15

                                **      fetch - fetch next instruction word from cm.
                                *
                                *      this is used if no partial prefetch was possible
                                *      on the previous instruction; this will do a
                                *      complete fetch.  if a partial prefetch was
                                *      possible, an instruction similar to this is
                                *      executed to finish the prefetch, but that
                                *      instruction will have a shorter "wait" field.
                                *

a7  ffffffff            fetch   begbuf -1
                                memchk
                                enable set  enprc
                                jmp    set  jpnop
a7  ffffffff03fffffc2efc wait    15

                                **      interrupts enter here.
                                *

80                                org      80x
                                equ      *

80                                80      intbeg
81                                trap  xv.timer    0 - timer interrupt
82                                trap  xv.cmfl     1 - cm fl error
                                trap  xv.brkpt     2 - breakpoints

                                *
                                *      3 - double bit error.
                                *
                                *      load info package address, jump to memory
                                *      error processing.

83  ffffffff            begbuf -1
                                addrs  set  io.dbe
                                enable  set  enmmar

```

```

      true      set      memerr
* true      set      *
      jmp      set      jp
83 0380093e03fffffd6ffc      wait      15

      *
      *      4 - single bit error.
      *
      *      load info package address, jump to memory
      *      error processing.

84 ffffffff          begbuf -1
      addr      set      io.sbe
* addr      set      400400o
      enable    set      enmmar

      true      set      memerr
* true      set      *
      jmp      set      jp
84 0380083e03fffffd6ffc      wait      15

85 ffffffff          trap      xv.iom1      5 - interrupt from i/o a
86 ffffffff          trap      xv.iom2      6 - interrupt from i/o b

      2      intr      dup      2
      intr      halt
      89      nfree     endd
      equ      *

**      memerr - process memory errors.
*
*      entry (mar) = fwa of 4 word error buffer.
*
*      increment first word of buffer
*      if no previous unprocessed error (1st word was 0)
*      then write buffer with error info.
*      resume normal processing.

3      org      free

*      load cmac with (mar), select t registers.

3 ffffffff          memerr      begbuf -1

      enable    set      enxt
      select    set      sett

      bzbsa     set      zaddr
      bzd       set      ldcmac

      true      set      *+1
      jmp      set      jp
3 04ff3ffe01ff3bfceffc      wait      15

      *      t0 <-- first word of buffer.

4 ffffffff          begbuf -1

```

```

enable      set      setxjff

bzbsa       set      zscm
scmad        set      cmac
azbsa        set      zbenz
azd          set      x

wrs          set      mfld
wmfld        set      0

true        set      *+1
jmp          set      jp
wait        15

4  05fffffe03fecee28ffc

*           first word of buffer <-- t0 + 1
*           check for double it error; trap out if so.

5  ffffffff

begbuf      -1
qt          set      enone
pt          set      x60
prs         set      mfld
rmfld       set      0
select      set      add
azbsa       set      zsalu
bzbsa       set      zaenz
bzd         set      zdcn
scmad       set      cmac

enable      set      inccmbm

cond        set      dbe
true        set      dbetrap
false       set      *+1
jmp         set      jp
wait        15

5  0a068dce39f8e3lc4ffc

*           test (t0) = 0. (first error)
*           if so, trap out on error.
*           if not, exit.

6  ffffffff

begbuf      -1
qt          set      x60
qrs         set      mfld
rmfld       set      0

cond        set      azero
true        set      sbetrap
false       set      *+1
jmp         set      jp
wait        15

6  0b07f9felbfffflfeffc

*           change back to x registers

7  ffffffff

begbuf      -1
enable      set      enxtnxt
select      set      setx

```

		true	set	*+1
		jmp	set	jp
7	08ff0ffe03ffffffceffc		wait	15
		*	exit from temporary monitor mode	
8	ffffffffffffffff		begbuf	-1
		enable	set	clrxjff
		true	set	*+1
		jmp	set	jp
8	09ffffffe03ffffffeefc		wait	15
		*	clear interrupt, resume normal processing.	
9	ffffffffffffffff		begbuf	-1
		enable	set	clrbe
		jmp	set	jpnop
9	fffffffc03ffffffaffc		wait	15
		*	set mar to double bit error trap address	
a	ffffffffffffffff	dbetrp	begbuf	-1
		addr	set	xv.dbe
		enable	set	enmmar
		true	set	memtrap
		jmp	set	jp
a	0c00173e03fffffd6ffc		wait	15
		*	set mar to single bit error trap address	
b	ffffffffffffffff	sbetrp	begbuf	-1
		addr	set	xv.sbe
		enable	set	enmmar
		true	set	memtrap
		jmp	set	jp
b	0c0018be03fffffd6ffc		wait	15
		*	write error data for memory traps at (cmac).	
c		memtrap	setequ	*
		merr	echo	,src=(zbmara,zenk,zscmra)
			begbuf	-1
		bzbsa	set	src
		bzd	set	zdcma
		scmad	set	cmac
		enable	set	inccmbm
		true	set	*+1
		jmp	set	jp
			wait	15
		merr	endd	

```

*          clear interrupt, trap out.

f  ffffffff            begbuf -1
                                enable set   clrbe

                                true  set   intrap
* true      set   fetch          dont trap out -- z
                                jmp    set   jp
f  a6fffffe03ffffffaafc    wait   15
                                10    setequ *
                                free

b0                                org    0b0x
                                **      rj k          0100K
                                *
                                * the return jump sequence is as follows.
                                *
                                * 1) pc <-- pc+1; cmac <-- k.
                                * 1) store -eq- at cmac; cmac <-- cmac+1.
                                * 2) pc <-- cmac; jump nextop.
                                *

b0  ffffffff            begbuf -1
                                spc     set   bump

                                bzbsa   set   zenk
                                bzd      set   ldcmac

                                azbsa   set   zbenz          cl has most recent rj target
                                azd      set   c
                                wrs      set   mfld
                                wmfld    set   1

                                true     set   rj.cont
                                jmp      set   jp
b0  10fffffe01fe2be7edfc    wait     15

10                                org    free
10  ffffffff            rj.cont begbuf -1
                                bzbsa   set   zsrj
                                bzd      set   zdcn
                                scmad    set   cmac

                                enable   set   inccmbm

                                true      set   *+1
                                jmp       set   jp
10  11fffffe01ff83fc4ffc    wait     15

11  ffffffff            begbuf -1
                                memchk
                                bzbsa   set   zscmac          cmac on bits 0-17 of z-bus
                                spc      set   ldpc18
                                enable   set   enprc

                                jmp       set   jpnop
11  fffffffc03ff1ffc28fc    wait     15

```

```

12      free      setequ *

**      re bj+k          011jK
*
*      read (bj+k) words of extended memory to cm.
*      (a0) has the cm fwa.  (x0) has the em fwa.
*
*      this instruction must reside in the first parcel.
*
*
*      refer to the zephyr hardware manual or to
*      mark zvilius's thesis for a description of
*      the re microcode.
*
*
*      The select field is used as 4 control bits whenever the
*      the enable field is set to "bmstuff".  The four bits from
*      left to right are: stbmcy, incbmar, dincmbm, latasp.
*
*      stbmcy = start bulk memory cycle,
*      incbmar = increment bm address register,
*      dincmbm = double increment the cm address counter,
*               the bm word pointer, and the bm length register
*      latasp = latch the scratch pads.

bmread  macro
bmwrite  set      fromdram
         endm

*      bmlr <-- (bj)+k;
*      if there has been a parity error, deal with it
*      by doing a nextop jump back to this instruction.
*      (the jump will be interrupted due to the error.)

b1      org      0b1x
b1      ffffffff begbuf -1
         qt      set      b
         qrs     set      jfld
         pt      set      penkf

         select   set      add
         azbsa   set      zsalu
         bzbsa   set      zaenz
         bzd     set      zdbmlr

*      cond      set      parity
         true     set      re.cont
         false    set      nextop
         jmp      set      jp
b1      12b880be03f8ebffeffc      wait  15

*      bmac <-- (x0).

12      org      free
12      ffffffff re.cont begbuf -1

```

```

qt      set      x60
qrs     set      mfld
rmfld   set      0

select  set      xfra
azbsa   set      zsalu
bzbsa   set      zaenz
enable  set      ldbmar

true    set      *+1
jmp     set      jp
wait    15
12  13ff09fe03f8ef1f6ffc

*      cmac <-- (a0); prc <-- prc+2;
*      if bmlr=0, go to okgo.

13  ffffffff

begbuf  -1
qt      set      a
qrs     set      mfld
rmfld   set      0

select  set      xfra
azbsa   set      zsalu
bzbsa   set      zaenz
bzd     set      ldcmac

pci     set      2

cond    set      bmlrz
true    set      okgo
false   set      *+1

jmp     set      jp
wait    15
13  251403fe31f8eb1feffe

*      request the bulk memory controller.

14  ffffffff

begbuf  -1
bmread
enable  set      reqxfr

cond    set      bmrpy
true    set      *+1
false   set      *
jmp     set      jp
wait    3
14  1514fffe57ffffdaffc

*      initiate a read

15  ffffffff

begbuf  -1
bmread
enable  set      bmstuff
select  set      0111b      stbmcy

* addr  set      xv.bmer
* enable set      enmmar

```

```

15 16ff7ffe03ffffffcaffc      true      set      *+1
                               jmp      set      jp
                               wait     10

*                               wait for bulk-memory-ready.

16 ffffffff            rloop    begbuf -1
                               bmread

                               cond     set      bmrdr
                               true      set      *+1
                               false     set      *
                               jmp      set      jp
16 1716fffe43ffffffeffc      wait     6

*                               latch scratch pad

17 ffffffff            begbuf -1
                               bmread

                               select    set      1110b      latcp
                               enable     set      bmstuff

                               true      set      fastread
                               false     set      *+1
                               cond      set      fastrd
17 1f18effe07ffffffcaffc      jmp      set      jp
                               wait     5

*                               bmar = bmar+1

18 ffffffff            fastexit begbuf -1

                               bmread
                               select    set      1011b      incbmar
                               enable     set      bmstuff

                               cond      set      bmrdr
                               false     set      *+1
                               true      set      rdexit
18 1d19bffe6ffffffcaffc      jmp      set      jp
                               wait     3

*                               null step

19 ffffffff            begbuf -1
                               bmread

                               true      set      *+1
                               jmp      set      jp
19 1afffffe03ffffffeffc      wait     3

*                               initiate a read.

1a ffffffff            begbuf -1
                               bmread

                               enable     set      bmstuff
                               select     set      0111b      stbmcy

```

	true	set	*+1	
	jmp	set	jp	
1a 1bff7ffe03ffffc		wait	10	
	*	read words from scratch pad to cm.		
1b ffffffff		begbuf	-1	
		bmread		
	bzbsa	set	zsbm	
	bzd	set	zdc	
	scmad	set	cmac	
	enable	set	bmstuff	
	select	set	1101b	dincmbm
	cond	set	bmxf	
	true	set	*	
	false	set	*+1	
	jmp	set	jp	
1b 1blcdffe59ffd3fc8ffc		wait	15	
	*	if not(end of transfer or some error),		
	*	then go to rloop.		
1c ffffffff		begbuf	-1	
		bmread		
	cond	set	bmrdone	
	true	set	*+1	
	false	set	rloop	
	jmp	set	jp	
1c 1dl6fffe6fffffeffc		wait	3	
	*	wait for bulk-memory-ready.		
1d ffffffff	rdexit	begbuf	-1	
		bmread		
	cond	set	bmrdy	
	true	set	*+1	
	false	set	*	
	jmp	set	jp	
1d 1eldfffe43fffffeffc		wait	4	
	*	latch scratch pad (so read command goes away).		
1e ffffffff		begbuf	-1	
		bmread		
	enable	set	bmstuff	
	select	set	1110b	latasp
	true	set	endbm	
	jmp	set	jp	
1e 24ffeffe03ffffc		wait	5	

```

*          read words from scratch pad to cm;  fastread
*          bmar = bmar + 1

1f  ffffffff            fastread begbuf -1
                             bmread

                             bzbsa  set    zsbm
*  bzd    set    zdcn
ffpu     set    11010b      disable cmac increment, double word
scmad    set    cmac

enable   set    bmstuff
select  set    1001b      incbmar,dincmbm

cond     set    memoops
false    set    rdexit
true     set    *+1
jmp      set    jp
1f  201d9ffe53ffdfc8fe8  wait    3

*          read words from scratch pad to cm;  fastread
*          start bulk memory cycle

20  ffffffff            begbuf -1
                             bmread

                             bzbsa  set    zsbm
bzd      set    zdcn
scmad    set    cmac

enable   set    bmstuff
select  set    0101b      stbmcy,dincmbm
ffpu     set    11011b      double word

cond     set    memoops
false    set    rdexit
true     set    *+1
jmp      set    jp
20  211d5ffe51ffd3fc8fec  wait    3

*          read words from scratch pad to cm;  fastread

21  ffffffff            begbuf -1
                             bmread

                             bzbsa  set    zsbm
bzd      set    zdcn
scmad    set    cmac

enable   set    bmstuff
select  set    1101b      dincmbm
ffpu     set    11011b      double word

cond     set    memoops
false    set    rdexit
true     set    *+1

```

```

21  221ddffe51ffd3fc8fec      jmp      set      jp
                                wait     3

                                *
                                *      read words from scratch pad to cm; fastread

22  ffffffff                begbuf -1
                                bmread

                                bzbsa    set      zsbm
                                bzd       set      zdcn
                                scmad     set      cmac

                                enable    set      bmstuff
                                select    set      1101b      dincmbm
                                ffpu      set      11011b     double word

                                cond      set      memoops
                                false     set      rdexit
                                true      set      *+1
                                jmp       set      jp
                                wait      3

22  231ddffe51ffd3fc8fec      *      read words from scratch pad to cm; fastread
                                *      latch the scratch pads

23  ffffffff                begbuf -1
                                bmread

                                bzbsa    set      zsbm
                                bzd       set      zdcn

                                scmad     set      cmac

                                enable    set      bmstuff
                                select    set      1100b      dincmbm, latasp
                                ffpu      set      11001b     double word, disable bmar and bmlr

                                * cond      set      memoops
                                * true      set      *+1
                                * false     set      rdexit
                                cond      set      fastrd
                                true      set      fastread
                                false     set      fastexit
                                * true      set      endbm
                                * false     set      endbm
                                jmp       set      jp
                                wait      3

23  1f18cffe05ffd3fc8fe4      *      end bulk-memory transfer. these steps are
                                *      the same for bulk memory read and writes.
                                *
                                *      release bulk memory controller.

24  ffffffff                endbm      begbuf -1
                                enable    set      endxfr

```

```

cond      set      bmrpy
true      set      *
false     set      *+1
jmp       set      jp
wait      15

24  2425ffffe57fffffe2ffc

*          prc <-- prc+2.
*          else nextop.

25  ffffffff

okgo      begbuf -1
memchk
pci       set      2
enable    set      enprc

cond      set      bmaabort
jmp       set      jp
true      set      *+1
* true    set      fetch
false     set      fetch
wait      15

25  26a7ffffe7bfffffc2efe

26  ffffffff

trap      xv.emfl

27  free    setequ *

**         we      bj+k                      012jK
*
*
*         write (bj+k) words of extended memory from cm.
*         (a0) has the cm fwa.  (x0) has the em fwa.
*
*         this instruction must reside in the first parcel.
*         normally execution resumes in the first parcel of
*         the next instruction word.  in case of a failed
*         interlock write an exit to the 3rd parcel is
*         performed.
*
**         interlock operation.
*
*         if bit 29 of the em address in (x0) is set, an
*         interlock operation is performed.
*
*         the word in em is first read, and its sign bit
*         is checked.  if the sign bit is set, the write
*         is inhibited and the half exit is taken.  if the
*         sign bit is clear, the write is performed and
*         the normal exit is taken.
*
*         notes  it is up to the software to ensure that
*                 the data written includes a 1 in bit 59.
*
*         interlock operations may only be
*         performed with single word writes.
*

```

```

*          refer to the zephyr hardware manual or to
*          mark zvilius's thesis for a description of
*          the we microcode.
*
bmwrite macro
bmwrite set    fromcpu
endm

*          bmlr <-- (bj)+k;
*          if there has been a parity error, deal with it
*          by doing a nextop jump back to this instruction.
*          (the jump will be interrupted due to the error.)

b2          org    0b2x
b2  ffffffff begbuf -1
          qt      set    b
          qrs     set    jfld
          pt      set    penkf

          select  set    add
          azbsa   set    zsalu
          bzbsa   set    zaenz
          bzd     set    zdbmlr

* cond      set    parity
true        set    we.cont
false       set    nextop
jmp         set    jp
b2  27b880be03f8ebffeffc wait 15

*          bmac <-- (x0).

27          org    free
27  ffffffff begbuf -1
          we.cont
          qt      set    x60
          qrs     set    mfld
          rmfld   set    0

          select  set    xfra
          azbsa   set    zsalu
          bzbsa   set    zaenz
          enable  set    ldbmar

          true    set    *+1
          jmp     set    jp
27  28ff09fe03f8ef1f6ffc wait 15

*          cmac <-- (a0); prc <-- prc+2;
*          if bmlr=0, go to okgo.

28  ffffffff begbuf -1
          qt      set    a
          qrs     set    mfld
          rmfld   set    0

```

```

        select    set    xfra
        azbsa     set    zsalu
        bzbsa     set    zaenz
        bzd       set    ldcmac

        pci       set    2

        cond      set    bmlrz
        true      set    okgo
        false     set    *+1

        jmp       set    jp
28  252903fe31f8eb1feffe    wait    15

        *         request the bulk memory controller.

29  ffffffff       begbuf -1
                        bmwrite
        enable    set    reqxfr

        cond      set    bmrpy
        true      set    *+1
        false     set    *
        jmp       set    jp
29  2a29fffe57ffffdaff7c    wait    3
        *         if interlock operation, go to intlok.

2a  ffffffff       begbuf -1
                        bmread

        cond      set    b29
        true      set    *+1
        false     set    intlok
        jmp       set    jp
2a  2b31fffe4bfffffeffc    wait    3

        *         write words from cm to scratch pad.
        *         this write is for the first write
        *         of a transfer

2b  ffffffff       begbuf -1
                        bmwrite

        bzbsa     set    zscm
        scmad     set    cmac
        enable    set    bmstuff

        select    set    1100b        dincmbm,latasp

        cond      set    bmxfr
        true      set    write2
        false     set    errchk
        jmp       set    jp
2b  2d2ecffe5bffcffc8f7c    wait    13

        *         write words from cm to scratch pad.
        *         this is for the first of all other writes.

```

```

*          this initiates and completes a write cycle.

2c  ffffffff            write  begbuf -1
                                bmwrite

                                bzbsa  set    zscm
                                scmad  set    cmac

                                enable  set    bmstuff
                                select  set    0000b          stbmcy,incbmar,dincmbm,latasp

                                cond    set    bmxfr
                                true   set    write2
                                false  set    errchk
                                jmp     set    jp
2c  2d2e0ffe5bffcffc8f7c  wait    15

*          write words from cm to scratch pad.
*          this is for the rest of the writes

2d  ffffffff            write2 begbuf -1
                                bmwrite

                                bzbsa  set    zscm
                                scmad  set    cmac

                                enable  set    bmstuff
                                select  set    1100b          dincmbm,latasp

                                cond    set    bmxfr
                                true   set    *
                                false  set    errchk
                                jmp     set    jp
2d  2d2ecffe5bffcffc8f7c  wait    7

*          if there is no error and there are more words to
*          transfer, go to write loop.

2e  ffffffff            errchk begbuf -1
                                bmwrite

                                cond    set    bmwrlp
                                true   set    write
                                false  set    *+1
                                jmp     set    jp
2e  2c2ffffe73fffffef7c  wait    5

*          if some error has occurred, go to endbm.

2f  ffffffff            begbuf -1
                                bmwrite

                                cond    set    memoops
                                true   set    *+1
                                false  set    endbm
                                jmp     set    jp

```

```

2f 3024ffffe53fffffef7c          wait    3
                                   *
                                   start and complete a bm write cycle

30 ffffffffefefefefefef          begbuf -1
                                   bmwrite

                                   enable set    bmstuff
                                   select set    0111b          stbmcy

                                   true  set    endbm
                                   jmp    set    jp
30 24ff7ffe03ffffcaf7c          wait    15

                                   *
                                   interlock operation.
                                   *
                                   transfer word to z bus;
                                   *
                                   if not(bit 59 = 1), go to write.
                                   *
                                   initiate a read

31 ffffffffefefefefefef          intlok  begbuf -1
                                   bmread
                                   enable set    bmstuff
                                   select set    0111b          stbmcy

                                   true  set    *+1
                                   jmp    set    jp
31 32ff7ffe03ffffcaffc          wait    15

                                   *
                                   wait for bulk-memory-ready.

32 ffffffffefefefefefef          begbuf -1
                                   bmread

                                   cond  set    bmrdy
                                   true  set    *+1
                                   false set    *
                                   jmp    set    jp
32 3332ffffe43fffffeffc          wait    4

                                   *
                                   latch scratch pad;

33 ffffffffefefefefefef          begbuf -1
                                   bmread
                                   select set    1110b          latasp
                                   enable set    bmstuff

                                   true  set    *+1
                                   jmp    set    jp
33 34ffeffe03ffffcaffc          wait    15

34 ffffffffefefefefefef          begbuf -1
                                   bmread
                                   bzbsa set    zsbm

                                   cond  set    b59

```

```

true      set      write2
false     set      *+1
* true    set      *
34 2d35fffe47ffdfffeffc  jmp      set      jp
                        wait     15

*          release bulk memory controller; go to fetch.

35 ffffffff          begbuf -1
                        enable   set      endxfr

cond      set      bmrpy
true      set      *
false     set      fetch
35 35a7fffe57fffffe2ffc  jmp      set      jp
                        wait     15
36 free      setequ *

**          xj - exchange jump.          013...
*

*          instruction is assumed to be in the first parcel;
*          results are unpredictable otherwise.
*

**          exchange jump sequences.
*
*          *) set temporary monitor mode; if in monitor,
*          go to xj.mon, else go to xj.prob.
*
* xj.mon  1) cmac <-- pc.
* xj.cont 2) load pc, mon/ioa from memory; increment cmac.
*          3) load cm ra/fl from memory;
*          a6 <-- t4; increment cmac.
*          4) load bm ra/fl from memory;
*          a7 <-- t5; if monitor, go to fetch.
*          5) select x registers; go to fetch.
*
* xj.prob 6) mar <-- (monitor xj address); pc <-- pc+1.
*
*          note   traps and interrupts enter here.
*
* intrap2 7) cmac <-- mar; select t registers.
*          8) write -xj- word to memory; increment cmac.
*          9) write out cm ra/fl; t4 <-- a6; increment cmac.
*          10) write out bm ra/fl; t5 <-- a7;
*              increment cmac; go to xj.cont.
*
*          set temporary monitor mode; if in monitor,
*          go to xj.mon, else go to xj.prob.

b3          org      0b3x
b3 ffffffff          begbuf -1
                        enable   set      setxjff

bzbsa      set      zspc
azbsa      set      zbenz

```

	azd	set	c	
	wrs	set	mfld	
	wmfld	set	2	c2 has most recent xj location
	cond	set	mon	
	true	set	xj.mon	
	false	set	xj.prob	
	jmp	set	jp	
b3	363bffffe67fe6feaaffc	wait	15	
	*	cmac <-- pc.		
36		org	free	
36	ffffffffffffffff	xj.mon	begbuf -1	
		bzbsa	set zspc	
		bzd	set ldcmac	
		true	set *+1	
		jmp	set jp	
36	37fffffe01ff6bffeffc	wait	15	
	*	load pc, mon/ia from memory; increment cmac.		
37	ffffffffffffffff	xj.cont	begbuf -1	
		memchk		
		bzbsa	set zscm	
		scmad	set cmac	
		spc	set ldpc20	
		bzd	set ldmon	
		enable	set inccmbm	
		true	set *+1	
		jmp	set jp	
37	38fffffe01ffcffc4afc	wait	15	
	*	load cm ra/fl from memory;		
	*	a6 <-- t4; increment cmac.		
38	ffffffffffffffff	begbuf	-1	
		memchk		
		bzbsa	set zscm	
		scmad	set cmac	
		bzd	set ldcmra	
		enable	set inccmbm	
		qt	set x60	
		qrs	set mfld	
		rmfld	set 4	
		select	set xfra	
		azbsa	set zsalu	
		azd	set a	
		wrs	set mfld	

	wmfld	set	6
	true	set	*+1
	jmp	set	jp
38 39ff09fe03f8c1984efc		wait	15
	* load bm ra/fl from memory; clear temp mon mode;		
	* a7 <-- t5; if monitor, go to fetch.		
39 ffffffff	begbuf	-1	
	memchk		
	bzbsa	set	zscm
	scmad	set	cmac
	bzd	set	ldbmra
	enable	set	clrxjff
	qt	set	x60
	qrs	set	mfld
	rmfld	set	5
	select	set	xfra
	azbsa	set	zsalu
	azd	set	a
	wrs	set	mfld
	wmfld	set	7
	cond	set	mon
	true	set	fetch
	false	set	*+1
	jmp	set	jp
39 a73a09fe67f8c5becefc		wait	15
	* select x registers; go to fetch.		
3a ffffffff	begbuf	-1	
	select	set	setx
	enable	set	enxt
	true	set	fetch
	jmp	set	jp
3a a7ff0ffe03ffffceffc		wait	15
	* mar <-- (monitor xj address); pc <-- pc+1.		
3b ffffffff	xj.prob	begbuf	-1
	addrs	set	xv.pmode
	enable	set	enmmar
	spc	set	bump
	true	set	*+1
	jmp	set	jp
3b 3c0003be03ffffd6dfc		wait	15
	* cmac <-- mar; select t registers.		
3c ffffffff	intrap2	begbuf	-1

	bzbsa	set	zaddr
	bzd	set	ldcmac
	select	set	sett
	enable	set	enxt
	true	set	*+1
3c	3dff3ffe01ff3bfceffc	jmp	jp
		wait	15
	*	write -xj- word to memory; increment cmac.	
3d	ffffffffffffffff	begbuf -1	
	bzbsa	set	zsjmp
	bzd	set	zdcn
	scmad	set	cmac
	enable	set	inccmbm
	true	set	*+1
3d	3effffffe01ff93fc4ffc	jmp	jp
		wait	15
	*	write out cm ra/fl; t4 <-- a6; increment cmac.	
3e	ffffffffffffffff	begbuf -1	
	bzbsa	set	zscmra
	bzd	set	zdcn
	scmad	set	cmac
	qt	set	a
	qrs	set	mflld
	rmflld	set	6
	select	set	xfra
	azbsa	set	zsalu
	azd	set	x
	wrs	set	mflld
	wmflld	set	4
	enable	set	inccmbm
	true	set	*+1
3e	3fff03fe01f852d04ffc	jmp	jp
		wait	15
	*	write out bm ra/fl; t5 <-- a7;	
	*	increment cmac; go to xj.cont.	
3f	ffffffffffffffff	begbuf -1	
	bzbsa	set	zsbmra
	bzd	set	zdcn
	scmad	set	cmac
	qt	set	a
	qrs	set	mflld

	rmfld	set	7	
	select	set	xfra	
	azbsa	set	zsalu	
	azd	set	x	
	wrs	set	mfld	
	wmfld	set	5	
	enable	set	inccmbm	
	true	set	xj.cont	
	jmp	set	jp	
3f	37ff03fe01f842f44ffc	wait	15	
	40	free	setequ *	
b4		org	0b4x	
	**			014xx
	*			
b4	ffffffffffffffff	halt		
b5		org	0b5x	
	**			015xx
	*			
b5	ffffffffffffffff	halt		
b6		org	0b6x	
	**	xxj	xj+tk	016jk
	*			
	*	add tk to xj.		
	*			
b6	ffffffffffffffff	begbuf	-1	
	qt	set	x60	
	qrs	set	jfld	
	pt	set	x60	
	prs	set	kfld	
	enable	set	enxt	
	select	set	addtttox	
	azbsa	set	zsalu	
	azd	set	x	
	wrs	set	jfld	
	pci	set	1	
	cond	set	validcm	
	jmp	set	jpnop	
	false	set	fetch	
b6		wait	15	
b7		org	0b7x	
	**			017xx
	*			

```

b7  ffffffff            halt

**      cm accesses - second step for -sa- instructions.
*
*      in the first step, the appropriate a-register
*      was loaded, and an indexed jump based on the
*      i-field was performed to get here.  now the
*      appropriate memory access, based on the register
*      involved, must be performed.
*

b8                                org      0b8x

**      a0 - no memory access is performed.
*
*      this instruction is also used in cases where the
*      parcel register is already valid and simply a
*      *nextop* jump is required.
*
*      the bulk memory micro-codes jump here.
*

b8  ffffffff            nextop  begbuf -1
b8  ffffffff03ffffffc  jmp      set    jpnop
                                wait    15

**      a1 through a5 - load corresponding x register
*      from memory using address in cmac.
*

      read      echo      ,n=(1,2,3,4,5)
                        begbuf -1
                        memchk
bzbsa      set      zscm
scmad      set      cmac
azbsa      set      zbenz
azd        set      x

wrs        set      mfld
wmfld      set      n

jmp        set      jpnop
                        wait    15

      read      endd

**      a6 and a7 - write memory from corresponding x
*      register using address in cmac.
*

      write     echo      ,n=(6,7)
                        begbuf -1
qt          set      x60
qrs         set      mfld
rmfld       set      n

```

```

        select    set    xfra
        azbsa     set    zsalu
        bzbsa     set    zaenz
        bzd       set    zdcn
        scmad     set    cmac

        jmp       set    jpnop
        wait      15

        write     endd

c0                                     org    0c0x
        **        ps                                00xxx
        *
        *        in monitor mode, further decoding is achieved
        *        by and indexed jump on the i-field offset by a8h.
        *

c0  ffffffff          begbuf -1
        cond      set    mon
        false     set    ps.cont
        jmp       set    jpifd2
c0  ff40ffff67ffffeffc wait    15

        *        problem mode ps                                00xxx

40                                     org    free
40  ffffffff          ps.cont trap    xv.stop
        41          free     setequ *

        *        monitor mode ps                                000xx

a8                                     org    0a8x
a8  ffffffff          begbuf -1
        true       set    *
        jmp       set    jp
a8  a8fffffe03ffffeffc wait    15

        **        xcj xk                                001jk
        *
        *        load cj from xk.  monitor mode only.
        *

a9  ffffffff          begbuf -1
        qt         set    x60
        qrs        set    kfld

        select     set    xfra
        azbsa     set    zsalu
        azd        set    c
        wrs        set    jfld

        pci        set    1
        cond       set    validcm
        jmp        set    jpnop
        false      set    fetch
a9  ffa7097c0be8ffffeffd wait    15

```

```

**          xxj ck          002jk
*
*          load xj from ck.  monitor mode only.
*

aa  ffffffff

qt      set      zeros
pt      set      a          c registers on p bus

prs     set      kfld

select  set      or
azbsa   set      zsalu
azd     set      x
wrs     set      jfld

pci     set      1
cond    set      validcm
jmp     set      jpnop
false   set      fetch

aa  ffa72f980be8feffefd      wait  15

**          xpj xk          003jk
*
*          set breakpoint register.
*
*          load pj from xk.  monitor mode only.
*

ab  ffffffff

qt      set      x60
qrs     set      kfld

select  set      xfra
azbsa   set      zsalu
bzbsa   set      zaenz
bzd     set      zdbp

pci     set      1
cond    set      validcm
jmp     set      jpnop
false   set      fetch

ab  ffa7097c09f8e7ffeffd      wait  15

**          xxj pk          004jk
*
*          read breakpoint register.
*
*          load xj from pk.  monitor mode only.
*

ac  ffffffff

bzbsa   set      zsbp
azbsa   set      zbenz
azd     set      x
wrs     set      jfld

```

```

pci      set      1
cond     set      validcm
jmp      set      jpnop
false    set      fetch
ac ffa7fffc0beaeffefd      wait      15

**      xtj      xk      005jk
*
*      load tj from xk.
*

ad ffffffff          begbuf -1
qt      set      x60
qrs     set      kfld
pt      set      x60
prs     set      kfld

enable  set      enxt
select  set      xtot
azbsa   set      zsalu
azd     set      x
wrs     set      jfld

pci      set      1
cond     set      validcm
jmp      set      jpnop
false    set      fetch
ad ffa729480be8fefceffd      wait      15

**      xxj      tk      006jk
*
*      load xj from tk.
*

ae ffffffff          begbuf -1
qt      set      x60
qrs     set      kfld
pt      set      x60
prs     set      kfld

enable  set      enxt
select  set      ttox
azbsa   set      zsalu
azd     set      x
wrs     set      jfld

pci      set      1
cond     set      validcm
jmp      set      jpnop
false    set      fetch
ae ffa719480be8fefceffd      wait      15

**      undefined instruction      007xx
*

af ffffffff          begbuf -1

```

```

true      set      *
jmp        set      jp
af  affffffe03ffffffeffc      wait    15

c1
**        org      0c1x
**        rj and friends.          01xxx
*
*        further decoding is achieved by an indexed
*        jump on the i-field, offset by b0h.
*

c1  ffffffff
c1  ffffffff03ffffffeffc      begbuf -1
                                set      jpifdl
                                wait    15

c2
**        org      0c2x
**        jp  bi+k          02iiK
*

c2  ffffffff
                                begbuf -1
                                qt       set      b
                                qrs      set      ifld
                                pt       set      penkf

                                select  set      add
                                azbsa   set      zsalu
                                bzbsa   set      zaenz
                                spc     set      ldpc18

                                azd      set      c          c0 has most recent jp target
                                wrs      set      mfld
                                wmfld   set      0

                                true     set      fetch
                                jmp      set      jp
c2  a7ff803e03f8efe3e9fc      wait    15

c3
**        org      0c3x
**        x register tests          03xjK
*
*        the hardware decodes the required test.
*
*        zr  xj,k          030jK
*        nz  xj,k          031jK
*        pl  xj,k          032jK
*        ng  xj,k          033jK
*        ir  xj,k          034jK
*        or  xj,k          035jK
*        df  xj,k          036jK
*        id  xj,k          037jK
*

c3  ffffffff
                                begbuf -1
                                qt       set      x60
                                qrs      set      jfld
                                bzbsa   set      zenk

```

```

                                spc      set      cldpc0

                                pci      set      2
                                true     set      fetch
                                jmp      set      jp
c3  a7fff8fe03ff2fffe7fe      wait     15

c4                                org      0c4x
                                **        the b-register conditional jumps are all
                                *          executed by the same micro-code instruction.
                                *
                                *          eq  bi,bj,k          04ijK
                                *          ne  bi,bj,k          05ijK
                                *          ge  bi,bj,k          06ijK
                                *          lt  bi,bj,k          07ijK
                                *

                                list      e
                                4         btest    dup      4
                                begbuf    -1
                                qt         set      b
                                qrs        set      ifld
                                pt         set      b
                                prs        set      jfld

                                select     set      sub
                                azbsa      set      zsalu

                                bzbsa      set      zenk
                                spc        set      cldpc1

                                pci         set      2
                                true        set      fetch
                                jmp         set      jp
                                wait        15

                                btest      endd
c4  ffffffff            begbuf    -1
                                qt         set      b
                                qrs        set      ifld
                                pt         set      b
                                prs        set      jfld

                                select     set      sub
                                azbsa      set      zsalu

                                bzbsa      set      zenk
                                spc        set      cldpc1

                                pci         set      2
                                true        set      fetch
                                jmp         set      jp
c4  a7ff900603f82fffe5fe      wait     15

                                endd
c5  ffffffff            begbuf    -1

```

```

qt      set      b
qrs     set      ifld
pt      set      b
prs     set      jfld

select  set      sub
azbsa   set      zsalu

bzbsa   set      zenk
spc     set      cldpcl

pci      set      2
true    set      fetch
jmp     set      jp
wait    15

c5  a7ff900603f82fffe5fe

endd
begbuf  -1
qt      set      b
qrs     set      ifld
pt      set      b
prs     set      jfld

select  set      sub
azbsa   set      zsalu

bzbsa   set      zenk
spc     set      cldpcl

pci      set      2
true    set      fetch
jmp     set      jp
wait    15

c6  a7ff900603f82fffe5fe

endd
begbuf  -1
qt      set      b
qrs     set      ifld
pt      set      b
prs     set      jfld

select  set      sub
azbsa   set      zsalu

bzbsa   set      zenk
spc     set      cldpcl

pci      set      2
true    set      fetch
jmp     set      jp
wait    15

c7  a7ff900603f82fffe5fe

endd
list    *

c8                                org      0c0x+10o

```

```

3      bxwait    equ    3
      **      bxi    xj      10ijk
      *

c8  ffffffffffffffff      alu60    xfra,bxwait
      **      bxi    xj*xk      11ijk
      *

c9  ffffffffffffffff      alu60    and,bxwait
      **      bxi    xj+xk      12ijk
      *

ca  ffffffffffffffff      alu60    or,bxwait
      **      bxi    xj-xk      13ijk
      *

cb  ffffffffffffffff      alu60    xor,bxwait
      **      bxi    -xk      14ikk
      *

cc  ffffffffffffffff      alu60    cxfrb,bxwait
      **      bxi    -xk*xj      15ijk
      *

cd  ffffffffffffffff      alu60    cand,bxwait
      **      bxi    -xk+xk      16ijk
      *

ce  ffffffffffffffff      alu60    cor,bxwait
      **      bxi    -xk-xj      17ijk
      *

cf  ffffffffffffffff      alu60    cxor,bxwait
      **
      *      shift - execute barrel shifter instructions.
      *
      *      shift type
      *
      *      entry type = select field.
      *              shift count on q bus.
      *              *prs* selects register for p bus.
      *
      *      this macro terminates the microinstruction.
      *

      shift      macro    type
      pt         set      x60

      select     set      type

```

	azbsa	set	zsbsh	
	azd	set	x	
	wrs	set	ifld	
	pci	set	1	
	cond	set	validcm	
	jmp	set	jpnop	
	false	set	fetch	
		wait	10	
		endm		
d0		org	0c0x+20o	
	**	lxi	jk	20ijk
	*			
d0	ffffffffffffffff	begbuf	-1	
	qt	set	qenjk	
	prs	set	ifld	
d0	ffa7c7c00be2feffeffd	shift	lxjk	
	**	axi	jk	21ijk
	*			
d1	ffffffffffffffff	begbuf	-1	
	qt	set	qenjk	
	prs	set	ifld	
d1	ffa7d7c00be2feffeffd	shift	axjk	
	**	lxi	bj,xk	22ijk
	*			
d2	ffffffffffffffff	begbuf	-1	
	qt	set	b	
	qrs	set	jfld	
	prs	set	kfld	
d2	ffa7e0c80be2feffeffd	shift	lxbj	
	**	axi	bj,xk	23ijk
	*			
d3	ffffffffffffffff	begbuf	-1	
	qt	set	b	
	qrs	set	jfld	
	prs	set	kfld	
d3	ffa7f0c80be2feffeffd	shift	axbj	
d4		org	0c0x+24o	
	**	nxi,bj	xk	24ijk
	*			
	*			
	*	1)	load xk and normalize shift count to fpu.	
	*	2)	bj <-- normalize shift count.	
	*	3)	xi <-- mantissa shifted by nsc.	
	*	4)	xi <-- xi \$ars\$ 1. (post-shift)	
	*		if exponent underflowed, go to expund.	
	*	5)	pack xi with normalize exponent (from fpu).	

```

*          go to next compass instruction.
* expund 6) set xi to all bits zero.
*          go to next compass instruction.
*

```

```

d4  ffffffff ffffffff          begbuf -1
                                qt      set    x60
                                qrs      set    kfld

                                ffpu      set    ldnorm
                                select    set    unround

                                pci        set    1                      start instruction fetch

                                cond       set    special
                                true       set    sosexec
                                false     set    nx.cont
                                jmp        set    jp
                                wait      15

d4  a441097e23fffffefe5

41
41  ffffffff ffffffff          nx.cont begbuf -1
                                ffpu      set    onms

                                qt        set    enfp
                                select    set    xfra
                                azbsa     set    zsalu
                                azd       set    b
                                wrs       set    jfld

                                true      set    *+1
                                jmp       set    jp
                                wait      15

41  42ff0bfe03e8fcffefe8

42  ffffffff ffffffff          begbuf -1
                                ffpu      set    onms

                                qt        set    enfp
                                pt        set    enfp
                                select    set    lxbj
                                azbsa     set    zsbsh
                                azd       set    x
                                wrs       set    ifld

                                true      set    *+1
                                jmp       set    jp
                                wait      15

42  43ffebde03e2feffefe8

43  ffffffff ffffffff          begbuf -1
                                qt        set    enone
                                pt        set    x60
                                prs       set    ifld
                                select    set    axbj
                                azbsa     set    zsbsh
                                azd       set    x
                                wrs       set    ifld

```

	ffpu	set	one	for overflow check only
	cond	set	normovl	exponent overflow check
	true	set	*+1	
	false	set	expund	return zero in x register
43 4445fdc263e2feffefec	jmp	set	jp	
		wait	15	
44 ffffffff		begbuf	-1	
	qt	set	x60	
	qrs	set	ifld	
	select	set	pack	
	ffpu	set	one	
	azbsa	set	zsenz	
	azd	set	x	
	wrs	set	ifld	
	enable	memchk	set	enprc
				fetch was started above
44 ffffa87c03e5fefc2eec	jmp	set	jpnop	
		wait	15	
45 ffffffff	expund	begbuf	-1	
	qt	set	zeros	
	select	set	xfra	
	azbsa	set	zsalu	
	azd	set	x	
	wrs	set	ifld	
	enable	memchk	set	enprc
				fetch was started above
45 ffff0ffc03e0fefc2efc	jmp	set	jpnop	
		wait	15	
46	free	setequ	*	
d5		org	0c0x+25o	
	**	zxi,bj	xk	25ijk
	*			
	*	rounding	normalize	- see regular normalize.
	*			
d5 ffffffff		begbuf	-1	
	qt	set	x60	
	qrs	set	kfld	
	ffpu	set	ldnorm	
	select	set	round	
	pci	set	1	start instruction fetch
	cond	set	special	

```

true      set      sosexec
false     set      nx.cont
jmp       set      jp
d5  a441197e23ffffffefe5      wait    15

d6                                     org      0c0x+26o

**        uxi,bj xk                26ijk
*
*        1) unpack exponent to bj.
*        2) unpack mantissa to xi.
*

d6  ffffffff          begbuf -1
qt      set      x60
qrs     set      kfld

select   set      unpkb

azbsa    set      zsenz
azd      set      b
wrs      set      jfld

true     set      ux.cont
jmp      set      jp
d6  46ff797e03edfcffeffc      wait    15

46                                     org      free
46  ffffffff          ux.cont begbuf -1
qt      set      x60
qrs     set      kfld

select   set      unpkx

azbsa    set      zsenz
azd      set      x
wrs      set      ifld

pci      set      1
cond     set      validcm
jmp      set      jpnop
false    set      fetch
46  ffa7c97c0be5feffeffd      wait    15
47      free      seteque *

d7                                     org      0c0x+27o

**        pxi      bj,xk                27ijk
*

d7  ffffffff          begbuf -1
qt      set      x60
qrs     set      kfld
pt      set      b
prs     set      jfld

select   set      pack

```

```

d7 ffa7a9040be5feffeffd

```

```

azbsa    set    zsenz
azd       set    x
wrs       set    ifld

pci       set    1
cond      set    validcm
jmp        set    jpnop
false     set    fetch
wait      15

```

```

**      common sequence for floating add/sub and
*      rounding floating add/sub.
*
*      notes  mutiple precision arithmetic is necessary
*              for these operations.  the *mpadd* alu
*              function provides this capability.
*
*              this routine needs more than one scratch
*              register, and thus it selects t-registers,
*              uses t0, t1, and t2, selects x-registers,
*              and copies the result to xi.
*
*              due to the above, these instructions should
*              only be executed in problem mode.
*
*      the first step in the sequence is the only one
*      not common to all 4 instructions.  it must
*
*      a) load xk to the fpu (ldadd2 or ldsub2),
*      b) clear the saved carry (enable initcy), and
*      c) set/clear the round flipflop.
*
*      the rest of the sequence follows.
*
*      1) load xj to fpu (ldadd1); select t registers.
*      2) t1 <-- small mantissa shifted for high add.
*      3) t1 <-- mpadd(large mantissa,t1).  (high add)
*      4) t0 <-- small mantissa shifted for low add.
*      5) t3 <-- mask to apply to small mantissa.
*      6) t2 <-- t0 $mask$ t3.
*      if bigsgn is positive
*      7) t0 <-- comp(t3).
*      else
*      8) t0 <-- comp(0).
*      endif
*      9) mpadd(t0,t2)  (low add, w/ end-around-carry).
*         t0 <-- comp(0).
*      10) t1 <-- mpadd(t0,t1).  (add carries to high)
*      if post-normalize needed
*      11) t1 <-- t1 $ars$ 1  (postnormalize).
*      endif
*      12) t1 <-- t1 $ars$ 1  (post-shift).
*      13) select x registers.
*      14) xi <-- t1.
*      15) pack mantissa and exponent for final result;
*           jump nextop.

```

```

*
* 1)      load xj to fpu; select t registers.

47          org      free
47  ffffffff          fad.seq begbuf -1
          qt      set   x60
          qrs     set   jfld
          ffpu    set   ldaddl

          enable  set   enx
          select  set   sett

          pci     set   1          start instruction fetch

          cond    set   special
          true    set   sosexec
* false      set   sosexec      temporary--always trap out
          false   set   *+1
          jmp     set   jp
47  a44838fe23ffffcfa9      wait  15

* 2)      t1 <-- small mantissa shifted for high add.

48  ffffffff          begbuf -1
          ffpu    set   addshf
          qt      set   enfp
          pt      set   enfp

          select  set   axbj
          azbsa   set   zsbsh
          azd     set   x
          wrs     set   mfld
          wmfld   set   1

          true    set   *+1
          jmp     set   jp
48  49ffffbde03faf7efb4      wait  15

* 3)      t1 <-- mpadd(large mantissa,t1).

49  ffffffff          begbuf -1
          ffpu    set   addman
          qt      set   x60
          qrs     set   mfld
          rmfld   set   1
          pt      set   enfp

          select  set   mpadd
          azbsa   set   zsalu
          azd     set   x
          wrs     set   mfld
          wmfld   set   1

          true    set   *+1
          jmp     set   jp
49  4affa9de03f8fe27efb8      wait  15

```

```

* 4)      t0 <-- small mantissa shifted for low add.

4a  ffffffff            begbuf -1
                                set  addshfl
                                qt   set  enfp
                                pt   set  enfp

                                select set  lxbj
                                azbsa set  zsbsh

                                azd   set  x
                                wrs   set  mflld
                                wmflld set  0

                                true  set  *+1
                                jmp    set  jp
4a  4bffebede03fafee3eff4      wait  15

* 5)      t3 <-- mask to apply to small mantissa

4b  ffffffff            begbuf -1
                                set  addmask
                                qt   set  enfp
                                pt   set  zeros

                                select set  mask
                                azbsa set  zsbsh
                                azd   set  x
                                wrs   set  mflld
                                wmflld set  3

                                true  set  *+1
                                jmp    set  jp
4b  4cffbbfe03fafeefeff8      wait  15

* 6)      t2 <-- t0 $mask$ t3.

4c  ffffffff            begbuf -1
                                qt   set  x60
                                enable set  qreg0
                                pt   set  x60
                                prs   set  mflld
                                rmflld set  3

                                select set  and
                                azbsa set  zsalu
                                azd   set  x
                                wrs   set  mflld
                                wmflld set  2

                                cond   set  bigsgn
                                true    set  *+1           positive
                                false   set  negman        negative
                                jmp     set  jp
4c  4d4e19ce77f8fe69effc      wait  15

* 7)      t0 <-- comp(t3).

```

```

*          sign extend zeros to right for large mantissa,
*          down to the lsb of small mantissa.
*          below the lsb of the small mantissa insert ones
*          to propagate the end-around-carry.

4d  ffffffff            begbuf -1
                                set  x60
                                set  mfld

                                rmfld set  3
                                select set  cxfrb
                                azbsa set  zsalu
                                azd   set  x
                                wrs   set  mfld
                                wmfld set  0

                                true  set  facont
                                jmp    set  jp
4d  4fff4fce03f8fe63effc      wait  15

* 8)          t0 <-- comp(0).
*          sign extend ones to right for large mantissa,
*          down to the lsb of small mantissa.
*          below the lsb of the small mantissa insert ones
*          to propagate the end-around-carry.

4e  ffffffff            negman begbuf -1
                                pt    set  zeros
                                select set  cxfrb
                                azbsa set  zsalu
                                azd   set  x
                                wrs   set  mfld
                                wmfld set  0

                                true  set  *+1
                                jmp    set  jp
4e  4fff4ffe03f8fee3effc      wait  15

* 9)          mpadd(t0,t2) (low add, with end-around-carry).

4f  ffffffff            facont begbuf -1
                                qt    set  x60
                                enable set  qreg0
                                pt    set  x60
                                prs   set  mfld
                                rmfld set  2

                                select set  mpadd
                                azbsa set  zsalu

                                true  set  *+1
                                jmp    set  jp
4f  50ffa9ce03f8ff5deffc      wait  15

*
*          t0 <-- comp(0)
*

```

```

50  ffffffff            begbuf -1
                                pt      set      zeros
                                select   set      cxfrb
                                azbsa    set      zsalu
                                azd       set      x
                                wrs       set      mfld
                                wmfld     set      0

                                true      set      *+1

                                jmp        set      jp
50  51ff4ffe03f8fee3effc      wait      15

* 10)      t1 <-- mpadd(t0,t1); add end-around-carry to high
*           word of result; jump over next step if post-
*           normalize not necessary; latch post-normalize
*           bit in exponent computation hardware.

51  ffffffff            begbuf -1
                                qt        set      x60
                                enable    set      qreg0
                                pt        set      x60
                                prs       set      mfld
                                rmfld     set      1

                                select    set      mpadd
                                azbsa     set      zsalu
                                azd        set      x
                                wrs        set      mfld
                                wmfld     set      1

                                ffpu      set      addman      latch post-normalize bit

                                cond       set      addnorm
                                true       set      nonorm
                                false      set      *+1
                                jmp        set      jp
51  5352a9ce27f8fe25efb8      wait      15

* 11)      t1 <-- t1 $ars$ 1 (postnormalize).

52  ffffffff            begbuf -1
                                qt        set      enone
                                pt        set      x60
                                prs       set      mfld
                                rmfld     set      1

                                select    set      axbj
                                azbsa     set      zsbsh
                                azd        set      x
                                wrs        set      mfld
                                wmfld     set      1

                                true       set      *+1
                                jmp        set      jp
52  53fffdce03fafe27effc      wait      15

```

```

* 12)      t1 <-- t1 $ars$ 1 (post-shift).

53  ffffffff            nonorm  begbuf -1
                        qt      set   enone
                        pt      set   x60
                        prs      set   mfld
                        rmfld    set   1

                        select   set   axbj

                        azbsa    set   zsbsh
                        azd      set   x
                        wrs      set   mfld
                        wmfld    set   1

                        true     set   *+1
                        jmp      set   jp
53  54fffdce03fafe27effc      wait  15

* 13)      select x registers.

54  ffffffff            begbuf -1
                        enable   set   enx
                        select   set   setx

                        true     set   *+1
                        jmp      set   jp
54  55ff0ffe03ffffceffc      wait  15

* 14)      xi <-- t1.

55  ffffffff            begbuf -1
                        rmfld    set   1
                        qt      set   x60
                        qrs      set   mfld
                        pt      set   x60
                        prs      set   mfld

                        enable   set   enx
                        select   set   ttox
                        azbsa    set   zsalu
                        azd      set   x
                        wrs      set   ifld

                        true     set   *+1
                        jmp      set   jp
55  56ff19ce03e0fe3ceffc      wait  15

* 15)      pack mantissa and exponent for final result;
*          jump nextop.

56  ffffffff            begbuf -1
                        memchk
                        enable   set   enprc      instruction fetch started above

                        qt      set   x60
                        qrs      set   ifld

```

		ffpu	set	addexp	
		select	set	pack	
		azbsa	set	zsenz	
		azd	set	x	
		wrs	set	ifld	
56	fffffa87c03e5fefc2ebc	jmp	set	jpnop	
			wait	15	
d8	57	free	setequ	*	
			org	0c0x+30o	
		**	fxi	xj+xk	(30ijk)
		*			
		*		see documentation of common add sequence.	
		*			
d8	fffffffffffffffff		begbuf	-1	
		qt	set	x60	
		qrs	set	kfld	
		ffpu	set	ldadd2	
		enable	set	initcy	clear saved carry
		select	set	unround	
		cond	set	mon	
		true	set	*	
		false	set	fad.seq	
		jmp	set	jp	
d8	d847097e67fffffd2fac		wait	15	
		**	fxi	xj-xk	(31ijk)
		*			
		*		see documentation of common add sequence.	
		*			
d9	fffffffffffffffff		begbuf	-1	
		qt	set	x60	
		qrs	set	kfld	
		ffpu	set	ldsub2	
		enable	set	initcy	clear saved carry
		select	set	unround	
		cond	set	mon	
		true	set	*	
		false	set	fad.seq	
		jmp	set	jp	
d9	d947097e67fffffd2fb0		wait	15	
		**	dxi	xj+xk	(32ijk)
		*			
		*		instruction implemented in compass for now.	
		*			

```

da ffffffff            trap    xv.op,1

                        **      dxi    xj-xk                (33ijk)
                        *
                        *      instruction implemented in compass for now.
                        *

db ffffffff            trap    xv.op,1

                        **      rxi    xj+xk                (34ijk)
                        *
                        *      see documentation of common add sequence.
                        *

dc ffffffff            begbuf -1
                        qt      set    x60
                        qrs     set    kfld
                        ffpu    set    ldadd2

                        enable   set    initcy      clear saved carry
                        select  set    round

                        cond     set    mon
                        true     set    *
                        false    set    fad.seq
                        jmp      set    jp
dc dc47197e67ffffd2fac  wait    15

                        **      rxi    xj-xk                (35ijk)
                        *
                        *      see documentation of common add sequence.
                        *

dd ffffffff            begbuf -1
                        qt      set    x60
                        qrs     set    kfld
                        ffpu    set    ldsub2

                        enable   set    initcy      clear saved carry
                        select  set    round

                        cond     set    mon
                        true     set    *
                        false    set    fad.seq
                        jmp      set    jp
dd dd47197e67ffffd2fb0  wait    15

de                                org    0c0x+360

                        6      ixwait equ    6
                        **      ixi    xj+xk                36ijk
                        *

de ffffffff            alu60  add,ixwait

                        **      ixi    xj-xk                37ijk
                        *

```

```

df  ffffffffffffffff          alu60  sub,ixwait

                                **      fxi      xj*xk          40ijk
                                *
                                *      load second operand from (xk).

e0                                org      0c0x+40o
e0  ffffffffffffffff          begbuf -1
                                qt      set      x60
                                qrs      set      kfld
                                select    set      unround

                                ffpu      set      ldmul2

                                true      set      fx.cont
                                jmp       set      jp
e0  57ff097e03fffffef84      wait      15

                                *      load first operand from (xj).
                                *      trap out for special case operands.

57                                org      free
57  ffffffffffffffff          fx.cont begbuf -1
                                qt      set      x60
                                qrs      set      jfld

                                pci      set      1          start instruction fetch

                                ffpu      set      ldmul1

                                cond      set      special
                                true      set      sosexec
                                false     set      **+1
                                jmp       set      jp
57  a458f8fe23fffffef81      wait      15

                                *      perform multiply.

58  ffffffffffffffff          mpy      sel=single,wt=10

                                *      write result to x(i)
                                *      check for exponenet overflow
                                *      jump nextop

59  ffffffffffffffff          begbuf -1
                                memchk
                                enable    set      enprc

                                ffpu      set      omulsp
                                select    set      postn          enable postnormalize

                                azbsa     set      zsenz
                                azd        set      x
                                wrs        set      ifld

```

		cond	set	mulovl	
		false	set	sosovl	
		jmp	set	jpnop	
59	ffa30ffc5fe5fetc2ea4		wait	15	
	5a	free	setequ	*	
		**	rxl	xj*xk	41ijk
		*			
		*	load	second operand from (xk).	
e1			org	0c0x+410	
e1	ffffffffffffffff		begbuf	-1	
		qt	set	x60	
		qrs	set	kfld	
		select	set	round	
		ffpu	set	ldmul2	
		true	set	fx.cont	
		jmp	set	jp	
e1	57ff197e03fffffef84		wait	15	
		**	dxl	xj*xk	42ijk
		*			
		*	load	second operand from (xk).	
e2			org	0c0x+420	
e2	ffffffffffffffff		begbuf	-1	
		qt	set	x60	
		qrs	set	kfld	
		select	set	unround	
		ffpu	set	ldmul2	
		true	set	dx.cont	
		jmp	set	jp	
e2	5aff097e03fffffef84		wait	15	
		*	load	first operand from (xj).	
		*	check	for special operands.	
5a			org	free	
5a	ffffffffffffffff	dx.cont	begbuf	-1	
		qt	set	x60	
		qrs	set	jfld	
		ffpu	set	ldmul1	
		pci	set	1	start instruction fetch
		cond	set	special	
		true	set	sosexec	
		false	set	*+1	

```

5a  a45bf8fe23ffffffef81      jmp      set      jp
                                wait     15

                                *          perform multiply.

5b  ffffffff
                                mpy      sel=single,wt=10

                                *          write result to x(i)
                                *          check for exponenet overflow
                                *          jump nextop

5c  ffffffff
                                begbuf -1
                                memchk
                                enable   set      enprc

                                ffp      set      omuldp
                                select   set      postn          enable postnormalize

                                azbsa    set      zsenz
                                azd       set      x
                                wrs       set      ifld

                                cond      set      mulovl
                                false     set      sosovl
                                jmp        set      jpnop
5c  ffa30ffc5fe5fefc2ea0      wait     15
5d                                setequ  *

**          divmac - perform a step in the divide algorithm.
*
*          divmac f=f
*          divmac f=f,src=source,dst=dest,sel=sel,next=next
*
*          entry  f = fpu function.
*          source = (optional) source register (0..7)
*          dest   = (optional) destination reg (0..7).
*          sel    = (optional) *select* field.
*          next   = (optional) next micro-code address.
*
*          see the hardware manual for documentation on
*          the divide microcode and algorithm
*

divmac    macroe f,src,dst,sel,next
          begbuf -1
          ffp      set      f
*
          src      ifc      ne,*src**
          qt       set      x60
          qrs      set      mfld
          rmfld    set      src
          src      endif
*
          dst      ifc      ne,*dst**
          azbsa    set      zsenz
          azd       set      x
          wrs       set      mfld

```

```

        wmfld    set    dst
        dst      endif
*
        select   ifc    ne,*sel**,1
        select   set    sel
*
        nxt      ifc    ne,*next**
        true     set    next
        nxt      else
        true     set    *+1
        nxt      endif
        jmp      set    jp
                wait   15
                endm

**          fxi      xj/xk                (44ijk)
*
*          divide is done in three stages.
*
*          in the first stage, the operands are loaded and
*          are checked for special values.  the numerator
*          is normalized.
*
*          in the second stage, an approximate value for the
*          result is generated by an iterative algorithm
*          using multiplies.
*
*          in the third stage, the result from the second
*          stage is made exact by an iterative correction
*          algorithm, using trial multiplies to check for the
*          correct result.
*
*          notes   this algorithm uses t-registers as scratch,
*                  and thus can only be executed in problem
*                  mode.
*
*                  this algorithm requires that both operands
*                  be normalized.  to this end the numerator
*                  is normalized in the initializations, but
*                  it is assumed that the denominator is
*                  already normalized.  if the denominator
*                  is not normalized, a trap is taken.
*
*          select t-registers and initialize multiplier chip
*          controls (rounding, signed numbers, etc).

4          org      0c0x+44o
4  ffffffff          begbuf -1

        ffpu      set    ldmul2          initialize multipliers
        enable    set    enx
        select     set    sett

        cond      set    mon
        true       set    *              illegal in monitor
        false      set    fdiv
        jmp        set    jp

```

```

e4 e45d3ffe67ffffcef84      wait 15

*      t7 <-- 0; to flag -fx- instruction.

5d      org free
5d ffffffff             fdiv begbuf -1
      qt      set zeros
      select set xfra
      azbsa   set zsalu

      azd     set x
      wrs     set mfld
      wmfld   set 7

      true    set dv.cont

      jmp     set jp
5d 5fff0ffe03f8feffeffc    wait 15
      free    setequ *
      5e

**      rxi    xj/xk      (45ijk)
*
*      same as 44 instruction.
*

e5      org 0c0x+45o
e5 ffffffff             begbuf -1
      ffpu    set ldmul2      initialize multipliers
      enable  set enxt
      select  set sett

      cond    set mon
      true    set *          illegal in monitor
      false   set rdiv
      jmp     set jp
e5 e55e3ffe67ffffcef84    wait 15

*      t7 <-- 1; to flag -rx- instruction.

5e      org free
5e ffffffff             rdiv begbuf -1
      qt      set enone
      select  set xfra
      azbsa   set zsalu

      azd     set x
      wrs     set mfld
      wmfld   set 7

      true    set dv.cont
      jmp     set jp
5e 5fff0dfe03f8feffeffc    wait 15
      free    setequ *
      5f

*
*      stage 1 of divide.
*
*      copy operands to t-registers, and also check

```

```

*          for special case input operands.  note that
*          the check for special operands for divides
*          requires "mantissa=0" from the x register test
*          circuit -- so loading t-registers must be done
*          in separate instructions than testing operands.
*
*          t1 <-- xk          (denominator)
*          t0 <-- xj          (numerator)
*          test t1 in fpu (requires x-register test).
*          test t0 in fpu; trap out if needed.
*
*          t1 <-- xk.

5f          org          free
5f  ffffffff            dv.cont begbuf -1
                     enable set      enxtnxt      read x, write t
                     select set      xtntot

                     qt      set      x60          read xk
                     qrs     set      kfld
                     pt      set      x60
                     prs     set      kfld

                     azbsa   set      zsalu        write t1
                     azd     set      x
                     wrs     set      mfld
                     wmfld   set      1

                     true    set      *+1
                     jmp     set      jp
5f  60ff294a03f8fee4effc      wait    15

*          t0 <-- xj.

60  ffffffff            begbuf -1
                     enable set      enxtnxt      read x, write t
                     select set      xtntot

                     qt      set      x60          read xj
                     qrs     set      jfld
                     pt      set      x60
                     prs     set      jfld

                     azbsa   set      zsalu        write t0
                     azd     set      x
                     wrs     set      mfld
                     wmfld   set      0

                     true    set      *+1
                     jmp     set      jp
60  61ff28c603f8fee0effc      wait    15

61  ffffffff            divmac f=llddiv2,src=1      test denominator
*          divmac f=llddiv1,src=0      test numerator
62  ffffffff            begbuf -1
                     qt      set      x60

```

```

        qrs      set      mfld
        rmfld    set      0

        ffp      set      lddiv1

        pci      set      1          point to next instruction

        cond     set      special
        true     set      sosexec
        false    set      *+1
        * false  set      sosexec    temporary -- always trap out
        jmp      set      jp

62  a463f9fe23ffff1fefc1      wait 15

        *        pre-normalize the numerator (in t0).
        *
        *        load t0 and normalize shift count to fpu
        *        t0 <-- mantissa shifted by nsc
        *        t0 <-- t0 $ars$ 1 (post-shift)
        *        if exponent underflows go to "itiszer"
        *        pack t0 with normalize exponent (from fpu).
        *        check if t0 = 0; go to "notzer" if not.
        *        itiszer select x registers.
        *        xi <-- 0; jump nextop.
        *        notzer

63  ffffffff            begbuf -1
        qt          set      x60
        qrs         set      mfld
        rmfld       set      0

        ffp         set      ldnorm
        select      set      unround

        true        set      *+1
        jmp         set      jp

63  64ff09fe03ffff1fefe4      wait 15

        *        t0 <-- mantissa shifted by nsc.

64  ffffffff            begbuf -1
        ffp         set      onms
        qt          set      enfp
        pt          set      enfp

        select      set      lxbj
        azbsa       set      zsbsh
        azd         set      x
        wrs         set      mfld
        wmfld       set      0

        true        set      *+1
        jmp         set      jp

64  65ffebde03fafee3efe8      wait 15

        *        t0 <-- xi $ars$ 1. (post-shift)

```

```

65  ffffffff            begbuf -1
                                qt      set      enone
                                pt      set      x60
                                prs     set      mfld
                                rmfld   set      0

                                select  set      axbj
                                azbsa   set      zsbsh
                                azd     set      x
                                wrs     set      mfld
                                wmfld   set      0

                                ffpu    set      one          for overflow check only
                                cond     set      normovl       exponent overflow check
                                true     set      *+1          did not underflow
                                false    set      itiszer       return zero in x register
                                jmp      set      jp
65  6668fdce63fafe03efec    wait    15

                                *          pack t0 with normalize exponent (from fpu).

66  ffffffff            begbuf -1
                                qt      set      x60
                                qrs     set      mfld
                                rmfld   set      0
                                select  set      pack

                                ffpu    set      one

                                azbsa   set      zsenz
                                azd     set      x
                                wrs     set      mfld
                                wmfld   set      0

                                * cond   set      normovl
                                true     set      *+1
                                * false  set      sosexec
                                jmp      set      jp
66  67ffa9fe03fdfe03efec    wait    15

                                *          check if the numerator is +0 or -0.
                                *          (check only exponent, mantissa can be non-zero,
                                *          but underflow exponent is all that matters)
                                *          if so, +0 is the answer.

67  ffffffff            begbuf -1
                                qt      set      x60
                                qrs     set      mfld
                                rmfld   set      0

                                cond     set      aexpzer
                                true     set      itiszer
                                false    set      notzer
                                jmp      set      jp
67  686af9fe2bffff1feffc    wait    15

```

```

*          it is zero.  select x registers.

68  ffffffff            itiszer begbuf -1
                             enable set  enxtnxt
                             select set  setx

                             true  set  **+1
                             jmp   set  jp
68  69ff0ffe03ffffffc   wait  15

*          xi <-- 0; jump nextop.

69  ffffffff            begbuf -1
                             qt     set  zeros
                             select set  xfra
                             azbsa set  zsalu

                             azd    set  x
                             wrs    set  ifld

                             true  set  fetch
                             jmp   set  jp
69  a7ff0ffe03e0fefffc   wait  15

*
*          stage 2 of divide.
*
*          approximate i = j/k by repetitive multiplication.
*
*          1) y = f(k) = approximately 1/k.  (look-up rom)
*          2) d1 = k*y; r1 = j*y.
*          3) d2 = d1*(2-d1); r2 = r1*(2-d1).
*          4) r3 = r2*(2-d2).
*
*          calculate d1 = k*y.
*
*          k is the denominator which is in t1.
*          y is approximately 1/k from lookup rom.

6a  ffffffff            notzer  divmac f=divlook,src=1,sel=unround
6b  ffffffff            divmac f=divlook,src=1,sel=unround

6c  ffffffff            divmac f=lddiv2,src=1
6d  ffffffff            mpy     sel=double,wt=15

*          calculate r1 = j*y; write d1 to t3.

6e  ffffffff            divmac f=lddiv1,src=0
6f  ffffffff            divmac f=oiters,sel=nopostn,dst=3
70  ffffffff            mpy     sel=double,wt=15

*          calculate d2 = d1*(2-d1); write r1 to t3.

71  ffffffff            divmac f=lditer1,src=3          loads 2-d1
72  ffffffff            divmac f=lditer2,src=3
73  ffffffff            divmac f=oiters,sel=nopostn,dst=3
74  ffffffff            mpy     sel=double,wt=15

```

```

*          calculate r2 = r1*(2-d1); write d2 to t3.

75  ffffffffffffffff      divmac f=lditer2,src=3
76  ffffffffffffffff      divmac f=oiters,sel=nopostn,dst=3
77  ffffffffffffffff      mpy      sel=double,wt=15

*          calculate r3 = r2*(2-d2).

78  ffffffffffffffff      divmac f=lditer1,src=3          loads 2-d2
79  ffffffffffffffff      divmac f=oiters,sel=nopostn,dst=3  r2

*          decrement intermediate result
*          due to rounding errors it may be 1 too big.

7a  ffffffffffffffff      begbuf -1
                                qt      set      x60
                                qrs      set      mfld
                                pt      set      enone
                                rmfld    set      3
                                select    set      sub
                                azbsa    set      zsalu
                                azd      set      x
                                wrs      set      mfld
                                wmfld    set      3

                                true     set      *+1
                                jmp      set      jp
7a  7bff99ee03f8fe6feffc      wait    15

7b  ffffffffffffffff      divmac f=lditer2,src=3          loads r2
7c  ffffffffffffffff      mpy      sel=double,wt=15

*          write approximate result r3 to t3;
*          check for overflow or underflow conditions.

7d  ffffffffffffffff      begbuf -1
                                ffpu     set      odiv
                                select    set      postn

                                azbsa    set      zsenz
                                azd      set      x
                                wrs      set      mfld
                                wmfld    set      3

                                cond      set      mulovl
                                false     set      sosexec
                                true      set      *+1
                                jmp      set      jp
7d  7ea40ffe5ffdfefefed0      wait    15

*
*          stage 3 of divide.
*
*          description of correction algorithm.
*
*          ldmul2 with t1 (denominator).

```

```

* man      t4 <-- t3.
*          t3 <-- t3 increased.
*          ldmul1 with t3 (test result).
*          multiply (result * denominator).
*          t2 <-- omulsp.
*          t5 <-- t0 - t2 (compare with numerator).
*          test t5.
*          if nonzero
*            t0 - t2; if bit 59 same, go to *man*, else *done*.
*
*        endif
*
* zero     t5 <-- omuldp (check bottom of trial multiply).
*          t5 <-- unpack mantissa(t5)
*          test t5.
*          if zero
*            t4 <-- t3 (latest result is exact answer).
*          endif
*
* done     xi <-- t4; xi <-- t3 for -rx- instruction.
*          select x-registers and done.
*
*
*          load second multiply input with t1.

7e ffffffff            man      begbuf -1
                        qt       set    x60
                        qrs      set    mfld
                        rmfld    set    1
                        ffpu     set    ldmul2
                        select   set    unround

                        true     set    *+1
                        jmp      set    jp
7e 7fff09fe03ffff3fef84      wait   15

*          t3 <-- t4
*          disable integer multiply mantissa blanking

* man      begbuf -1          moved label up: noise problems
                        begbuf -1
                        qt       set    x60
                        qrs      set    mfld
                        rmfld    set    3

                        select   set    xfra
                        azbsa    set    zsalu
                        bzbsa    set    zaenz
                        ffpu     set    disblank

                        azd      set    x
                        wrs      set    mfld
                        wmfld    set    4

                        true     set    nfree
                        jmp      set    jp

```

```

7f 89ff09fe03f8ee73efd4      wait    15

80      ofree    equ    *
89      org      nfree

*      set x3 to increased x3.
*      increased x3 means:
*          x3+1 if x3 is positive;
*          x3-1 if x3 is negative;
*          flip bit 47 if bit 48 changes.

89 ffffffff            begbuf -1
      qt         set    x60
      qrs        set    mfld
      rmfld      set    3
      pt         set    enone
      select     set    sub
      enable     set    divfix      choose add/sub, fix overflows
      azbsa      set    zsalu
      azd        set    x
      wrs        set    mfld
      wmfld      set    3

      true       set    *+1
      jmp        set    jp
89 8aff99ee03f8fe6f2ffc      wait    15
*      load first multiply input with t3.
*      check for -0 exponent; will be generated when
*      2**47 (which has a +0 exponent) is approached
*      from below.

8a ffffffff            begbuf -1
      qt         set    x60
      qrs        set    mfld
      rmfld      set    3

      ffpu       set    ldmul1

      cond       set    special
      true       set    sosexec
      false      set    *+1
      jmp        set    jp
8a a48bf9fe23ffff7fef80      wait    15

8b ffffffff            mpy      sel=single,wt=10
*      output single precision result to t2.

8c ffffffff            begbuf -1
      ffpu       set    omulsp
      select     set    postn

      azbsa      set    zsenz
      azd        set    x
      wrs        set    mfld
      wmfld      set    2

```

```

      true      set    *+1
      jmp       set    jp
8c  8dff0ffe03fdfeebefa4      wait    15

      *          t5 <-- numerator - approximation * denominator.

8d  ffffffff
      qt        begbuf -1
      enable    set    x60          t0
               set    qreg0

      pt        set    x60          t2
      prs       set    mfld
      rmfld     set    2

      select    set    sub
      azbsa     set    zsalu
      azd       set    x
      wrs       set    mfld
      wmfld     set    5

      true      set    *+1
      jmp       set    jp
8d  8eff99ce03f8fe55effc      wait    15

      *          if t5 is zero, check if lower part of multiply
      *          is zero, indicating an exact match.

8e  ffffffff
      qt        begbuf -1
      qrs       set    x60
      rmfld     set    mfld
               set    5

      cond      set    azero
      true      set    zero
      false     set    *+1
      jmp       set    jp
8e  908ff9felbffffbfeffc      wait    15

      *          compute t0 - t2; bit 59 change means target value
      *          has been passed; otherwise iterate.

8f  ffffffff
      qt        begbuf -1
      enable    set    x60          t0
               set    qreg0

      pt        set    x60          t2
      prs       set    mfld
      rmfld     set    2

      select    set    sub
      azbsa     set    zsalu

      cond      set    fdivlt
      true      set    done
      false     set    man
      jmp       set    jp
8f  947e99ce2ff8ff5deffc      wait    15

```

```

*          the top part of (numerator - approx * denominator)
*          was found to be zero.  if the bottom part of
*          (approx * denominator) is nonzero, then the target
*          value was passed.
*          output double precision result to t5.

90  ffffffff            zero    begbuf -1
                        ffpu    set    omuldp
                        select   set    postn

                        azbsa    set    zsenz
                        azd      set    x
                        wrs      set    mflld
                        wmflld   set    5

                        true     set    *+1
                        jmp      set    jp
90  91ff0ffe03fddef7efa0      wait   15

*          t5 <-- unpack mantissa(t5)

91  ffffffff            begbuf -1
                        qt       set    x60
                        qrs      set    mflld
                        rmflld   set    5

                        select   set    unpkx

                        azbsa    set    zsenz
                        azd      set    x
                        wrs      set    mflld
                        wmflld   set    5

                        true     set    *+1
                        jmp      set    jp
91  92ffc9fe03fdfeb7effc      wait   15

*          if t5 is zero, use latest iteration result.

92  ffffffff            begbuf -1
                        qt       set    x60
                        qrs      set    mflld
                        rmflld   set    5

                        cond     set    azero
                        true     set    *+1
                        false    set    done
                        jmp      set    jp
92  9394f9fe1bffffbfeffc      wait   15

*          t4 <-- t3.

93  ffffffff            begbuf -1
                        qt       set    x60
                        qrs      set    mflld

```

	rmfld	set	3	
	select	set	xfra	
	azbsa	set	zsalu	
	azd	set	x	
	wrs	set	mfld	
	wmfld	set	4	
	true	set	*+1	
93	94ff09fe03f8fe73effc	jmp	set jp	
		wait	15	
	*	t4 is the -fx- result; t3 is the -rx- result.		
94	ffffffffffffffff	done	begbuf -1	
	qt	set	x60	read t7
	qrs	set	mfld	
	rmfld	set	7	
	ffpu	set	ldmul2	clear disblank flip-flop
	enable	set	enxt	select x registers
	select	set	setx	
	cond	set	azero	
	true	set	fdivx	
	false	set	rdivx	
94	959609felbffffcfef84	jmp	set jp	
		wait	15	
	*	it was a non-rounding instruction.		
	*	xi <-- t4; jump nextop.		
95	ffffffffffffffff	fdivx	begbuf -1	
	enable	set	enxt	read t, write x
	select	set	ttox	
	rmfld	set	4	read t4
	qt	set	x60	
	qrs	set	mfld	
	pt	set	x60	
	prs	set	mfld	
	azbsa	set	zsalu	write xi
	azd	set	x	
	wrs	set	ifld	
	true	set	fetch	
95	a7ff19ce03e0fe9ceffc	jmp	set jp	
		wait	15	
	*	it was a rounding instruction.		
	*	xi <-- t3; jump nextop.		
96	ffffffffffffffff	rdivx	begbuf -1	
	enable	set	enxt	read t, write x
	select	set	ttox	

```

rmfld    set    3                read t3
qt       set    x60
qrs      set    mfld
pt       set    x60
prs      set    mfld

azbsa    set    zsalu            write xi

azd      set    x
wrs      set    ifld

true     set    fetch
jmp      set    jp
96  a7ff19ce03e0fe7ceffc  97  wait    15
                                setequ  *
free

e3                                     org    0c0x+43o

**       mxi   jk                43ijk
*

e3  ffffffff

                                begbuf -1
                                qt       set    qenjk
                                pt       set    zeros

                                select   set    mask
                                azbsa    set    zsbsh
                                azd      set    x
                                wrs      set    ifld

                                pci       set    1
                                cond     set    validcm
                                jmp      set    jpnop
                                false    set    fetch
e3  ffa7b7fc0be2feffeffd  wait    15

e6                                     org    0c0x+46o
                                no       46000
**
*

e6  ffffffff

                                begbuf -1
                                pci       set    1
                                * enable set    clrbe
                                cond     set    validcm
                                jmp      set    jpnopni    no interrupts allowed here
                                false    set    fetch
e6  ffa7fffe8bfffffeffd  wait    2

e7                                     org    0c0x+47o

**       cxi   xk                47ikk
*

e7  ffffffff

                                begbuf -1
                                qt       set    x60
                                qrs      set    kfld

```

```

ffpu      set      enpop

azbsa     set      zsenz
azd       set      x
wrs       set      ifld

pci       set      1
cond      set      validcm
jmp       set      jpnop
false     set      fetch
wait      15

e7 ffa7f97c0be5feffeffl

***      notes on execution of -sa- instructions.
*
*      1) two micro-code steps are required. the
*      first sets the register, and the second performs
*      the memory access. the second step is reached
*      by an indexed jump on the i field, offset by b8h.
*
*      2) if a new compass instruction word is needed,
*      an extra step is inserted between the first and
*      second step to finish fetching the instruction.
*
**      seta - common actions for -sa- instructions.
*      this macro terminates the microinstruction.
*

seta      macro
memchk

qrs       set      jfld
prs       set      kfld
azbsa     set      zsalu
bzbsa     set      zaenz

azd       set      a
wrs       set      ifld
bzd       set      ldcmac

cond      set      validcm
jmp       set      jpifd0
false     set      safetch
wait      15
endm

*      this is the extra microcode step for the set a
*      instruction, if it needs to fetch a new compass
*      instruction word.

97      org      free
97 ffffffff      begbuf -1

safetch   memchk
enable    set      enprc
jmp       set      jpifd0
97 ffffffff83fffffc2efc      wait      15
98      free     setequ *
e8      org      0c0x+50o

```

	**	sai	aj+k	50ijk
	*			
e8	ffffffffffffffff	begbuf	-1	
	qt	set	a	
	pt	set	penkf	
	select	set	add	
	pci	set	2	
e8	ff9782b889e0e9ffeeffe	seta		
	**	sai	bj+k	51ijk
	*			
e9	ffffffffffffffff	begbuf	-1	
	qt	set	b	
	pt	set	penkf	
	select	set	add	
	pci	set	2	
e9	ff9780b889e0e9ffeeffe	seta		
	**	sai	xj+k	52ijk
	*			
ea	ffffffffffffffff	begbuf	-1	
	qt	set	x18	
	pt	set	penkf	
	select	set	add	
	pci	set	2	
ea	ff9784b889e0e9ffeeffe	seta		
	**	sai	xj+bk	53ijk
	*			
eb	ffffffffffffffff	begbuf	-1	
	qt	set	x18	
	pt	set	b	
	select	set	add	
	pci	set	1	
eb	ff97848889e0e9ffeeefd	seta		
	**	sai	aj+bk	54ijk
	*			
ec	ffffffffffffffff	begbuf	-1	
	qt	set	a	
	pt	set	b	
	select	set	add	
	pci	set	1	
ec	ff97828889e0e9ffeeefd	seta		
	**	sai	aj-bk	55ijk
	*			
ed	ffffffffffffffff	begbuf	-1	
	qt	set	a	
	pt	set	b	

```

ed ff97928889e0e9ffeefd      select  set  sub
                                pci    set  1
                                seta
                                **      sai  bj+bk      56ijk
                                *

ee ffffffff             begbuf -1
                                qt    set  b
                                pt    set  b
                                select set  add
                                pci    set  1
ee ff97808889e0e9ffeefd      seta
                                **      sai  bj-bk      57ijk
                                *

ef ffffffff             begbuf -1
                                qt    set  b
                                pt    set  b
                                select set  sub
                                pci    set  1
ef ff97908889e0e9ffeefd      seta
                                **      setb - common actions for -sb- instructions.
                                *      this macro terminates the microinstruction.
                                *

                                setb  macro
                                qrs    set  jfld
                                prs    set  kfld
                                azbsa  set  zsalu

                                azd    set  b
                                wrs    set  ifld

                                cond    set  validcm
                                jmp     set  jpnop
                                false   set  fetch
                                wait    6
                                endm

f0                                org  0c0x+60o
                                **      sbi  aj+k      60ijk
                                *

f0 ffffffff             begbuf -1
                                qt    set  a
                                pt    set  penkf
                                select set  add
                                pci    set  2
f0 ffa782b80be0fcffeffe      setb
                                **      sbi  bj+k      61ijk
                                *

```

f1	ffffffffffffffff		begbuf -1	
		qt	set	b
		pt	set	penkf
		select	set	add
		pci	set	2
f1	ffa780b80be0fcffeffe		setb	
		**	sbi	xj+k
		*		62ijk
f2	ffffffffffffffff		begbuf -1	
		qt	set	x18
		pt	set	penkf
		select	set	add
		pci	set	2
f2	ffa784b80be0fcffeffe		setb	
		**	sbi	xj+bk
		*		63ijk
f3	ffffffffffffffff		begbuf -1	
		qt	set	x18
		pt	set	b
		select	set	add
		pci	set	1
f3	ffa784880be0fcffeffd		setb	
		**	sbi	aj+bk
		*		64ijk
f4	ffffffffffffffff		begbuf -1	
		qt	set	a
		pt	set	b
		select	set	add
		pci	set	1
f4	ffa782880be0fcffeffd		setb	
		**	sbi	aj-bk
		*		65ijk
f5	ffffffffffffffff		begbuf -1	
		qt	set	a
		pt	set	b
		select	set	sub
		pci	set	1
f5	ffa792880be0fcffeffd		setb	
		**	sbi	bj+bk
		*		66ijk
f6	ffffffffffffffff		begbuf -1	
		qt	set	b
		pt	set	b
		select	set	add
		pci	set	1
f6	ffa780880be0fcffeffd		setb	

```

**          sbi  bj-bk          67ijk
*

f7  ffffffff            begbuf -1
      qt      set    b
      pt      set    b
      select  set    sub
      pci     set    1
f7  ffa790880be0fcffefd      setb

**          setx - common actions for -sx- instructions.
*          this macro terminates the microinstruction.
*

      setx    macro
      qrs     set    jfld
      prs     set    kfld
      azbsa   set    zsalu

      azd     set    x
      wrs     set    ifld

      cond    set    validcm
      jmp     set    jpnop
      false   set    fetch
              wait   6
              endm

f8                                org    0c0x+70o

**          sxi  aj+k          70ijk
*

f8  ffffffff            begbuf -1
      qt      set    a
      pt      set    penkf
      select  set    add
      pci     set    2
f8  ffa782b80be0feffeffe      setx

**          sxi  bj+k          71ijk
*

f9  ffffffff            begbuf -1
      qt      set    b
      pt      set    penkf
      select  set    add
      pci     set    2
f9  ffa780b80be0feffeffe      setx

**          sxi  xj+k          72ijk
*

fa  ffffffff            begbuf -1
      qt      set    x18
      pt      set    penkf
      select  set    add

```

fa	ffa784b80be0feffeffe	pci	set	2	
			setx		
		**	sxi	xj+bk	73ijk
		*			
fb	ffffffffffffffff		begbuf	-1	
		qt	set	x18	
		pt	set	b	
		select	set	add	
		pci	set	1	
fb	ffa784880be0feffeffd		setx		
		**	sxi	aj+bk	74ijk
		*			
fc	ffffffffffffffff		begbuf	-1	
		qt	set	a	
		pt	set	b	
		select	set	add	
		pci	set	1	
fc	ffa782880be0feffeffd		setx		
		**	sxi	aj-bk	75ijk
		*			
fd	ffffffffffffffff		begbuf	-1	
		qt	set	a	
		pt	set	b	
		select	set	sub	
		pci	set	1	
fd	ffa792880be0feffeffd		setx		
		**	sxi	bj+bk	76ijk
		*			
fe	ffffffffffffffff		begbuf	-1	
		qt	set	b	
		pt	set	b	
		select	set	add	
		pci	set	1	
fe	ffa780880be0feffeffd		setx		
		**	sxi	bj-bk	77ijk
		*			
ff	ffffffffffffffff		begbuf	-1	
		qt	set	b	
		pt	set	b	
		select	set	sub	
		pci	set	1	
ff	ffa790880be0feffeffd		setx		
	0		errng	intbeg-ofree	*ofree* too big
	b		errng	rsvbeg-free	*free* too big
	b	instfree	equ	intbeg-ofree+rsvbeg-free	

APPENDIX 2. SYMBOL DEFINITION LISTING

This appendix is a listing of the symbol definitions used to set the microcode fields in Appendix 1. Descriptions of the function of each microcode field may be found in §2-§8; for an overview of the microcode fields, see §1.8.

```
true      def    8          true jump address
false     def    8          false jump address

*          define some dummy fields to be used later

select    def    4
qt         def    3
qrs        def    2
pt         def    3
prs        def    2

*          now redefine a 18 bit address field

* addr     equdef false,18

jmp        def    3          jump type

**         jump types

*          all jumps are on condition.

jpnop      equ.    0          jump on next 6 bit compass op
                                biased by 192. (11xxxxxx b)
                                this performs a "jpint" if
                                problem mode and an interrupt
                                is pending.

jpifd0     equ.    1          indexed on instrcn i field
                                biased by 184. (10111xxx b)

jpifd1     equ.    2          indexed on instrcn i field
                                biased by 176. (10110xxx b)

jpint      equ.    3          indexed on interrupt vector
                                biased by 160. (101xxxxx b)

jp         equ.    4          jump to true address from
                                micro code

jpnopni    equ.    5          same as "jpnop" except no
                                interrupts are allowed.

jpifd2     equ.    6          indexed on instrcn i field
                                biased by 168. (10101xxx b)

***        condition types
*****     note true condition is assumed low

cond       def    5
```

```

*          cond=0 is universal true.

unitru    equ    0          universal true

**          bmwxfr - condition for inner transfer loop in
*              bulk memory write.
*
*          bmwxfr = last-word-of-bm-transfer or
*                  last-word-of-record or dbe or sbe or
*                  out-of-cmfl or out-of-bmfl.
*
*          note    bmwxfr = bmrxftr or out-of-bmfl.
*
fastrd    equ    1          use fast or slow BM read
*                          transfer loop for this record
*                          -- use fast loop if CM address
*                          lsb = BM address lsb, and if
*                          not I/O memory, and complete
*                          record

validcm    equ    2          used only if pci 0: FALSE
                          when next instruction is a new
                          word compass binary.

cclldpc0   equ    3          if program counter will not load
                          at next clock.  x register test

cclldpc1   equ    4          if program counter will not load
                          at next clock.  b register test

intrp      equ    5          if an interrupt is occurring

azero      equ    6          if a-bus is all zeros
                          can be used to check for +0 or
                          +_0; depends on ALU function
                          (see zephyr hardware manual)

bigdiff    equ    7          exponent differ by more 128

special    equ    8          input operand(s) to floating
                          add, subtract, multiply,
                          divide, or normalize are
                          special; the floating operation
                          must be performed by software

addnorm    equ    9          add result must be normalized

aexpzer    equ    10         if a-bus exponent is all zeros

fdivlt     equ    11         floating divide algorithm; used
                          to verify/fix answer (checks if
                          bit59 changed through the alu)

bmlrz      equ    12         bmlr = 0

```

lastrec	equ	13	bmlr < 8
dbe	equ	14	double bit error
bmrly	equ	16	Bulk Memory multibus cycle done
b59	equ	17	sign bit of b board z bus: for bulk memory interlocking and divide algorithm
b29	equ	18	bit from BMAR which is used for bulk memory interlocking
memoops	equ	20	dbe, sbe, out of cm/bm fl
bmrpy	equ	21	"you got it" from BM controller
bmxfr	equ	22	finished doing transfers for this BM record -- last transfer in record or "memoops" or end of transfer.
* * *			
mulovl	equ	23	if multiply exponent overflow then take false address
* normovl	equ	24	if normalize exponent overflow then take false address
mon	equ	25	true if in monitor mode
parity	equ	26	true if not(dbe or sbe)
bmrdone	equ	27	true if not(memoop or bmlrz)
bmwrlp	equ	28	finished doing all records in a BM transfer -- "memoops" or BM not ready or end of transfer
* *			
bigsgn	equ	29	sign of larger add/sub operand
bmabort	equ	30	true if bm addr oor
muldone	equ	31	multiply partial products accumulation has completed
*** parcel counter increment (0, 1, 2)			
* pci	def	2	
bzd2	def	1	bit 2**2 of the 3 bit *bzd*
*** number of master clocks for this micro instrctn			
wait	def	4	
*** p register and q register definitions			

```

*
*      note that sign extension is enabled by
*      *pt* or *qt* less than 4.
*

b      equ.    0      select b register r/w
                        (sign extension)
a      equ.    1      select a register r/w
                        (sign extension)
x18    equ.    2      select 18 bit x read
                        (sign extension)
qenjk  equ.    3      select j and k field from
                        instruction register.
                        (only 6 bits, so always pos.)
penkf  equ.    3      select the large k from
                        the instruction register
                        (sign extension)
*
x60    equ.    4      select 60 bit x read
enfp   equ.    5      select the floating point
enone  equ.    6      enable a constant "1"
zeros  equ.    7      enable nothing

qt      equdef dum2,3    q register selections
pt      equdef dum4,3    p register selections

ifld    equ.    0      use instructn i field
jfld    equ.    1      use instructn j field
kfld    equ.    2      use instructn k field
mfld    equ.    3      use microe code field

qrs     equdef dum3,2    q register field selections
prs     equdef dum5,2    p register field selections
wrs     def      2      write register field selects

***      define the z bus source and destination selections
*
a board source selections

zsalu   equ.    0      alu
zsbsh   equ.    2      barrel shifter
zsenz   equ.    5      enable d board (fp) z bus
zbenz   equ.    6      enable zbus from b board

azbsa   def      3      zbus source address for board a

*
b board source selections

zbmar   equ.    0      select the following to be
                        put on the z bus
                        synd 0-7, bmwp 0-2, bmar 00-23
zscmac  equ.    1      put cmac and bmlr on the z bus
zenk    equ.    2      put large k and the cm error
                        address on the z bus
zaddr   equ.    3      put micro-code generated

```

```

                                address on the z bus
zsbmra    equ.    4            bm ra and fl onto the bus
zscmra    equ.    5            cm ra and fl onto the bus
zspc      equ.    6            put the pc onto the bus
*          7 must be left open.
zsrj      equ.    8            return jump
zsjjmp    equ.    9            exchange jump
zsbp      equ.    10           p register (breakpoint)
zscm      equ.    12           read central memory
zsbm      equ.    13           read from bm memory
zaenz     equ.    14           enable zbus from a board

bzbsa     def     4            zbus source address for board b

*          b-board z-bus destinations.

zdcml     equ.    0            write cm
zdbp      equ.    1            write p (breakpoint) register
ldcmac    equ.    2            load cmac
ldmon     equ.    3            load monitor/ioa flip-flops
ldcmra    equ.    4            load cm ra/fl
ldbmra    equ.    5            load bm ra/fl
zdbmlr    equ.    6            load bm length register
*          7 must be unused.

bzd       def     2

*          a-board z-bus destinations.

b         equ.    0            write b registers
a         equ.    1            write a registers
x         equ.    2            write x registers
c         equ.    3            write c registers

azd       def     2
***       device function definitions
*         field called "select" in the microcode

*         alu inputs are called a and b
*         a corresponds to q bus, b corresponds to p bus

*         if the alu is not selected, x register tests
*         are performed.

xfra      equ.    0            transfer a
and       equ.    1            a $and$ b
or        equ.    2            a $or$ b
xor       equ.    3            a $xor$ b
cxfrb     equ.    4            transfer -b
cand      equ.    5            -b $and$ a
cor       equ.    6            -b $or$ a
cxor      equ.    7            -b $xor$ a

add       equ.    8            a + b
sub       equ.    9            a - b
mpadd     equ.    10           a + b for multi-precision adds
*         (use old carry; save new carry)

```

```

mask      equ.   11          form mask

*          shift/mask selection codes

lxjk      equ.   12          left by jk
axjk      equ.   13          right by jk
lxbj      equ.   14          left by bn
axbj      equ.   15          right by bn

*          pack and unpack functions.
*
*          bit definitions - all bits are negative logic.
*          bit 0 = mantissa --> z bus.
*          bit 1 = z bus sign extender.
*          bit 2 = pack exponent --> z bus.
*          bit 3 = unpack exponent --> z bus.

unpkx     equ.   1100b
unpkb     equ.   0111b
pack      equ.   1010b

*          x/t control

setx      equ.   0           set x/t ff to x
ttox      equ.   1           read t write x
xtot      equ.   2           read x write t
sett      equ.   3           set x/t ff to t
addtttox  equ.   8           add t and x, write x

*          floating point control

round      equ.   1           used on ldadd2, ldsub2, ldmul2,
                             divlook, and ldnorm steps to
                             indicate rounding operation
unround    equ.   0           used on ldadd2, ldsub2, ldmul2,
                             divlook, and ldnorm steps to
                             indicate non-rounding operation

nopostn    equ.   1           used in divide algorithm to
                             override multiply postnormalize
postn      equ.   0           allow multiply postnormalize

double     equ.   0           double accumulate multiply
single     equ.   2           singly accumulate multiply

select     equdef dum1,4      shared function selections

* define micro code address selection for read of p
* and q registers

rmfld      def      3

```

```

* define micro code address selection for write of p
* and q registers

wmfld    def    3

*
*      various register enables
*
enprc    equ    0          load parcel register / clear
*          exchange jump flipflop (exit
*          temporary monitor mode)
inccmbm  equ    1          inc cmaddr/bmwp, dec bm lth reg
bmstuff  equ    2          this enables BM select function
enxt     equ    3          load/set x/t control
initcy   equ    4          initialize carry which is used
*          in multiple precision adds
enmmar   equ    5          latch the micro code generated
*          cm address
reqxfr   equ    6          request the bulk memory xfer
qreg0    equ    7          force register 0 on q bus
endxfr   equ    8          end the bulk memory transfer
setxjff  equ    10         set exchange jump flipflop;
*          forces temporary monitor mode
*          during this step and through
*          end of next "clrxjff" step
clrxjff  equ    11         clear exchange jump flipflop
divfix   equ    12         used in floating divide, on a
*          subtract 1 instruction. this
*          increases the magnitude of the
*          mantissa (adds or subtracts 1),
*          and flips bit 47 if mantissa
*          overflows.
ldbmar   equ    13         load bm address register
clrbe    equ    14         clear sbe and dbf flags
*          15 must be unused.

enable   def    4

*
*      memory address buss control
*
cmac      equ    0          cm address counter/register
mar       equ    1          micro code address register
pc        equ    2          program counter (default)
zeroaddr  equ    3          memory address 0

scmad     def    2          memory address bus selection

*
*      program counter functions
*
cldpc1    equ    2          conditional load pc (b-test)
cldpc0    equ    3          conditional load pc (x-test)
ldpc18    equ    4          load pc
ldpc20    equ    5          load pc and parcel counter
bump      equ    6          bump the pc by one
enpci     equ    7          enable parcel counter increment

spc       def    3          program counter selections

```

```

* active low bit: "do memory DBE/SBE checking"

memchk    def    1

* define bulk memory write field

fromcpu   equ.    0           write scratch pad from cpu
fromdram  equ.    1           write scratch pad from dram

bmwrite   def    1

* define values for floating point unit

ldmul1    equ     0   Load first multiply operand into FPU.
ldmul2    equ     1   Load second multiply operand into FPU.
mpy       equ     2   Perform TTL partial products multiply.
omuldp    equ     8   Send DP multiply result to ALU.
omulsp    equ     9   Send SP multiply result to ALU.
ldadd1    equ    10   Load first add/sub operand into FPU.
ldadd2    equ    11   Load second add operand into FPU.
ldsub2    equ    12   Load second subtract operand into FPU.
addshf    equ    13   Send add/sub mantissa and shift count
*           to ALU for high order add word.
addman    equ    14   Send unshifted add/sub mantissa to ALU.
addexp    equ    15   Pack add/sub result (FPU provides the
*           exponent, ALU provides the mantissa).
lddiv1    equ    16   Load first divide operand into FPU.
lddiv2    equ    17   Load second divide operand into FPU.
lditer1   equ    18   Load first operand for divide iteration.
lditer2   equ    19   Load 2nd operand for divide iteration.
odiv      equ    20   Send final packed divide result to ALU.
disblank  equ    21   Disable the mantissa blanking (divide)
oiter     equ    23   Send divide iteration result to ALU.
divlook   equ    24   Load table-lookup value for divide
*           iterations into FPU.
ldnorm    equ    25   Load mantissa and exponent to normalize
*           into FPU.
onms      equ    26   Send normalize shift count and mantissa
*           to ALU.
one       equ    27   Send packed normalize result to FPU; ALU
*           provides shifted mantissa.
enpop     equ    28   Send population count to ALU; ALU
*           provides operand.
addshf1   equ    29   Send add/sub mantissa and shift count
*           to ALU for low order add word.
addmask   equ    30   Send mask count to ALU for computing
*           the low word add/sub mantissa.
packun    equ    31   ALU pack instruction, or FPU not in use.

ffpu      def     5           control for floating point

* define cond4 field

* cond4    def     1
pci        def     2

```

APPENDIX 3. REFERENCES

- [1] NovaNETSM is a development of the University of Illinois.
NovaNETSM is a registered Service Mark of University Communications, Inc.
- [2] The PLATO^R system is a development of the University of Illinois.
PLATO^R is a registered trademark of Control Data Corporation.
- [3] Mark Zvilius, "Microcode to Implement Compass on the Zephyr CPU,"
M.S. thesis, University of Illinois, Urbana, Illinois, 1988.
- [4] Douglas J. Hahn, "An Improved High Speed Memory Swapping Unit for the
Zephyr Mainframe," M.S. thesis, University of Illinois, Urbana, Illinois, 1988.
- [5] Mike Davis, "Overview of NovaNET," CERL document, University of Illinois,
Urbana, Illinois, 1988.
- [6] Fairchild Camera and Instrument Corporation, F100K ECL Databook,
Mountain View, California, 1986.
- [7] Control Data Corporation, Compass Version 3 Reference Manual,
St Paul, Minnesota, 1980.
- [8] Otto Spaniol, Computer Arithmetic, Chichester, Great Britain (John Wiley
and Sons, 1981), pp. 173-191.

INDEX

- 10 ns clock 7, 39
- 15-bit Compass instruction format 42
- 18-bit registers 63
- 30-bit Compass instruction format 42
- 60-bit integer operations 122, 191
- 60-bit registers 63
- 100K ECL series chips 21
- 400100 octal 117
- A Board 6, 63
- A Registers 64
- ABCD field 12, 53, 54
- ABCD FLIP-FLOPS 54
- ABUS 67
- ACCUMULATOR LATCH 91
- acknowledge signal 102, 105
- ACTIVE SCRATCH PAD FLIP-FLOPS 113
- add exponents 85
- ADD/SUBTRACT SELECTION 86, 91, 93
- ADD/SUBTRACT/NORMALIZE EXPONENT PROCESSING 91, 92, 93, 135, 139
- ADD/SUBTRACT/NORMALIZE MANTISSA LATCHES 93, 139
- ADDER (addresses) 47, 49, 50, 56
- ADDEXP signal 79, 80, 86, 91, 92, 93
- ADDMAN signal 79, 86, 91, 93
- ADDMASK signal 79, 86, 92, 93
- ADDRESS FAN-OUT 95, 103, 105, 106
- ADDRESS MUX 102
- ADDRESS/CONTROL MUX 112
- ADDRVAL signal 41
- ADDSHF signal 78, 86, 91, 92, 93
- ADDSHFL signal 79, 86, 91, 92, 93
- AEQB signal 85
- AGTB signal 85, 86
- alternate registers 63, 66, 67, 71
- ALU 6, 49, 63, 68, 70, 71, 120, 122
- ALU and BARREL SHIFTER Inputs 67
- ALU functions 68
- ALU60 macro 122, 148
- AND 18, 122
- AND gate 26
- Anderson-Earle-Goldschmidt-Powers method 140
- AX instruction 122, 180
- AZBSA field 14, 70, 121, 122
- AZD field 14, 66, 121
- B BOARD DATA RECEIVER 103
- B Board 6, 7, 33, 67
- B REGISTER TEST 74, 123
- B Registers 64
- BANK 0 107, 112
- BANK 1 107, 112
- BARREL SHIFTER 6, 63, 67, 68, 70, 71, 122
- BARREL SHIFTER functions 68
- BARREL SHIFTER stages 71
- BBUS 67
- BEGBUF 147
- Bit Mask 70, 71
- Bit Shift 71
- BLANKME signal 81, 86
- BLANKMM signal 81
- BLOCK TRANSFER CONTROL LOGIC 56
- block diagram 2
- block diagram notation 3
- BM 8, 9, 10, 49, 55, 95, 101, 105, 128, 144
- BM ADDRESS CIRCUIT 55
- BM ADDRESS COMPARATOR 56
- BM BANK 0 107
- BM BANK 1 107
- BM Data Board 6, 8, 107, 111, 113, 115
- BM DRIVER 95, 96, 101, 105, 108
- BM error correction 96
- BM instructions 128, 156, 162
- BM interlock 57
- BM reads 108, 113
- BM RECEIVER 95, 109
- BM records 56, 107, 108, 113
- BM REQUEST PROCESSOR 108, 111, 112, 113
- BM transfers 56
- BM writes 108, 113
- BMAR 55, 58, 129
- BMAR/SYN 60, 127
- BMC Board 6, 8, 108
- BMFL field 13
- BMFL LATCH 56
- BMLR 57, 60, 129
- BMLR/CMAC 60
- BMRA field 13
- BMRA LATCH 56
- BMRAFL 60
- BMREAD macro 129, 156
- BMSTUFF signal 129
- BMWP 55, 58, 129
- BMWRITE field 16, 129
- BMWRITE macro 129, 163
- BMXFR signal 59
- boolean instructions 179

breakpoints 50, 58, 127, 144, 174
 Buffered ZBUS (BZB BUS) 66
 buffering 22
 Bulk Memory (see BM)
 bus 26
 BX instructions 122, 179
 Byte Mask 70, 71
 Byte Shift 71
 BZB BUS 63, 66
 BZBSA field 15, 60, 121, 122
 BZD field 15, 49, 50, 122
 C Registers 65, 145
 carry-out 17
 CAS signal 115
 Central Memory (see CM)
 CERL 1
 CIW 42, 121
 CIW fetch 13, 43, 51, 121
 CIW LATCH 43, 45, 51
 CLDPC0 signal 48, 73
 CLDPC1 signal 48, 74
 CLOCK CIRCUIT 37, 51
 CLOCK LATCH 38
 clock signals 39
 CLOCK1 signal 90
 CLOCKX signal 90
 CLOCKY signal 90
 CM 10, 63, 121, 128, 144
 CM ADDRESS circuit 48
 CM ADDRESS COMPARATOR 50
 CM ADDRESS COUNTER 51
 CM ADDRESS MUX 49, 50, 51
 CM address 47, 48, 50, 64
 CM Board 6, 7, 108, 109
 CM ERROR LATCH 49, 60
 CM SPECIAL ADDRESSES 94
 CMAC 48, 49, 121, 122, 129
 CMAD 50
 CMCLK signal 41, 45, 90
 CMFL 13, 48, 49, 51, 60
 CMFL INTERRUPT FLIP-FLOP 51
 CMFL LATCH 49, 50
 CMRA 13, 48, 50, 52, 60
 CMRA LATCH 50
 CMRAFL 60
 CMSPEC signal 54
 column address bits 112
 COMBINE CHECK BITS 105
 COMBINE ERROR CORRECTION BITS 104
 COMPARATOR 49
 COMPASS COMPUTATION circuit 48
 COMPASS INSTRUCTION PIPELINE 42, 48

COMPASS PC COMPUTATION 46
 Compass Instruction Word (see CIW)
 Compass instruction 10, 39, 67, 117
 Compass instruction format 42
 Compass instruction number 35
 Compass PC 7, 12, 44, 46, 122, 123
 Compass simulation 78, 79, 81
 COMPLEMENT AND SIGN EXTEND 6, 67
 COMPLEMENT AND SIGN EXTEND control signals 70
 complement controls 68
 Computer-based Education Research Laboratory (CERL) 1
 COND field 13, 34, 120
 condensor 9
 CONDITION MUX 34, 36, 39, 76
 conditional jump instructions 73, 74, 123, 176, 177
 CONTROL (CM address) 48
 CONTROL MUX 112
 converter chips 90
 converter enables 90
 COUNTER 38, 41, 54, 96
 COUNTER initialization 96
 CPU 108, 113
 CPU memory cycle 115
 CX instruction 133, 207
 D and E BOARD CONTROL 75
 D Board 6, 67, 75
 DATA IN RECEIVER 108
 DATA IN RECEIVER Hardware Details 109
 DATA IN SHIFTER 108
 DATA IN SHIFTER Hardware Details 109
 DATA OUT SHIFTER 109
 DBCLK signal 58
 debugging memory 96
 decoder 3, 24
 decoupling capacitor 90
 detailed board drawings 3
 DF instruction 123, 176
 differential receiver 23
 differential signal 23
 DISBLNK signal 78, 81, 202
 DIVFIX signal 71, 142
 DIVIDE ROM 87
 divide 75, 77, 83, 84, 87, 139, 194
 divide exponents 85
 DIVLOOK signal 77, 83, 84
 DIVMAC macro 194
 DOUB signal 58
 DOUBLE ACCUMULATE FLIP-FLOP 90
 DOUBLE signal 58
 double bit errors 99, 127, 151

- double precision floating add 190
- double precision floating subtract 191
- double precision multiply instruction 135, 193
- double-word transfer 57, 95, 96, 101, 105, 108, 109, 113
- drawings, hardware 2
- DX instructions 136, 139, 190, 191, 193
- DYNAMIC RAM TIMING 112, 114
- dynamic RAM 107, 108, 119, 129
- E Board 6, 75, 87
- ECL chips 21
- ECL TO TTL CONVERTORS 108
- ENABLE field 15, 43, 49, 66, 129
- end request 111
- end-around-carry 17, 71, 138
- ENDBM 161
- ENPOP signal 80, 94
- ENXT signal 66
- EQ instruction 123, 177
- ERROR ADDR/K 60, 127
- ERROR CORRECTION BIT GENERATION 103
- ERROR LATCH 99
- error correction 7, 8, 101, 105, 106
- error correction bits 95, 96, 101, 104, 105, 106, 107
- error correction bits for 16 bits of data 104
- error correction bits for 64 bits of data 98
- error correction, disabling 96
- EVEN BANK 95, 101
- even CM addresses 95
- EXCHANGE JUMP FLIP-FLOP 52
- exchange jump 12, 124, 167
- exchange package 11, 49, 50, 124
- exclusive OR 18
- EXECIO 13, 52, 54
- executing microcode 9
- execution 37, 38
- executor 9, 10
- EXPC signals 84
- EXPONENT COMPARATOR 85, 86
- EXPONENT LATCH A 85
- EXPONENT LATCH B 85
- EXPONENT LATCHES 85
- exponent 18
- FADDFHS signal 70
- FALSE field 13, 34, 49, 120
- fan-out 22, 115
- FASTRD signal 59
- FBUS 72
- FETCH 127, 151
- fetch 36, 43
- FFPU field 15, 76, 83, 129
- flip-flops 25
- floating point 6, 18, 75
- floating point add 78, 82
- floating point add instructions 67, 70, 137, 184
- floating point divide 71, 139
- floating point multiply 75, 76, 82, 135, 192
- floating point normalization 19, 134
- floating point subtract 78, 82, 137, 184
- FORMAT signal 90
- FPU CONTROL DECODER 76, 91
- FPUDC signal 90
- FX instruction 135, 137, 139, 184, 192, 194
- GE instruction 123, 177
- HALT macro 148
- hidden CIW fetches 44, 51
- HIO/LIO LATCH 52
- HIO field 12, 103
- HIO/LIO COMPARATOR 52
- i field 35, 42, 63, 65, 66, 73, 120, 121
- ID instruction 123, 176
- ijk fields 12, 42
- INCBMAR signal 58
- INCCMBM signal 58
- indefinite 20, 76, 81, 92
- integer multiply 75, 76, 82, 135
- integer non-zero 74
- INTERLOCK FLIP-FLOP 57
- internal carries 71
- INTERRUPT BUFFER 106
- INTERRUPT JUMP 35
- INTERRUPT OVERRIDE 35, 36
- Interrupt Vector field 35, 36
- interrupt processing 65, 66
- interrupt signals 106
- interrupts 12, 38, 46, 53, 99, 127, 151
- INTRAP 151
- inverted address 103
- inverted output 23
- IO FLIP-FLOP 41
- IO REQUEST PROCESSOR 101, 102, 103, 105
- IO REQUEST signal 105
- IO.DBE 128
- IO.SBE 127
- IOCM 9, 12, 37, 50, 59, 96, 101
- IOCM addresses 103
- IOCM Board 6, 8, 95
- IOCM RECEIVER 96, 101, 105
- IOREQ signal 55

- IR instruction 123, 176
- iterative divide algorithm 75, 87, 139
- IV field 36
- IX instructions 122, 191
- j field 42, 63, 65, 66, 67, 120, 122
- jk field 67
- JMP field 13, 35, 36, 120, 121
- JP instruction 122, 176
- jump instruction 122, 176
- K field 42, 43, 61, 63, 65, 66, 67, 120, 122, 123
- KCLK signal 39, 41, 47, 48
- KDC signal 90
- KECLK signal 36, 39, 41
- KEXPC01 signal 84
- KEXPC02 signal 84
- KHIGH signal 25
- KIOWAIT signal 41
- KNOPC signal 51
- KPRCD signal 45
- KPRCEN signal 43, 45
- KREFRSH signal 115
- KRIGHT signal 70
- KUP signal 25
- KXIO signal 54
- KXTES signal 71
- latch control 21
- LATCH/BUFFER TO CM BOARD 105
- LDADD1 signal 78, 81, 85, 93
- LDADD2 signal 78, 83, 84, 85, 93
- LDBMAR signal 58
- LDCMAC signal 51
- LDDIV1 signal 77, 81, 84, 85
- LDDIV2 signal 77, 84, 85
- LDITER1 signal 77, 84
- LDMORM signal 93
- LDMUL1 signal 76, 81, 85
- LDMUL2 signal 76, 83, 85, 90
- LDNORM signal 79, 81, 83, 85
- LDSUB2 signal 78, 83, 84, 85, 93
- LED DISPLAY 37, 47
- LIO field 13, 103
- LOAD signal 41
- LOADER Board 33, 37, 38, 41, 48
- LOADER Board decoder 33
- LOADER microcomputer 33
- LOADER PP microcode address 36
- logic levels 25
- logical operations 18
- loop counters 64
- LT instruction 123, 177
- LX instruction 122, 180
- MANTISSA LATCH A 93
- MANTISSA LATCH B 93
- mantissa 18
- MANZERO signal 81, 82, 92, 93
- MAR 48, 49
- MAR LATCH 49
- mask count 71
- mask instruction 122, 207
- masks 71, 75
- MBUS 84
- MBUS COMPLEMENT 84
- MEMCHK field 16
- MEMCHK macro 149
- MEMORY TIMING GENERATOR 102
- memory errors 127
- memory testing 97
- MI instruction 123, 176
- MICROCODE COUNTER 39
- MICROCODE LATCH 3, 34, 37, 38, 39, 41, 43
- MICROCODE MEMORY 33, 35, 37, 38, 41
- microcode 9, 13
- microcode address 34
- microcode execution 9
- microcode fields 9
- microcode sequences 117
- MMAR 60
- MMAR field 16, 49
- MON field 12, 55
- MON FLIP-FLOP 52
- monitor mode 36, 50, 51, 52, 53, 56, 66, 81, 117, 125, 143
- MPY macro 149
- MPY signal 80, 91
- MPYGO signal 76, 80
- MULDONE signal 80
- MULOVL signal 76, 77, 81, 83, 86
- multiplexers 25, 70
- MULTIPLIER ARRAY 76, 77, 80, 87, 90, 136
- multiplies 6, 82, 135, 192
- MULTIPLY AND DIVIDE MANTISSA PROCESSING 7
- MULTIPLY COMPLEMENT 90
- MULTIPLY PARTIAL PRODUCTS ACCUMULATOR 86
- MULTIPLY SEQUENCER 76, 80, 90, 91
- multiply exponents 85
- multiply instructions 135, 192
- multiply post-normalize 86, 90
- MULTIPLY/DIVIDE EXPONENT PROCESSING 86, 92, 136

MULTIPLY/DIVIDE MANTISSA
PROCESSING 87

MUX 1 47
MUX 2 47
MUX 3 47
MX instruction 122, 207
NE instruction 123, 177
negative zero 16, 69, 81
network 1
NEWINST signal 45
Newton's Method 140
NEXTOP field 35, 45
NEXTOP JUMP 35, 36, 38, 45, 117
NG instruction 123, 176
NO instruction 123, 207
non-rounding add 93
non-rounding normalize 79
non-rounding subtract 93
NOR gates 25
normalization 19
NORMALIZE SHIFT COUNT 134
NORMALIZE SHIFT COUNT LATCH 85
normalize 19, 79, 82, 90, 134, 180
normalize exponents 85
normalize shift count 74, 79, 82, 91
NORMOVL signal 80, 82, 92
NovaNET 1
NPC 47, 48, 49
NPRC signal 43
NX instruction 134, 180
NZ instruction 123, 176
ODD BANK 95, 101
ODIV signal 77
OITER signal 77
OMULDP signal 76, 81
OMULSP signal 76, 81
ONE signal 80, 81, 91, 93
one's complement numbers 16
one-bit post-normalize (see post-normalize)
ONMS signal 80, 81, 91, 92, 93
OP field 12, 42
operating system 10, 52, 125
OR 18, 122
OR gates 25
OR instruction 123, 176
overflow 20, 76, 77, 81
overflow detection 86
OVERRIDE signal 50, 55, 56
P registers 50, 58, 61, 127
pack operation 18, 93, 133, 183
PACK/UNPACK 93, 133
PACKUN signal 79, 80, 83, 93

PARCEL REGISTER 35, 42, 43, 45, 67
PARCEL SELECT MUX 35, 43
parcel 42, 46
PARTIAL PRODUCTS ACCUMULATOR 80,
90, 136
partial product 80, 87, 89, 91
partially hidden CIW fetches 44, 51, 120,
121
PBUS 6, 14, 65, 67, 71, 120
PBUS COMPLEMENT AND SIGN EXTEND
71
PC 7, 46, 60, 122, 124
PC ADDER 48
PC LATCH 47, 48
PCI field 15, 47, 120
PCK signal 41, 43
PCMAD 50, 54
Peripheral Port (see PP)
PL instruction 123, 176
PLATO 1
POPULATION COUNT 7, 94, 133
population count instruction 133, 207
POST-NORMALIZE FLIP-FLOP 79, 91
post-normalize 75, 76, 77, 86, 90, 91, 92,
136, 139
power pins 28
PP ADDRESS RECEIVERS 103
PP addresses 102, 103
PP BUS INTERFACE Boards 8, 55, 103,
106
PP DATA RECEIVER 105
PP DATA SHIFTER 105
PP LATCH/DRIVERS 106
PP microcomputers 8, 12, 33, 101, 103,
127
PP OFFSET ADDERS 103
PP OFFSET SWITCHES 103
PP QUARTER WORD SELECT 105
PP1 signal 80
PRC 47
pre-shifts 70
private CM 7, 38, 95, 96
problem mode 52, 125
PROBTRAP 151
program stop instruction 143, 173
PRS field 65, 120
PS instruction 143, 173
PT field 14, 49, 65, 67, 120
PX instruction 133, 183
QBUS 6, 14, 65, 67, 71, 73, 120
QBUS COMPLEMENT AND SIGN EXTEND
71
QREG0 signal 65

QRS field 14, 49, 65, 120
 QT field 14, 49, 65, 67, 120
 quarter word 101, 103, 105
 RE instruction 128, 156
 RECEIVERS 112
 reflections 28
 REFRESH ADDRESS COUNTER 112
 REFRESH REQUEST COUNTER 111
 refresh 112
 refresh memory cycle 115
 refresh request 111
 REGISTER ADDRESS MUX 65
 register hardware details 65
 registers 63, 67
 RELERR signal 60
 request signals 101, 111
 resistors 4, 27, 28, 108
 return jump instruction 123, 155
 RJ instruction 60, 123, 155
 RMFLD 14, 65
 ROUND FLIP-FLOP 83
 ROUND signal 83, 90
 round bit 78, 79, 80, 89, 93
 rounding add 93, 137, 184
 rounding multiply 135, 193
 rounding normalize 79, 134, 182
 rounding subtract 93, 137, 184
 row address bits 112
 run mode 34, 40
 RX instructions 137, 139, 184, 193, 194
 SA instructions 121, 172, 208
 SAVED CARRY FLIP-FLOP 71
 SB instructions 120, 210
 SCMAD field 16, 49, 121, 122
 SCRATCH PAD 113
 SCRATCH PAD LATCHES 108, 115
 SELECT field 15, 49, 66, 67, 70, 72, 76, 77, 78, 79, 80, 83, 94, 120, 121, 129
 SELECT field values 68
 sequence of microcode words 10, 47
 set instructions 117
 SETA macro 121, 208
 SETB macro 120, 210
 SHIFT macro 122, 179
 shift count 67, 71
 shift mask generator 70
 SHIFTER CONTROLS 113
 shifts 69, 70, 75, 122, 180
 sign 18, 85
 sign extension 64
 signal connector 90
 single-bit error 99, 127, 152
 single-word transfer 95, 96, 101, 108, 109
 SINGLE-STEP FLIP-FLOP 38, 41
 single-step mode 40
 sip pins 28
 SOSEXEC 81, 128, 150
 SOSOVL 150
 SPC field 15, 48, 122, 123
 SPECIAL ADDRESS COMPARATOR 53
 SPECIAL CM ADDRESS DECODER 49, 50, 51
 SPECIAL INPUT DETECT 81, 135
 SPECIAL signal 76, 77, 78, 79, 81
 special floating point values 20
 start-up 34, 66, 117, 149
 STBM signal 115
 student bank 10
 subroutines 123
 subtract exponents 85
 swapping 10
 switches 103
 SX instructions 120, 212
 syndrome bits 99
 T Registers 63, 65, 66, 71, 137, 139, 144
 table look-up 87, 140
 TASP 27
 termination 28
 time slice 10
 TIMER 53, 59, 127, 151
 timing skews 38
 TOP 12 BITS MUX 97
 transfer C to X 144, 174
 transfer P to X 144, 174
 transfer T to X 143, 171, 175
 transfer X to C 144, 173
 transfer X to P 144, 174
 transfer X to T 143, 175
 TRAP macro 127, 128, 148
 traps 48, 66, 76, 77, 78, 79, 80, 81, 128, 151
 TRUE ADDRESS MUX 36
 TRUE field 13
 TRUE/FALSE MUX 34
 TTL chips 25, 87
 TTL DYNAMIC MEMORY 107, 111
 TTL TO ECL CONVERTORS 107
 TUTOR 9, 10
 twisted pair 23, 27
 UGOTIT signal 39, 41, 102
 underflow 20, 76, 77, 80, 81
 University of Illinois 1
 unpack 93, 133, 183
 UX instruction 133, 183

VALIDCM signal 45, 120, 121
 voltage levels 25
 WAIT field 13, 35, 37, 39, 41, 44, 147
 WAIT macro 147
 WE instruction 128, 162
 wire listing 27, 31
 wire OR 26
 WLCLK signal 41
 WMFLD field 14, 66
 WRITE signal 102
 WRS field 14, 66, 120
 X REGISTER TEST 71, 73, 79, 81, 83, 85,
 123, 134, 140
 X Registers 63, 64, 65, 66, 67, 71, 73
 X/T FLIP-FLOP 64, 66
 XC instruction 145, 173
 XJ instruction 12, 60, 124, 167
 XOR 18, 122
 XP instruction 144, 174
 XR/S signal 38
 XT instruction 144, 175
 XV.DBE 128
 XV.PMODE 125
 XV.SBE 127
 XV.STOP 143
 XX instructions 144, 145, 171, 174, 175
 ZBUS 6, 14, 47, 49, 120
 ZBUS COMPLEMENT AND SIGN EXTEND
 72
 ZBUS MUX 59, 123, 124, 127
 Zephyr 1, 6
 ZERO signal 83
 zero mantissas 83
 ZPL signal 70
 ZR instruction 123, 176
 ZSALU signal 71
 ZX instruction 134, 182

