```
  SSSSSSSS    PPPPPPPP      EEEEEEEEEE      CCCCCCCC
  SSSSSSSS    PPPPPPPP      EEEEEEEEEE      CCCCCCCC
SS            PP      PP    EE            CC
SS            PP      PP    EE            CC
SS            PP      PP    EE            CC
SS            PP      PP    EE            CC
  SSSSSS      PPPPPPPP      EEEEEEEE      CC
  SSSSSS      PPPPPPPP      EEEEEEEE      CC
        SS    PP            EE            CC
        SS    PP            EE            CC
        SS    PP            EE            CC
        SS    PP            EE            CC
SSSSSSSS      PP            EEEEEEEEEE      CCCCCCCC
SSSSSSSS      PP            EEEEEEEEEE      CCCCCCCC



LL            PPPPPPPP      TTTTTTTTTT                  222222
LL            PPPPPPPP      TTTTTTTTTT                  222222
LL            PP      PP        TT                  22      22
LL            PP      PP        TT                  22      22
LL            PP      PP        TT                          22
LL            PP      PP        TT                          22
LL            PPPPPPPP          TT                      22
LL            PPPPPPPP          TT                      22
LL            PP                TT                    22
LL            PP                TT                    22
LL            PP                TT          ....    22
LL            PP                TT          ....    22
LLLLLLLLLL    PP                TT          ....  2222222222
LLLLLLLLLL    PP                TT          ....  2222222222



*START* Job SPEC Req #703 for EGB    Date  3-Dec-82  1:56:07 Monitor: Rational M
File RM:<MICRO-ARCH.TYPE>SPEC.LPT.2, created:  8-Aug-82 21:15:06
        printed:  3-Dec-82  1:56:07
Job parameters: Request created: 3-Dec-82  1:55:20   Page limit:72   Forms:NORMAL
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:A
```

Functional Specification of the Type Board

DRAFT 2

People exaggerate the things they've never had, they admire types because they have no experience with them.

- George Bernard Shaw

1. Introduction

This document describes the functionality of the Type board of the R1000. The specification defines in detail the microcode and hardware interfaces to the board. The reader is assumed to be familiar with both the R1000 architecture and the specifications of the other boards in the R1000 processor.

The type board is extremely similar to the value board. The similarities are designed in to allow the microcode to have the resources of two 64 bit CPU's operating in parallel. This point can not be stressed enough! TWO 64 BIT CPU'S OPERATING IN PARALLEL. Because of this fact the functionality, hardware, and microcode, for the two boards are extremely similar.

In general the differences are the value board has a zero detector, a multiplier, and a shift mux and the type board doesn't. While the type board has some checking circuitry (privacy and class) and the value board doesn't.

This spec will only explain the sections of the type board that are drastically different from the value board. In most cases these difference are obvious from the differences in the microword.

2. Block Diagram Functional Definition

This section refereces the block diagram of the Type board attached to this document. The functionality of each block in the diagram is discussed in detail in the following sections.

2.1. Register File

Same as the value board except there is no zero detector or multiplier.

2.1.1. Register File Addressing

Same as the last comment.

2.1.2. Control Stack Accelerator

Identical to the value board.

2.2. ALU

The ALU control and operation is exactly the same as the value ALU, except there are no conditional ALU operations. (See the microword specification for the exact ALU control available.)

The random field of the type microword allows the selection of the Q bit (a bit supplied by the value board over the backplane) as the carry-in to the ALU. (See the random field of the microword.) When this random micro-order is not selected the T_ALU field of the type microword determines the carry-in to the ALU.

2.3. Mux

The type Mux determines the source of data for storage into the C address of the register file. The two data paths that the Mux can select are:

1. The unmodified output of the ALU.

2. The Write Data Register (WDR). This option is selected by the hardware when a START WRITE command has been issued and the location being written to resides in the Control Stack Accelerator (see the previous CSA section). The microcode should only select this option when the WDR needs to be saved in the RF as a piece of microstate.

(This mux operates differently than the corresponding mux on the value board.)

2.4. Checker

The checker circuitry on the type board can be divided into three function units.

1. Privacy Checker -- does a first level privacy check on one or two operands.

2. Class Check -- checks class compatability of one or two operands.

3. Of_Kind conditions -- detects special type conditions.

The privacy and class check can cause micro events, and are testable as conditions.

2.4.1. Privacy Checker

The privacy checker is used to check if the operand(s) under test is(are) in the scope of privacy. The check facilities are mostly used for scalars, but additional testable conditions are available to test for structures. The privacy checker has a 32 bit outer_frame_name register that is loadable from bits (0:31) of the B_bus. This register must be reloaded during every context switch (and during some instructions like call and exit). To use the privacy checker the type-links of the control_stack operands(s) must be on the A_bus and/or B_bus. The check is selected from the privacy check field of the microcode.

The privacy checker can perform the following five checks:

1. Bin_eq -- Privacy check for equality and assignment. The operands on both the A and B bus are checked.

2. Bin_op -- Privacy check for a binary operation. The operands on both the A and B bus are checked.

3. A_op -- Privacy check for a unary operation. The operand on the A_bus is checked.

4. B_op -- Privacy check for a unary operation. The operand on the B_bus is checked.

5. Paths_equal -- Binary check for the same paths(name and offset) on both the A and B bus.

Using the following identifiers the privacy checks can be accurately expressed as boolean equations.

```
          o_f = outer_frame_name register(0:31)
       A_name = A_bus(0:31)
       B_name = B_bus(0:31)
       A_path = A_bus(0:31,37:56)
       B_path = B_bus(0:31,37:56)
   A_is_priv = A_bus(34)
   B_is_priv = B_bus(34)
  A_drv_priv = A_bus(35)
  B_drv_priv = B_bus(35)

Paths_same := (A_path = B_path)

A_op := (A_drv_priv) V (A_is_priv)(o_f = A_name)~

B_op := (B_drv_priv) V (B_is_priv)(o_f = B_name)~

Bin_op := (A_op) V (B_op)

Bin_eq := (Bin_op)(Paths_same)~
```

(The last four equations indicate that the first level check fails if the equation is true.)
The privacy checker control logic can enable one of six possible privacy checks during a micro-instruction. These six different privacy checks can generate one of four possible micro events. The following table indicates the correspondence between the privacy check enables and the micro events generated. (See the privacy_check field of the microword for the details on enabling a privacy micro event.)

Privacy Check	Micro Event Generated
Bin_eq	Bin_eq
Bin_op	Bin_op
A_[TOS]_op	[TOS]_op
A_[TOS-1]_op	[TOS-1]_op
B_[TOS]_op	[TOS]_op
B_[TOS-1]_op	[TOS-1]_op

Table 2-1: Privacy Micro Event Generation

(The hardware doesn't check if a operand is [TOS] or [TOS-1]. Therefore there is no guarantee that the correct micro-event is taken if the microcode is incorrect.)

All of the privacy events are early and non-persistent. Since they are early events the instruction that caused the event will be re-executed if the micro event handler returns. To prevent the privacy

event from re-occuring, when the privacy event handler returns, a "pass privacy state" exists in the logic. If the micro handler decides that the operand(s) passes the additional privacy check in the handler, the handler should set the "pass privacy check" state. This state is set by specifying the "pass privacy check" order of the privacy check micro-field. If a privacy micro event is enabled and the pass privacy state is set, the micro event doesn't occur and the pass privacy state is cleared. The pass privacy state can also be cleared from the random field. (NOTE: The pass privacy state has NO effect on the privacy test conditions.) The pass privacy state must be cleared during context switches, but does not need to be restored (the event will occur again). (NOTE: The pass privacy state will not change when the privacy check micro order is a "nop", unless the random field clears the state.)

2.4.2. Class Check

The Class checker is used to compare the of_kind bits of a type_link on the A_bus or B_bus to the other bus and/or to a literal. This check provides a parallel mechanism for ensuring that operand(s) are of the correct or same type for a specific instruction.

The class check hardware is capable of 3 different 7 bit compares. The conditions and events that can be generated are shown in the following table.

Hardware Test	Micro event	Condition
A_bus(57:63)=class_lit	X	X
B_bus(57:63)=class_lit	X	X
A_bus(57:63)=B_bus(57:63)	X	X
A_bus(57:63)=B_bus(57:63)=class_lit	X	X

Table 2-2: Class Checks

Only one class check micro event can be enabled during any particular micro-instruction. The random field of the type microword selects which class micro event is enabled (see the microword specification section). All of the class micro events are early events which prevent the current instruction from completing. (All of the class micro checks cause the SAME class event and branch to the same micro event handler.) Any of the above conditions can also be selected as the currently tested processor condition (see the microword specification section).

(Implementation Note: In an effort to reduce the width of the microword the frame bits of the type microword have been overloaded with five bits of the class lit. The class lit is seven bits. The most significant two bits are a field in the microword. The least significant five bits are overloaded with the frame microbits. If a micro-instruction uses both a frame address and a class check with a class lit, the frame address must be the same as the least significant five bits of the class lit. (For more details see the microword specification section.))

2.4.3. Of_Kind condition

The checker circuitry also provides a special test condition that can be used for "subrange" detection on the "of_kind" encodings. The hardware has 64 patterns programmed into two proms. (The class lit is overloaded to choose the patterns.) The first prom contains a seven bit pattern which is compared to bits (57:63) of the B_bus. The second prom contains a seven bit mask which indicates which bits are to be compared. The output of the comparator is one of the selectable conditions on the type board. The following table lists some of the patterns that are currently included in the test conditions.

PATTERN NUMBER (hex)	PATTERN NAME	BIT PATTERN (57:63)
00	TYPED	XXXXX00
01	IMPORT	XXXX0X0
02	VALUE	XXXX000
03	SCALAR	000X000
04	INDIRECT	XXXX100
05	VALUE REF	0XX0100
06	STRUCTURE	1XX0100
07	SUBINDEXED	11XX100
08	REFERENCE	XXXX010
09 - 1F	unused	
20	STATE_WORD	XXXX001
21	CONTROL_KEY	XXXX101
22	MARK_WORD	XXXX111
23 - 3F	unused	

(The "X" bits are not compared.)

Table 2-3: Of_Kind condition

(NOTE: The checker circuit also provides tests for some of the individual bits on the B_bus. These test conditions are very useful for some type checking and are enumerated in the microword specification section under the condition field.)

2.5. Loop Counter

Identical to the value board.

2.6. Bus Interfaces

The type board bus interfaces are exactly the same as the value board, except the type board connects to the TYPE bus where the value board connects to the VALUE bus. (This section and the following three sections, are identical to the value board spec if "TYPE" is substituted for "VALUE".)

2.6.1. VAL Data Bus

2.6.2. FIU Bus

2.6.3. Address Bus

3. Microword Specification

T_RF_A (6 bits): specify the A address of the register file

ENCODING	NAME	FUNCTION
00xxxx	t_gp	select GP register xxxx
010000	t_tos+0	select current top of control stack
010001	t_tos+1	
010010	spare	
010011	t_reg(loop_counter)	select reg. pointed to by loop counter
010100	random state*	outer_frame_name, pass_privacy bit, loop_counter
010101	spare	
010110	spare	
010111	loop_counter	select output of loop counter
011000	t_tos-8	
011001	t_tos-7	
011010	t_tos-6	
011011	t_tos-5	
011100	t_tos-4	
011101	t_tos-3	
011110	t_tos-2	
011111	t_tos-1	
1xxxxx	t_reg(t_frame,xxxxx)	select register xxxxx in the frame pointed to by t_frame field

T_RF_B (6 bits): specify the B address of the register file.

00xxxx	t_gp
010000	t_tos+0
010001	t_tos+1
010010	spare
010011	t_reg(loop_counter)
010100	t_bot-1
010101	t_bot
010110	type_bus (CSA)
010111	spare
011000	t_tos-8
011001	t_tos-7
011010	t_tos-6
011011	t_tos-5
011100	t_tos-4
011101	t_tos-3
011110	t_tos-2
011111	t_tos-1
1xxxxx	t_reg(t_frame,xxxxx)

T_RF_C (6 bits): specify the C address of the register file

00xxxx	t_gp
010000	t_tos+0
010001	t_tos+1
010010	random state* (write disable to RF) outer_frame_name, pass privacy bit, loop_counter
010011	t_reg(loop_counter)
010100	t_bot-1
010101	t_bot
010110	write disable
010111	loop_counter (write disable to RF)
011000	t_tos-8
011001	t_tos-7
011010	t_tos-6
011011	t_tos-5
011100	t_tos-4
011101	t_tos-3
011110	t_tos-2
011111	t_tos-1
1xxxxx	t_reg(t_frame,xxxxx)

(* The format for the random state is as follows:

outer_frame_name	bits 0:31
pass_privacy_bit	bit 32
zero's	bits 33:53
loop_counter	bits 54:63)

T_FRAME (5 bits): specify one of the 32 possible frames in the RF
(This field is overloaded with five bits of the class literal and five bits of the type Of_Kind condition number.)

xxxxx	frame, class literal (2:6), Of_Kind condition_number(2:6)

T_C_SRC (1 bit): specify which data source gets passed to the C PORT of the RF

0	t_c_fiu	FIU -> C address
1	t_c_mux	MUX -> C address

T_MUX (1 bit): specify the data source that the MUX will pass to the C address

0	t_alu	ALU
1	t_wdr	WDR register

T_ALU (5 bits): specify the ALU function

00000	dec_a	F = A - 1
00001	plus	F = A + B
00010	plus_inc	F = A + B + 1
00011	left_1_a	F = A + A
00100	left_1_inc_a	F = A + A + 1
00101	minus_dec	F = A - B - 1
00110	minus	F = A - B
00111	inc_a	F = A + 1
01000	plus_else_minus	
01001	minus_else_plus	
01010	passA_else_passB	
01011	passB_else_passA	
01100	passA_else_incA	
01101	incA_else_passA	
01110	passA_else_decA	
01111	decA_else_passA	
10000	not_a	F = A~
10001	nand	F = (A and B)~
10010	not_a_or_b	F = A~ or B
10011	ones	F = -1 (2's comp)
10100	nor	F = (A or B)~
10101	not_b	F = B~
10110	xnor	F = (A xor B)~
10111	or_not	F = A or B~
11000	not_a_and_b	F = A~ and B
11001	xor	F = A xor B
11010	pass_b	F = B
11011	or	F = A or B
11100	pass_a	F = A
11101	and_not	F = A and B~
11110	and	F = A and B
11111	zeros	F = 0

T_RAND (4 bits): specify the described random operation

0000	no_op	
0001	inc_loop_cntr	
0010	dec_loop_cntr	
0011	carry_in_Q	Carry_in = Q bit from val
0100	spare	
0101	write outer_frame_name	
0110	set pass privacy state	
0111	spare	
1000	spare	
1001	pass_A_high	pass upper 32 bits of A INPUT to ALU
1010	pass_B_high	pass upper 32 bits of B INPUT to ALU
1011	class_check(A_bus, lit)	
1100	class_check(B_bus, lit)	
1101	class_check(A_bus, B_bus)	
1110	class_check(A_bus, B_bus, lit)	
1111	spare	

T_CLASS_LIT (2 bits): specify the literal that is compared against during most of the class checks (the other five bits of the class literal are overloaded with the frame).

XX The 2 msb's of the class literal.

PRIVACY_CHECK (3 bits): This field enables one of the four privacy privacy micro events.

000	check privacy for equality
001	check privacy for A_bus and B_bus
010	check privacy for [TOS] on the A_bus
011	check privacy for [TOS-1] on the A_bus
100	check privacy for [TOS] on the B_bus
101	check privacy for [TOS-1] on the B_bus
110	nop
111	clear "pass privacy state"

T_CONDS: specify the selected condition to be sent to the sequencer for processing. Selected condition also gets latched on the TYPE board. The condition bit is set TRUE if the equation below is satisfied.

0011000	alu_eq_z	64 bit ALU output = 0
0011001	alu_ne_z	64 bit ALU output /= 0
0011010	a_gt_b or a_ge_b	
0011011	spare	
0011100	loop_cntr_z	loop counter = 0
0011101	spare	
0011110	spare	
0011111	spare	
0100000	alu_co	64 bit alu carry out
0100001	alu_of	64 bit alu overflow
0100010	alu_lt_z	MSB of alu = 1 (ALU out < 0)
0100011	alu_le_z	64 bit ALU output <= 0
0100100	t_last	last cycle's TYPE condition
0100101	spare	
0100110	one	condition bit = 1
0100111	zero	condition bit = 0
0101000	Of_Kind(##)	Of_Kind condition
0101001	class(A,1)	class_check(A_bus, lit)
0101010	class(B,1)	class_check(B_bus, lit)
0101011	class(A,B)	class_check(A_bus, B_bus)
0101100	class(A,B,1)	class_check(A_bus, B_bus, lit)
0101101	privacy(A)	privacy_check(A_bus)
0101110	privacy(B)	privacy_check(B_bus)
0101111	privacy(equal)	privacy_check(equality)
0110000	privacy(A,B)	privacy_check(bin_op)
0110001	privacy(names)	A_bus.path_name = B_bus.path_name
0110010	privacy(paths)	A_bus.path = B_bus.path
0110011	privacy(struc.)	both privacy(A,B) and privacy(paths)
0110100	B_bus(32)	bit 32 of the B_bus
0110101	B_bus(33)	bit 33 of the B_bus
0110110	B_bus(34)	bit 34 of the B_bus
0110111	B_bus(35)	bit 35 of the B_bus

0111000	B_bus(36)	bit 36 of the B_bus
0111001	B_bus(34 or 36)	B_bus(34) OR B_bus(36)
0111010	spare	
0111011	spare	
0111100	spare	
0111101	spare	
0111110	spare	
0111111	spare	

TOTAL NUMBER OF BITS IN THE MICROWORD = 39

4. Microcode Considerations

The following subsections detail microcode constraints, conditions, and events.

4.1. Context Switch Microstate

The microstate that exists on the TYPE board consists of:

1. The Register File. The Control Stack Accelerator (CSA) and general purpose (GP) registers, in general, will need to be saved on every context switch along with some number of scratch pad registers. There is no hardware checking of which RF locations need to be saved as microstate.

2. The Outer_frame_name (in the checker). The outer_frame_name register only needs to be loaded for the incoming tasks. (The outer_frame must be saved on every context switch, but presumably the full outer_frame from the sequencer is written out during the context switch.)

3. The loop counter.

4. The pass privacy bit. Since privacy is an early event, it should be o.k. to NOT save the pass privacy bit during a context switch. Then the bit only needs to be CLEARED during each context switch.

4.2. Conditions

The TYPE board generates 32 testable conditions. The conditions can be divided into three groups L - late, ML - meduim late, and E - early. The early conditions can be used as conditions for conditional branch types, and don't require a hint. The meduim late and late conditions require hints if used with conditional branch types. Only the early or medium late conditions can be used as conditions for

conditional memory references. And believe it or not, every condition can be latched in the microsequencer's latch.

ALU_EQ_Z (L)
This condition is TRUE whenever the 64 bit ALU output equals zero. The ALU carry out and ALU overflow bits are not taken into consideration when generating this condition.

ALU_NE_Z (L)
This condition is TRUE whenever the 64 bit ALU output does not equal zero. The ALU carry out and ALU overflow bits are not taken into consideration when generating this condition.

A_GT_B (L)
This condition is TRUE when the A input of the ALU is greater than the B INPUT. The comparison treats A and B as signed numbers (i.e. negative A is always less than positive B). To generate this condition the ALU must be executing the SUBTRACT instruction.

A_GE_B (L)
This condition is TRUE when the A input of the ALU is greater than or equal to the B INPUT. The comparison treats A and B as signed numbers (i.e. negative A is always less than positive B). To generate this condition the ALU must be executing the SUBTRACT instruction.

ALU_CO (L)
This condition is TRUE when there is a carry out of the most significant bit of the ALU.

ALU_OF (L)
This condition is TRUE when the result of an ALU operation overflows a 64 bit representation.

ALU_LT_Z (L)
This condition is TRUE when the MSB (sign bit) of the ALU = 1. This is equivalent to testing whether the 64 bit ALU output < 0.

ALU_LE_Z (L)
This condition is true whenever the 64 bit ALU output <= 0. This condition is logical or of the ALU_EQ_Z and the ALU_LT_Z conditions.

T_LAST (E)
At the end of every microcycle, the condition that was selected on the TYPE board gets latched on the TYPE

board. During any microcycle this latched condition can be selected as a testable condition. (This condition is available mostly as a diagnostic feature rather than a useful microcode feature. No provisions are made to keep the value of this latch consistent, across events or context switches.)

TRUE (E)
This condition is always true.

FALSE (E)
This condition is always false.

OF_KIND(#) (ML)
This condition is the result of the Of_Kind condition test. The number specified is used to select the pattern to match.

CLASS(A,LIT) (ML)
This condition is true if the class literal is equal to bit (57:63) of the A_bus.

CLASS(B,LIT) (ML)
This condition is true if the class literal is equal to bits (57:63) of the B_bus.

CLASS(A,B) (ML)
This condition is true if bits (57:63) of the A_bus are equal to bits (57:63) of the B_bus.

CLASS(A,B,LIT) (ML)
This condition is true if bits (57:63) of the A_bus are equal to bits (57:63) of the B_bus and equal to the class literal.

PRIVACY_A (ML)
This condition is true if the first order privacy check on the A_bus passes.

PRIVACY_B (ML)
This condition is true if the first order privacy check on the B_bus passes.

PRIVACY_EQ (ML)
This condition is true if the first order privacy for equality check passes.

PRIVACY_A&B (ML)
This condition is true if the first order privacy check for both A and B busses pass.

PRIVACY_NAMES (ML)

This condition is true if bits (0:31) of the A_bus equal bits (0:31) of the B_bus.

PRIVACY_PATHS (ML)
This condition is true if bits (0:31,37:56) of the A_bus equal bits (0:31,37:56) of the B_bus.

PRIVACY_STRUCT. (ML)
This condition is true both PRIVACY_A&B and PRIVACY_PATHS are both true.

B_BUS(32) (ML)
This condition is true if and only if bit 32 of the B_bus is a one.

B_BUS(33) (ML)
This condition is true if and only if bit 33 of the B_bus is a one.

B_BUS(34) (ML)
This condition is true if and only if bit 34 of the B_bus is a one.

B_BUS(35) (ML)
This condition is true if and only if bit 35 of the B_bus is a one.

B_BUS(36) (ML)
This condition is true if and only if bit 36 of the B_bus is a one.

B_BUS(34_OR_36) (ML)
This condition is true if either bit 34 or bit 36 of the B_bus are one.

4.3. Events

The type board can generate five micro events and no macro events. All of the micro events are early and non-persistent, therefore there are no mask bits for the events (they are each enabled by specific micro-orders). The micro events are class_check, Bin_eq, Bin_op, [TOS]_op, and [TOS-1]_op. The micro events are explained in detail in previous sections.

4.4. Microcode Restrictions

The microcode restrictions for the TYPE board are exactly the same as the CSA RESTRICTIONS and FIU RESTRICTIONS from the value spec. Please consult the value spec for the details.

5. Diagnostics

5.1. Philosophy

5.2. Hardware Support

5.3. Stand Alone Testing

5.4. System Integration Testing

5.5. Micro-Diagnostics

6. Hardware Considerations

6.1. Timing Issues

6.1.1. Data Path Timing

6.1.2. Clocking Issues

6.1.3. Potential Problems and Restrictions

6.2. Chip Count and Power Estimates

6.3. System Interconnections

6.3.1. Foreplane

6.3.2. Backplane

6.4. Layout

Table of Contents

List of Tables