```
  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                    333333
LL            PPPPPPPP      TTTTTTTTTT                    333333
LL            PP      PP        TT                      33      33
LL            PP      PP        TT                      33      33
LL            PP      PP        TT                              33
LL            PP      PP        TT                              33
LL            PPPPPPPP          TT                            33
LL            PPPPPPPP          TT                            33
LL            PP                TT                              33
LL            PP                TT                              33
LL            PP                TT            ....      33      33
LL            PP                TT            ....      33      33
LLLLLLLLLL    PP                TT            ....        333333
LLLLLLLLLL    PP                TT            ....        333333


*START* Job SPEC Req #693 for EGB    Date  3-Dec-82  1:52:04 Monitor: Rational M
File RM:<MICRO-ARCH.FIU>SPEC.LPT.3, created:  1-Dec-82 14:51:10
        printed:  3-Dec-82  1:52:04
Job parameters: Request created: 3-Dec-82  1:52:03   Page limit:81   Forms:NORMAL
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:A
```

Specification of the Field Isolation Unit

DRAFT 3

1. Summary

This document is a functional and physical specification of the R1000 Field Isolation Unit (FIU). It is assumed that the reader is familiar with the R1000 architecture and has access to documentation on other parts of the hardware.

Section 2 of this document describes the functionality of the FIU and includes a description of each major block of the FIU block diagram. Section 3 describes the microword, and provides detailed information about the hardware that must be considered when writing microcode for the FIU. Section 4 includes some examples of how to use the FIU. Section 5 discusses diagnostic capabilities provided by the hardware and section 6 is a physical specification of the FIU board.

2. Functional Overview

The basic operations of the FIU are insertion of data into a 128 bit word and extraction of data from a 128 bit word. An extract operation is used to take 0 to 64 bits of the data which is input to the rotator and right align them on the VO_BUS output of the merger. If the number of bits extracted is less than 64, then most significant bits on the VO_BUS would be the data output on the corresponding bits of the MERGE_VMUX. The TI_BUS will be passed to the TO_BUS on all extract operations. An insert operation is used to take data on the RDATA_BUS and insert it into the data on the TI_BUS and the output of the MERGE_VMUX, the result is output on the TO_BUS and VO_BUS.

The fields being manipulated are described by their OFFSET and LENGTH. For an extract, OFFSET is defined as the first bit of the field being extracted in the 128 bit source word, and LENGTH defines the number of bits in the field being extracted. For an insert, OFFSET is the position of the first bit position in the 128 bit destination word where data is to be inserted, and LENGTH is the number of bits in the field being inserted. The numbering convention of bits is that the MSB is bit 0.

The FIU will also be used for BLOCK COPY and APPEND operations, which can be accomplished with a series of INSERT and EXTRACT operations. BLOCK COPY moves multiple words of data from one contiguous block of memory to another, and APPEND is used to build a 64 bit word by appending bits onto the least significant part of a partial word stored in the VAL_ASSEMBLY_REGISTER.

2.1. Rotator

The ROTATOR is a 128 bit rotator used to position data fields for INSERT and EXTRACT operations. For sign extending, the ROTATOR outputs the most significant bit of the field being extracted to the

MERGE_VMUX. The ROTATOR selects which 64 bits of the 128 bit input will participate in the rotate, and then right rotates the selected 64 bits by up to 127 bit positions. The rotate amount is determined from the operation being performed and the OFFSET and LENGTH parameters. For extracts the data is rotated right by -(OFFSET+LENGTH)mod128 which is equivalent to a left rotate of (OFFSET+LENGTH)mod128. For all insert operations the data is rotated right by (OFFSET+LENGTH)mod128.

2.2. Rotator Details

The following discussion describes the implementation of the ROTATOR, which need not concern the microprogrammer who uses the FIU for vanilla inserts and extracts. The ROTATOR is implemented with four bit combinatorial (right) shifters as three "tiers" of rotators: the first tier nibble aligns the data, the second tier does a 16 bit alignment, and the third tier does the final 64 bit alignment. The first tier can be thought of as a 128 bit rotator which can rotate right by 0 to 3 bits. After this first rotation, the 64 bits which will participate in the final rotate are selected. The nibble that contains the first bit of the field being rotated and the 15 nibbles to the right of that (wrapping around through the most significant nibble) are used as the input to the second tier. Before passing through the first tier the first bit of the field being rotated is at OFFSET (for extracts) and 128-LENGTH (for inserts). The sign extractor uses OFFSET to find the sign bit of data being extracted for both INSERT and EXTRACT operations, so the sign bit is not really useful when the INSERT operation is selected.

2.3. Merger

The merger is a 128 bit 2 to 1 multiplexor which merges data from the RDATA_BUS with data coming from the TI_BUS, on the TYPE half, and the MERGE_VMUX on the VAL half. A merge mask is generated to control the select lines of the MERGER using the OFFSET, LENGTH, and operation parameters. These parameters are used to calculate a START_BIT and END_BIT which mark the beginning and end of the field where the RDATA_BUS will be selected on the merger outputs. The following table shows START_BIT and END_BIT definitions for the specified FIU operations.

OPERATION	START_BIT	END_BIT
EXTRACT	128-LENGTH	127
INSERT FIRST WORD	OFFSET	127
INSERT LAST	0	OFFSET+LENGTH-1

```
word            |                        |
                |                        |
INSERT          |     OFFSET             |        OFFSET+LENGTH-1
```

2.4. Merge Data Register

The MERGE DATA REGISTER (MDR) is a 64 bit register which can be loaded with data from the output of the ROTATOR. This is most commonly used for INSERTs, when data which would normally be right aligned on the ROTATOR's input is rotated to the position where it be inserted in to the destination word and loaded into the MDR. In the next cycle the destination word is merged with the rotated data in the MDR.

2.5. Merge Vmux

The MERGE_VMUX selects one of three sources of data to be merged with the ROTATOR output on the VALUE half of the MERGER. The three sources to this mux are the rotator sign bit output, the VI_BUS, and the FI_BUS. If the sign bit is selected as the MERGE_VMUX output, then the sign bit is driven on all 64 bits of the MERGE_VMUX output.

2.6. Type Assembly Register and Value Assembly Register

The TAR and VAR are used for storing intermediate results, and are also used when the timing does not allow the results to be written back out into the register file. Data from the register files on the TYPE and VAL boards does not make it through the FIU and back into the register file in one cycle, so the output of the MERGER would have to be loaded into TAR and VAR.

2.7. Bus interfaces

The FIU interfaces to the TYPE, VALUE, and FIU busses. The sources of these busses are selected by two fields in the microcode : VALUE_AND_TYPE_BUS_EN and FIU_BUS_EN, which are in the FIU part of the microword. These busses include byte parity checking and generation, and a machine check is generated if a parity error is detected on a bus from which the FIU will be receiving data.

2.8. Offset, Length, and Fill Mode Registers

The seven least significant bits of the ADDRESS_BUS or the OFFSET_LITERAL field of the microword can be loaded into the OFFSET REGISTER. The OREG_SRC field of the microword specifies whether the data is loaded from the ADDRESS_BUS or from the OFFSET_LITERAL, and the OFFSET REGISTER is loaded when the LOAD_OREG microorder is specified. The contents of the OFFSET_REGISTER or the OFFSET_LITERAL can be selected as a source for the OFFSET parameter. If the OFFSET REGISTER is selected, the data must have been loaded into the register at least one cycle previous to starting any FIU operations; however, the OFFSET_LITERAL can be specified and an operation using that literal can be performed in the same cycle. On a READ_MAR the contents of the OFFSET REGISTER are returned on the least significant bits of the VI_BUS; however, the RESTORE_MAR and LOAD_MAR have no effect on the loading of the OFFSET REGISTER.

The source of data to the LENGTH REGISTER is selected by microcode to be from the VI bus bits (25:31), from the literal specified in the LENGTH_LITERAL field, or from the TI bus bits (43:48). The LENGTH REGISTER or the LENGTH_LITERAL can be specified as the source for the length parameter. If the LENGTH REGISTER is selected the data must have been loaded into the register at least one cycle previous to starting any FIU operations using this value; however, a literal can be specified and an operation started using that literal can be performed in the same cycle. The LENGTH REGISTER actually contains the value LENGTH-1 and the LENGTH_LITERAL specified should be LENGTH-1. When loading the LENGTH REGISTER from the VI_BUS the value LENGTH should be specified and the hardware subtracts one from this value before laoding it into the LENGTH REGISTER. If a READ_MAR is issued the contents of the LENGTH REGISTER is returned on bits 43:48 of the TI_BUS; however, LOAD_MAR and RESTORE_MAR have no effect on the loading of the LENGTH REGISTER.

The source of data to the FILL MODE REGISTER is selected by microcode to be from the TI bus bit (36) or from the FILL_MODE_LITERAL field. If the FILL MODE REGISTER is selected, the data must have been loaded into the register at least one cycle prior to starting any FIU operations using this value; however a literal can be specified and an operation started using that literal can be performed in the same cycle. When a READ_MAR is issued the contents of the FILL MODE REGISTER is returned on bit 36 of the TI_BUS; however, LOAD_MAR or RESTORE_MAR have no effect on the loading of the FILL MODE REGISTER.

2.9. Conditions

The FIU generates two conditions: CROSS_WORD_FIELD, and OFFSET_BIT0. The CROSS_WORD_FIELD condition is set when the OFFSET and LENGTH registers describe a field that crosses a 128 bit word boundary.

CROSS_WORD_FIELD := (OFFSET + LENGTH) > 128

The OFFSET_BIT0 condition is tested mainly for determining whether the BADBITID generated by the ERCC hardware is in the upper or lower half of the 128 bit word.

Both of these are early conditions which are valid the cycle after loading the OFFSET, LENGTH, and FILL_MODE registers.

2.10. Saved state

When an early event stops occurs the clocks to some registers are stopped so that they can be saved and restored by the event handler. All registers in the FIU, except the microinstruction register, are left unmodified when clocks are stopped.

2.11. Zero Length Fields

Zero length fields can be specified and are handled as a special case by the FIU. The representation of a zero length field on the FIU is a field with a FILL MODE of ZERO FILL and a LENGTH REGISTER contents of 63 (A 64 bit field differs by having a FILL MODE of SIGN EXTEND). If the LENGTH register is loaded from the VI_BUS and the most significant bit of that field is a 1 (i.e. a length of 64) then the FILL MODE REGISTER will be loaded with a zero (SIGN EXTEND) independent of bit 36 of the TI_BUS. For zero length fields the MERGE MASK will select the TI_BUS and MERGE_VMUX output to be output from the MERGER so that inserting zero length fields will not alter the destination word. If extracting a zero length field and the SIGN BIT is selected as the output of the MERGE_VMUX then the VO_BUS output will be ZEROS.

3. Microword Specification

3.1. Microword Format

OFFSET_LITERAL (7bits) - specify a literal for the offset parameter.

MICROWORD_BITS(0:6)

XXXXXXX = OFFSET

LOAD_OFFSET_REG (1 bit) specifies the load control for the offset register

MICROWORD_BIT(20)

0	load OFFSET REGISTER
1	no load

OFFSET_REGISTER_SOURCE (1 bit) specifies the source of data to the OFFSET REGISTER

MICROWORD_BIT(7)

0	ADDRESS_BUS bits ADDR_OFFS(25:31)
1	OFFSET_LITERAL

OFFSET_SOURCE (1 bit) - specifies the source of the offset parameter

MICROWORD_BIT(19)

0	offset = OFFSET REGISTER
1	offset = OFFSET LITERAL

LENGTH_AND_FILL_MODE (7 bits) - specify a literal for the length and fill mode parameters. The most significant bit of this field is the fill mode literal and the least significant 6 bits are a literal that specify length-1. A length of 64 is differentiated from a length of 0 by defining the fill mode bit for 64 to be 0 (which will indicate sign extend) and the fill mode bit for 0 to be 1 (which will indicate zero fill).

MICROWORD_BITS(8:14)

FLLLLLL = FILL MODE||LENGTH-1

F = 0 => SIGN EXTEND
F = 1 => ZERO FILL

LENGTH_AND_FILL_REG_CONTROL (2 bits) Specify the load control for the LENGTH and FILL MODE REGISTERS.

MICROWORD_BITS(16:17)

	LENGTH PART	FILL MODE PART
00	Load VI (25:31)	Load TI (36)
01	Load literal	Load literal
10	Load TI (43:48)	Load TI (36)
11	no load	no load

FILL_MODE_SOURCE (1 bit) specifies the source of the FILL MODE parameter

MICROWORD_BIT(18)

0	FILL MODE = FILL MODE REGISTER
1	FILL MODE = FILL MODE LITERAL

LENGTH_SOURCE (1 bit) specifies the source of the length parameter

MICROWORD_BIT(15)

0	LENGTH = LENGTH REGISTER
1	LENGTH = LENGTH LITERAL

VI_AND_TI_BUS_SOURCES (4 bits) specify the source of the TI and VI busses

MICROWORD_BITS(28:31)

	TI_BUS source	VI_BUS source
0000	TAR	VAR
0001	TAR	VALUE_BUS
0010	TAR	FIU_BUS
0011	TAR	FRAME_ADDRESS
0100	FIU	VAR
0101	FIU	VALUE_BUS
0110	FIU	FIU_BUS
0111	FIU	FRAME_ADDRESS
1000	TYPE_BUS	VAR
1001	TYPE_BUS	VALUE_BUS
1010	TYPE_BUS	FIU_BUS
1011	TYPE_BUS	FRAME_ADDRESS
11XX	MAR	MAR

LOAD_MDR (1 bit) specifies whether or not to load the Merge Data Register

MICROWORD_BIT(23)

0	no load
1	LOAD MDR

OPERATION_SELECT (2 bits) specify the FIU operation, mod 128 of the results for start bit, end bit, and rotate amount are used by the hardware.

MICROWORD_BITS(24:25)

op	merge mask / start bit	end bit	rotate amount
00 EXTRACT	128-LENGTH	127	-(OFFSET+LENGTH)
01 INSERT LAST	0	OFFSET+LENGTH-1	OFFSET+LENGTH
10 INSERT FIRST	OFFSET	127	OFFSET+LENGTH
11 INSERT	OFFSET	OFFSET+LENGTH-1	OFFSET+LENGTH

MERGE_INPUT (1 bit) specifies the source of data to the merger

MICROWORD_BIT(37)

0	ROTATOR OUTPUT
1	MERGE DATA REGISTER

MERGE_VMUX_SELECT (2 bits) specify the output of the merge_vmux

MICROWORD_BITS(26:27)

0X	FILL VALUE
10	VI_BUS
11	FIU_BUS

LOAD_TAR (1 bit) specifies whether or not to load the TAR with TO

MICROWORD_BIT(21)

0	LOAD_TAR
1	no load

LOAD_VAR (1 bit) specifies whether or not to load the VAR with VO

MICROWORD_BIT(22)

0	LOAD_VAR
1	no load

MICROWORD_PARITY (1 bit)

MICROWORD_BIT(38)

Parity is checked on the contents of the microword, and a machine check is generated if a pariity error is detected. The combination of the microword and the parity bit should generate odd parity.

3.2. Miscellaneous microcode restrictions

* When using the FIU bus as a source to the MERGE_VMUX, the FIU_BUS must also be selected as a source to the TI or VI bus so that parity can be checked.

* On an INSERT, if both halfs of the ROTATOR input are not the same then [LENGTH + (OFFSET)mod4] must be <= 64. If any of the following conditions are met then don't worry about trying to figure this one out.

  - If the target field for this data is totally in the TYPE or VAL half of the desination word.

  - If the length of this field is less than 62.

* Data from the register file of the TYPE or VAL board cannot go through the FIU data path and be loaded back into the register file in the same cycle.

4. FIU Examples

4.1. Insert

CYCLE 1 - The OFFSET and LENGTH parameters are determined from a literal specified in this cycle or from a value latched in some previous cycle (it is possible to specify a literal for one and a previously latched value for the other). The 0 to 64 bit field is sent from the register file on the TYPE or VAL board, over the FIU_BUS and driven onto both the TI and VI busses. These bits are rotated right by (OFFSET+LENGTH), and the result is stored in the MDR. The data being inserted does not necessarily need to be driven onto the TI_BUS (see section on miscellaneous microcode restrictions).

```
data source
               +--------+----------------+
               |garbage |     field      |
               +--------+----------------+
destination
                  offset--+
                          V
 +-----------------+----------------+----------------+
 |                 |     field      |                |
 +-----------------+----------------+----------------+
rotator input
 +--------+----------------+--------+----------------+
 |garbage |     field      |garbage |     field      |
 +--------+----------------+--------+----------------+

rotator output (loaded into MDR)

 +--------+--------+-------+
 |eld     |garbage |     fi|
 +--------+--------+-------+
```

CYCLE 2 The output of the MDR is input to the rotator part of the merger on both the VAL and TYPE halves. The word into which the field is inserted, is being driven onto the TI and VI busses. The TI bus drives the unrotated of the merger input on the TYPE half, and the VI bus is selected as the MERGE_VMUX output to drive the unrotated part of the merger input on the VALUE half. The merge mask START BIT is (OFFSET) and the END BIT is (OFFSET+LENGTH-1). The output of the merger is loaded into the TAR and VAR, or the register file (if destination word comes from memory).

```
merger input
(rotated part)
 +--------+--------+-------+--------+--------+-------+
 |eld     |garbage |     fi|eld     |garbage |     fi|
 +--------+--------+-------+--------+--------+-------+
merger input
(unrotated part)
 +-------------------------+-------------------------+
 |         TI              |          VI             |
 +-------------------------+-------------------------+
merge mask

                    ----------------
 ------------------|                |-----------------

merger output
 +-----------------+----------------+----------------+
 |      TI         |     field      |   VI           |
 +-----------------+----------------+----------------+
```

4.2. Cross Word Insert

CYCLE1 - The OFFSET and LENGTH specified have resulted in a CROSS WORD FIELD condition. In this cycle the data to be inserted is driven over the FIU bus and onto both the TI and VI busses, then rotated left by (OFFSET+LENGTH) and stored in the MDR.

```
                          |<-  length    ->|
data source     +--------+----------------+
                |garbage |     field      |
                +--------+----------------+    offset ---+
destination                                              V
                +---------------------------------------+----+
word1           |                                       |  fi|
                +---------------------------------------+----+

                +------------+-------------------------------+
word2           |eld         |                               |
                +------------+-------------------------------+

rotator +--------+----------------+--------+----------------+
input   |garbage |     field      |garbage |     field      |
        +--------+----------------+--------+----------------+

rotator output (loaded into MDR)

 +--------+--------+-------+
 |eld     |garbage |     fi|
 +--------+--------+-------+
```

CYCLE 2 - Word1 is driven onto the TI and VI busses and merged with the MDR. The insert first word mask is used, the type half of word1 is stored in the register file on the TYPE board, and the result of the INSERT is driven onto the FIU bus and stored in the register file on the VAL board.

```
merger input
(rotated part)
        +--------+--------+-------+--------+--------+-------+
        |eld     |garbage |     fi|eld     |garbage |     fi|
        +--------+--------+-------+--------+--------+-------+
merger input
(unrotated part)
        +-------------------------+-------------------------+
word1   |         TI              |          VI             |
        +-------------------------+-------------------------+
insert first                                         ---------
word mask__________________________________________|
```

CYCLE 3 - Word2 is driven onto the TI and VI busses and merged with the MDR. The insert second word mask is used and the result is stored in the TAR and VAR.

```
merger input
(rotated part)
        +--------+--------+-------+--------+--------+-------+
        |eld     |garbage |     fi|eld     |garbage |     fi|
        +--------+--------+-------+--------+--------+-------+
merger input
(unrotated part)
        +-------------------------+-------------------------+
word2   |         TI              |          VI             |
        +-------------------------+-------------------------+
insert  __________
last word        |___________________________________________
mask
```

CYCLE 4 - The result of the first insert operation now stored on the register file are written back to word1.

CYCLE 5 - The result of the second insert operation now stored in the TAR and VAR are written back to word two.

4.3. Extract

CYCLE 1 - The OFFSET, LENGTH, and FILL MODE parameters are specified as literals or selected from previously latched values. The source word is driven onto the TI and VI busses and rotated left by (OFFSET+LENGTH) to right justify the field being extracted. The sign bit is is selected to be output from the MERGE_VMUX so that the extracted field and the sign bit can be merged and either driven out the FIU bus into the register file or stored in the VAR.

```
source word        offset--+
                           V
rotator +-----------------+----------------+----------------+
input   |                 |    field       |                |
        +-----------------+----------------+----------------+
                          |<- length     ->|

rotator         +--------+----------------+
output          |garbage |    field       |
                +--------+----------------+
merger input
(rotated part)
        +--------+----------------+--------+----------------+
        |garbage |     field      |garbage |     field      |
        +--------+----------------+--------+----------------+
merger input
(unrotated part)
        +-------------------------+-------------------------+
        |         TI              |       fill bits         |
        +-------------------------+-------------------------+
extract                                    _________________
mask    ___________________________________]
```

If FILL MODE = ZERO FILL then the fill bits will be all zero"s, and if the FILL MODE = SIGN EXTEND then the fill bits will be the MSB of the extracted field.

4.4. Cross Word Extract

CYCLE 1 - The OFFSET and LENGTH parameters have generated the CROSS WORD FIELD condition. The first source word is input to the rotator on the TI and VI busses and the VALUE half of the word is extracted and stored in the VAR. This is done by specifying a literal offset of 64, and a literal length of 64.

```
                                                 offset ----+
source word 1                                               V
                +-------------------------------------------+-------+
rotator input   |                                           |     fi|
                +-------------------------------------------+-------+

rotator output
                +-----------------+-------+
                |                 |     fi|
                +-----------------+-------+
merger input
(rotated part)
                +-----------------+-------+-----------------+-------+
                |                 |     fi|                 |     fi|
                +-----------------+-------+-----------------+-------+
merger input
(unrotated part)
                +-------------------------+-------------------------+
                |         TI              |      XXXXXXXXX          |
                +-------------------------+-------------------------+

extract                                    ----------------------------
mask            --------------------------|
```

CYCLE 2 - The type half of the second source word is driven onto the TI_BUS and the VAR is driven onto the VI_BUS. The latched OFFSET, LENGTH, and FILL MODE parameters are selected and an EXTRACT is done. The result can be stored in the VAR or driven onto the FIU bus and stored in the register file.

```
rotator input
                +---------+----------------+-----------------+--------+
                |eld      |                |                 |      fi|
                +---------+----------------+-----------------+--------+

rotator         +---------+----------------+
output          |garbage  |    field       |
                +---------+----------------+
merger input
(rotated part)
                +---------+----------------+--------+-----------------+
                |garbage  |     field      |garbage |      field      |
                +---------+----------------+--------+-----------------+
merger input
(unrotated part)
                +---------+----------------+--------------------------+
                |eld      |                |      fill bits           |
                +---------+----------------+--------------------------+
extract                                                ------------------
mask            ------------------------------------|
```

4.5. Using the FIU as a General Purpose Shifter

The following examples show how to use the FIU as a general purpose shifter to shift data on the VAL part of the rotator input. The FIU operation and parameters selected will depend on the direction of the shift and whether the field being shifted is left or right aligned.

4.5.1. Right Aligned data, Right Shift by n bits

This is really a normal extract operation!

```
Rotator Input                  128-length--+
        0                    63  64        v            127
        +-----------------------+---------+----------------+
        |       XXXXXX          |????????|unshifted data   |
        +-----------------------+---------+----------------+

Operation = extract
Offset    = 128-length
Length     = length-n

Rotator output
         64-length-n --+
                       |
        0   n-1        v        63
        +-----+-------+---------+
        |data |XXXXXX?|unshifted|
        +-----+-------+---------+

Merge Mask Start Bit = 128-length+n
Merge Mask End Bit   = 127
```

4.5.2. Left Aligned Data, Shift Right n bits

```
Rotator input
                                  64+length-1 ---+
        0                     63 64              V       127
        +-----------------------+----------------+---------+
        |     XXXXXXXXXXXXX     | unshifted data | ?????? |
        +-----------------------+----------------+---------+
```

```
Operation = Extract
Offset    = 64
Length    = 64-n
```

```
Rotator Output
               n+length-1 ---+
        0     n              V   63
        +----+----------------+---+
        |???X| unshifted data |??|
        +----+----------------+---+
```

```
Merge Mask Start Bit = 128-(64-n) = 64+n
Merge Mask End Bit   = 127
```

4.5.3. Right Aligned Data, Shift Left n Bits

```
Rotator Input                      128-length--+
        0                        63  64        V            127
        +-----------------------+--------+----------------+
        |       XXXXXX          |????????|unshifted data  |
        +-----------------------+--------+----------------+

Operation = Insert
Offset    = 128-length-n *** See section on miscellaneous microcode
                      restrictions -- this will not always work
                      if data is shifted past the TYPE/VAL
                      boundary  ***
Length    = Length

Rotator Output

64-length-n -+
             |      63-n ---+
        0    V              V   63
        +---+----------------+--+
        |???| unshifted data |XX|
        +---+----------------+--+

Marge Mask Start Bit = 128-length-n
Merge Mask End Bit   = 127-n
```

4.5.4. Left Aligned Data, Shift Left n Bits

Rotator input

```
                                  64+length-1 ---+
        0                      63 64             V      127
        +-----------------------+----------------+---------+
        |     XXXXXXXXXXXXX     | unshifted data | ?????? |
        +-----------------------+----------------+---------+
```

```
Operation = Insert
Offset    = 64
Length    = 64-n
```

Rotator Output

```
         length-n --+       +--- 64-n
        0           V       V    63
        +------------+-----+-----+
        |shifted data|?????| XX  |
        +------------+-----+-----+
```

```
Merge Mask Start Bit = 64
Merge Mask End Bit   = 127-n
```

5. Diagnostics

5.1. Diagnostic Hardware Subsystem

The basic hardware elements added to the FIU board for diagnostic purposes are an 8051 microprocessor, a diagnostic finite state machine and scannable registers. The 8051 is a single chip microprocessor with a serial communication line to a master diagnostic porocessor which resides on the I/O ADAPTER board. This communication line is used to transmit diagnostic commands to the slave processors, status information from slave processors to the master, data to be loaded into a scannable register on one of the CPU cards, or data being read from one of these registers. The 8051 transfers diagnostic commands to the diagnostic finite state machine which "personalizes" this command into the actual sequence signals which must be generated in order to execute the diagnostic command on the FIU board. These signals are used for such purposes as controlling a scannable register, disabling or enabling bus drivers, forcing the state of interboard signals, etc. The scannable registers on the board are implemented with 74S194 parts which in normal operation of the FIU will be used in parallel load mode, and in diagnostic mode will be capable of being shifted by selecting one of the shift modes. These scan chains are wired up so that diagnostically scanning a register has the side effect of loading it back into itself.

5.2. Scannable Registers

The microinstruction register of the FIU board is scannable. Using this mechanism, it is possible to stimulate the board the same way as normal microcode can. The output of the microinstruction register is also fed back to the input of the WCS RAM's which can be written under control of the diagnostic FSM, this mechanism is used to load microcode at IPL and also to load sequences of instructions for diagnostic purposes. The WCS RAM's are addressed from the UADR_BUS or under diagnostic control from a counter on the FIU board.

The MERGE DATA REGISTER is also scannable, which allows loading of data into the FIU data path. This scannable register in combination with diagnostic control of the microcode are sufficient to test the entire FIU data path at speed.

The FIU board also contains the MEMORY MONITOR logic. The ADDRESS part and SPACE part of the MAR are also a scannable register (except for the least significant seven bits of the ADDRESS part). The REFRESH COUNTER is used by diagnostic hardware as the address to the WCS and also to count the number of cycles to be run in any sequence of diagnostic instructions.

5.3. Other Diagnostic Hooks

All signals which are sourced by the FIU board and sent to only one other board are made visible to the diagnostic subsystem to aide in FRU isolation. These signals are either driven onto the 8051 diagnostic bus or loaded into a scannable register which can then be scanned into the 8051 in the normal manner.

The FIU board uses a non-volatile RAM for storing board serial numbers and keeping a history of failures that could be useful in isolating board defects and uncovering design or manufacturing problems.

6. Physical Specification

6.1. Printed Circuit Board

The FIU is layed out on a standard RMI 19.5 X 21 inch 8 layer printed circuit board.

6.2. Power Supply Requirements

The FIU board only requires a +5V power supply, and draws a maximum of 56 amps. This current requirement was calculated by taking the worst case (70 amps) and multiplying by 0.8. (See RMI power supply specification).

Table of Contents