DRAFT: Do not distribute

MIT Concurrent VLSI Architecture Memo #82

Massachusetts Institute of Technology
Artificial Intelligence Laboratory

The Design and Implementation of the Memory Unit

Draft Version

Parag Gupta

January 21, 1997

Abstract

This document describes the memory unit architecture. The interfaces, datapath, and control of the memory unit are presented in detail. Also, the RTL for the memory unit is included.

Contents

1 Memory Unit Overview **3**

2 Interfaces **5**

2.1 Overview 5

2.2 Interface With SZ Pipeline Stage 6

2.2.1 From SZ 6

2.2.2 To SZ 10

2.3 Result Interface 10

2.4 Validate/Invalidate Interface 11

2.5 MSW Interface 11

2.5.1 To MSW 12

2.5.2 From MSW 16

2.6 Exception Interface 16

3 MU Datapath **16**

3.1 Module Description 18

3.1.1 Formatter 18

3.1.2 Adder/Subtractor 19

3.1.3 Boolean Unit 19

3.1.4 Mask Generator/Segment Checker 19

3.1.5 Errval Generator 20

3.1.6 Result Mux 20

3.1.7 WBmux 21

3.1.8 Segment Checker Muxes 21

3.1.9 Driver 21

3.1.10 P and N Latches 21

3.2 Timing Issues 21

3.3 Floorplan 21

3.4 Testing 23

4 MU Control **23**

4.1 Constraints 25

4.2 Module Descriptions . . . 25
4.2.1 Control Input Latch . . . 25
4.2.2 Queue Muxes and Latches . . . 26
4.2.3 MSW Driver . . . 26
4.2.4 Integer Mux . . . 27
4.2.5 Destination Control . . . 27
4.2.6 Validation/Invalidation . . . 27
4.2.7 Datapath Control . . . 28
4.2.8 Operation Check . . . 28
4.2.9 Operation Decode . . . 28
4.2.10 Operand Decode . . . 29
4.2.11 Format . . . 29
4.2.12 Errval and Exception Detection and Generation . . . 29
4.2.13 Pipeline Status . . . 29
4.2.14 Pipeline Control . . . 31
4.2.15 Control Path . . . 31
4.2.16 Top Level Control . . . 31
4.3 Testing . . . 31

5 Integration Testing **31**

A Datapath Behavioral RTL **32**

B Control RTL **33**

C Integrated RTL **34**

D MU Constants **35**

E Errvals and Exceptions **36**

1 Memory Unit Overview

The memory unit is one of the three functional units in the execution stage of the M-Machine MAP chip cluster pipeline. It is responsible for executing the memory unit operations as specified in the MAP ISA, found in CVA memo 59. These instructions include load/store operations, address calculation operations, thread management operations, cache-related operations, and some integer operations. These integer operations include addition, subtraction, and boolean operations.

All of the integer operations are single cycle latency operations, whereas all of the other operations are potentially multicycle operations. When the memory unit is given a non-integer operation to execute, it needs to send out a packet of data over the M-Switch to one of the cache banks or the configuration space in the memory system. This packet contains all of the information that the memory system will need in order to carry out the correct operation. However, since there are four clusters, there are potentially four memory units that will want to send out a packet to the same place at the same time. In addition, since the instruction fetch unit also needs to send packets to the memory system, there are even more modules that may want to send packets out at the same time.

Therefore, there is an arbiter associated with the M-Switch that decides which unit gets to send a packet to each cache bank. Before the memory unit can send a packet across the M-Switch, it must send a request signal, along with the destination for the packet, to the arbiter. The arbiter will return a signal indicating if the memory gets to use the M-Switch or not. If the arbiter granted use of the M-Switch, then the memory unit can drive its packet to the memory system in the current cycle, otherwise it has to request M-Switch use again next cycle and cannot start a new instruction next cycle. The memory unit will keep making requests until it can send its packet.

Once the memory unit sends the packet associated with some instruction, it does not need to remember anything about that instruction. Also, a new instruction can then be accepted by the memory unit in the next cycle.

Under this system, it is possible for operations to be sitting in the memory unit for many cycles. However, it is important for certain instructions, such as those associated with the TLB miss handler, to be able to send its packets out to the memory system quickly. If the miss handler is stalled, then a deadlock may be reached since other outstanding memory requests will be waiting for the miss handler to return. To avoid this problem, another path to send out packets to the memory system is used in cluster 1, where the TLB miss handler is located. This second path, which is called the shadow path, has exactly the same functionality as the regular path, but it only accepts instructions from the event thread, whereas the regular path accepts all other instructions. Just like the regular path, the shadow path will send out requests for use of the M-Switch, receive grant signals from the arbiter, and be able to drive a memory packet to the memory system. However, since both the regular and shadow path will share a common line to the M-Switch, only one of the paths will be able to use the M-Switch at a time.

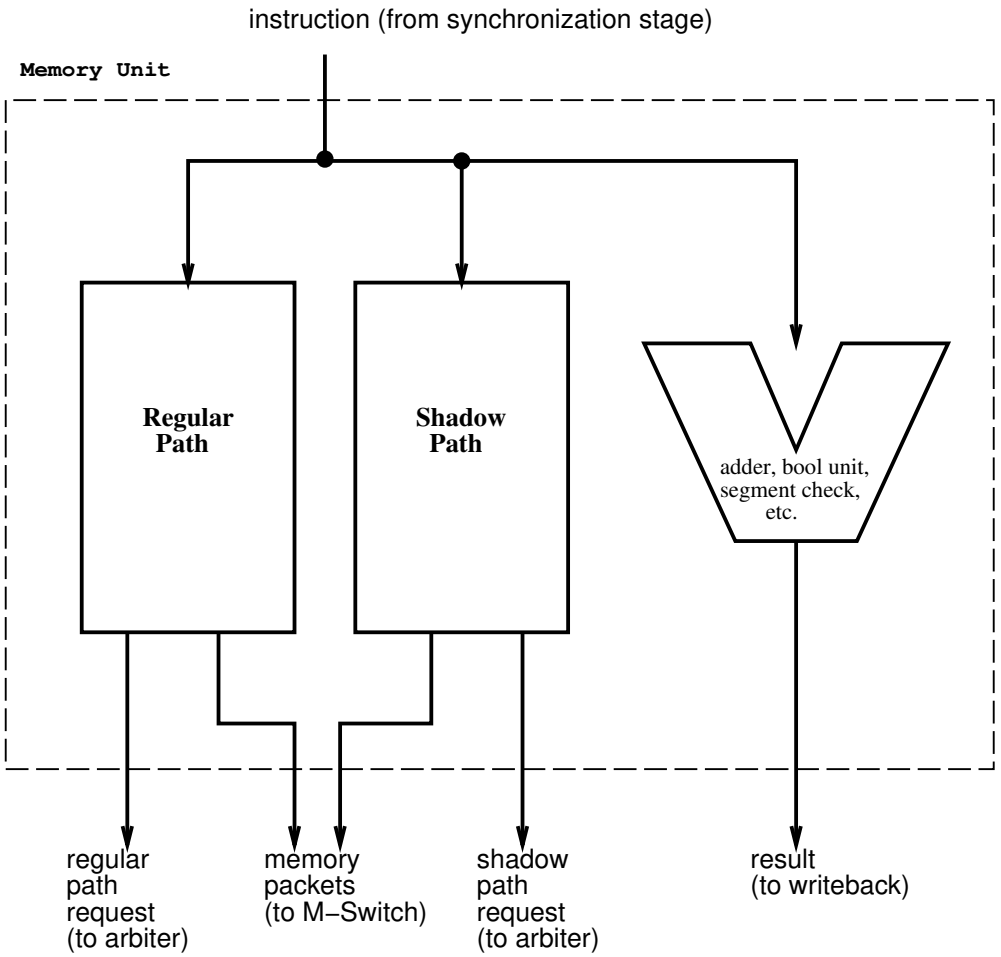

Figure 1: Memory Unit Internal Block Diagram

The actual packet that is driven to a memory system cache bank via the M-Switch is synthesized in the memory unit by a module called the formatter. Based on the instruction, operands, and destination information, the formatter will "format" the packet that needs to be sent across the M-Switch. Both the regular and shadow path have identical formatters.

Though both the regular and shadow paths will contain their own formatters, they will not have their own adders, boolean units, etc. Since the memory unit only handles one instruction at a time and since all instructions that use modules such as adders and boolean units only need those resources for one clock cycle, it is possible for these resources to be shared by both the regular and shadow path.

See figure 1 for a high level illustration of the memory unit organization.

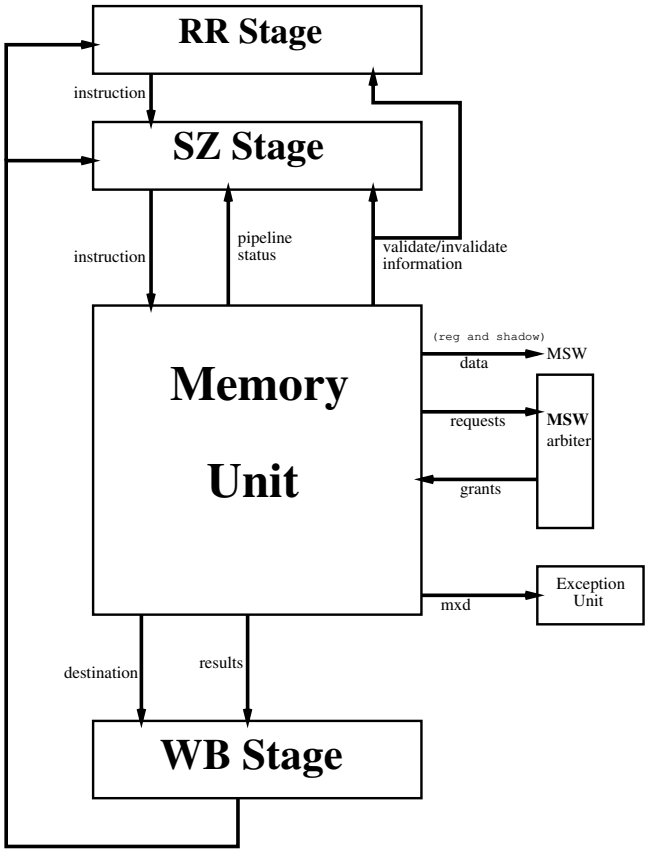

Figure 2: Memory Unit Interface

2 Interfaces

This section presents the interfaces between the memory unit and the modules with which it communicates.

2.1 Overview

The memory unit interfaces with many different modules. See figure 2 for a high level illustration of these interfaces. The following is an overview of these interfaces.

- Synchronization (SZ) Pipeline Stage
  - operations to execute

  - operand values
  - destination information
  - pipeline status

- Writeback Stage
  - operation result
  - destination

- Memory System via M-Switch
  - requests for switch use
  - request grants
  - memory request

- Validate/Invalidate Interface (to both SZ and RR stages)
- Exception Signal

2.2 Interface With SZ Pipeline Stage

The memory unit both receives signals from the synchronization pipeline stage and also sends signals to the synchronization stage. The following sections describe these interfaces.

2.2.1 From SZ

The memory unit receives operation, operands, and destination information from the SZ stage. The following tables list the signals as well as the width of each signal received from the SZ stage. Following the tables is a description of each signal.

Sources			**IP**	**Condition**		**Memory Ops**		
sz_s1_L_v1	sz_s2_L_v1	sz_s3_L_v1	sz_ip_L_1	mOpCnd	mOpCCV	in_mb	preC	postC
66	66	66	62	2	1	1	2	1

Destination					**Operation**				
mDh	mDRF	mDregId	mPincId	mDCCid	mGrp	mType	ill	mVld	vslotId
2	1	4	4	4	5	5	1	1	6

- Sources

- **sz_s1_L_v1**: source 1 for integer operations, address for memory operations (66 bits)
- **sz_s2_L_v1**: source 2 for integer operations, postincrement amount for memory operations (66 bits)
- **sz_s3_L_v1**: data for memory (store) operations (66 bits)

In order for the sources, address, postincrement amount, and data to come into the memory unit as the correct sources, the RR and/or SZ stages may have to move data to the correct operand position from their position as specified in the ISA. For memory operations, the address will always be in **sz_s1_L_v1** and any other non-postincrement operand will be in **sz_s3_L_v1**. For HFORK, the IP for the new thread should be in **sz_s3_L_v1**, while the 2 bits indicating which H-Thread slot to use will be in the low two bits of **sz_s1_L_v1**. For VAC, the value indicating which slot to acquire should be in **sz_s3_L_v1**. For arithmetic operations, the operands will always be in **sz_s1_L_v1** and **sz_s2_L_v1**. For MOV and NOT, the operand will be in **sz_s2_L_v1**.

- IP
  - **sz_ip_L_1**: instruction pointer of current operation (64 bits)

  It has been assumed that the IP will be a pointer, so the pointer bit is not passed to the memory unit. The control logic just needs to be sure that the pointer type is correct for the operation being performed.

- Condition
  - **mOpCnd**: condition for the execution of the operation (2 bits)
  - **mOpCCV**: CC value that must match the condition for execution (1 bit)

mOpCnd	**mOpCCV**	*execute?*
UA	x	yes
UN	x	no
CT	0	no
	1	yes
CF	0	yes
	1	no

- Memory Ops
  - **in_mb**: operation should increment the memory barrier counter (1 bit)
    This signal is used in conjunction with the MSW request signals to determine if the memory barrier counter should be incremented.
  - **preC**: precondition field specifier (2 bits)

  - **postC**: postcondition field specifier (1 bit)
    These conditions are passed on to the memory system for synchronizing memory operations. No processing of these signals is done in the memory unit.

- Destination
  - **mDh**: destination H-thread (2 bits)
  - **mDRF**: destination register file: integer or float (1 bit)
  - **mDregId**: destination register identifier (4 bits)
    A destination of 0 effectively means throw out the result.
  - **mPincId**: destination register identifier of a postincrement operation (4 bits)
    A destination of 0 effectively means throw out the result.
  - **mDCCid**: destination condition code register identifier (4 bits)

  These fields are used to determine where the result of the operations should go. They influence both the result interface signals as well as the MSW request data signals.

- Operation
  - **mGrp**: operation group (5 bits)
    These 5 bits indicate (through one-hot encoding) whether the current operation is an arithmetic op, boolean op, load op, store op, or special op.
  - **mType**: operation type within group (5 bits)

mGrp	**mType**	*Operation*	*Privileged*
boolean	00000	OR	no
00001	00001	AND	no
	00010	NOT	no
	00100	XOR	no
	10000	MOV	no
arithmetic	10100	LEA	no
00010	10101	LEAB	no
	00100	ADD	no
	00101	SUB	no
load	10000	LD	no
01000	11000	LDS	no
	11100	LDSU	no
	11010	LDSCND	no
	10001	LUC	no
store	10000	ST/FST	no
00100	11000	STS/FSTS	no
	00001	SRS/FSRS	no
	11010	STSCND/FSTSCND	no
	11100	STSU/FSTSU	no
special	11001	SSTPRB	yes
10000	11011	HEXIT	no
	11000	HFORK	no
	00101	GETCSTAT	no
	00100	PUTCSTAT	yes
	00010	FLNE	no
	00011	FCA	yes
	01001	VAC	yes

- **ill**: illegal operation (1 bit)
  When this signal is asserted, an illegal instruction exception should take place.
- **mVld**: valid memory operation (1 bit)
  If this signal is not asserted, execute a nop, meaning nothing should be written, validated, invalidated, or requested.

mVld	**ill**	*Result*
0	0	nop
1	0	valid op
x	1	exception

- **vslotId**: one hot encoded V-thread slot identifier (6 bits)
  This signal is encoded as (X,S,D,C,B,A).

2.2.2 To SZ

This section presents the signals that are sent back from the memory unit to the synchronization stage.

Flow Control	
mAvl_L_v1	shAvl_L_v1
1	1

- **mAvl_L_v1**: memory unit regular path availability (1 bit)
- **shAvl_L_v1**: memory unit shadow path availability (1 bit)

  These signals are used by the synchronization stage to decide whether to send an operation to one of the two paths in the memory unit or not. If a path is not available, the synchronization stage will not send an operation to the stage in the next cycle. Note that only cluster 1 actually makes use of both paths.

2.3 Result Interface

This section presents the result interface to the writeback stage. In reality, these signals go to the SZ stage, which in turn propagates these signals to the RR stage in the following cycle.

Result		Destination	
mwb_sync_ptr_v1	mwb_v1	mResRegId_v1	mResVSlotId_v1
2	64	4	6

- Result
  - **mwb_v1**: operation result data value (64 bits)
  - **mwb_sync_ptr_v1**: operation result sync and pointer bits (2 bits)
- Destination
  - **mResRegId_v2**: destination register identifier (4 bits)

– **mResVSlotId_v2**: one hot encoded destination V-thread identifier (6 bits)
This signal is encoded as (X,S,D,C,B,A). If all the bits are 0, then no result is to be written in this cycle. All arithmetic, boolean, and postincrementing operations have a latency of one clock cycle and thus will write a result in the same cycle that they are issued to the memory unit.

2.4 Validate/Invalidate Interface

This section presents the validate/invalidate interface. These signals go to both the RR and SZ stage. They indicate which register to validate or invalidate based on the current operation.

Validate/Invalidate					
mIVSlotId_v1	mIVRFId_v1	mIVRegId_v1	mIsCC_v1	mbInc_v1	mValInv_v1
6	1	4	1	1	1

- **mIVSlotId_v1**: one hot encoded thread identifier (6 bits)
This signal is encoded as (X,S,D,C,B,A). If all the bits are 0, then no validation or invalidation is to be done in this cycle. Invalidation should be done to the condition code destinations of operations which write to condition code registers. The destination of loads (with no ccdest) should be invalidated. Validation should be done to the destinations of arithmetic and boolean operations.
- **mIVRFId_v1**: register file identifier: integer or float (1 bit)
- **mIVRegId_v1**: register identifier (4 bits)
- **mIsCC_v1**: is the target a condition code register (1 bit)
This should never be asserted when a validation is being done.
- **mbInc_v1**: increment the memory barrier for this thread (1 bit)
This is asserted if **in_mb** was asserted and a request was issued.
- **mValInv_v1**: validate or invalidate (1 bit)
If asserted, do a validation, else do an invalidation.

2.5 MSW Interface

This section presents the interface between the memory unit and the M-Switch. The M-Switch is the mechanism by which memory requests reach the memory system. All load,

store, and special operations send some kind of request to the memory system. To do this, the memory unit asserts an M-Switch use request signal. Since four clusters are potentially competing for the same bus, an arbiter on the M-Switch decides which of the requesting units can use the bus. If the memory unit is granted the bus, it sends the request out to the M-Switch. At that point, the memory unit is no longer responsible for what happens to the request that was sent out.

2.5.1 To MSW

This section presents the signals that are sent from the memory unit to the M-Switch. The set of signals in the first table are the signals that the memory unit uses to indicate to the M-Switch that it wants to send a request to the memory system. These signals go to the M-Switch arbiter. The set of signals in the second and third tables below is the actual data found in a memory unit request which goes to the memory system via the M-Switch.

Handshake					
Regular Path			*Shadow Path*		
mReq	mReqCfg	mbank	mShReq	mShReqCfg	mShBank
1	1	2	1	1	2

Memory System Request					
Transfer			*Address*		
mAccType_v2	mpreC_v2	mpostC_v2	mcfg_v2	mphys_v2	mswa_v2
6	2	1	1	1	51

Memory System Request						
Return					*Data*	
mRtnClst_v2	mTslotId_v2	mRtnRf_v2	mRtnRegId_v2	mRtnCC_v2	msync_v2	mswd_v2
2	3	1	4	4	1	65

- Handshake
  - **mReq**: regular path request for use of M-Switch (1 bit)
  - **mReqCfg**: regular path request will go to config space (1 bit)

- **mbank**: cache bank that regular path request will go to (2 bits)
- **mShReq**: shadow path request for use of M-Switch (1 bit)
- **mShReqCfg**: shadow path request will go to config space (1 bit)
- **mShBank**: cache bank that shadow path request will go to (2 bits)

These signals are sent to the M-Switch in the first half of the clock cycle, with a reply coming back in the middle of the second half of the same clock cycle (at the latest).

- Transfer
  - **mAccType_v2**: specifies the memory operation to be performed (6 bits)
    There exists an acctype that effectively tells the memory system to do nothing with the request. This is the abort acctype and can be used if it is decided that a request for use of the M-Switch should not have been issued by the memory unit.
  - **mpreC_v2**: precondition (2 bits)
  - **mpostC_v2** : postcondition (1 bit)
    Both the precondition and postcondition are copied from the inputs **preC** and **postC**.
- Address
  - **mcfg_v2**: configuration space address flag (1 bit)
    This signal is asserted if the address pointer is of type configuration space.
  - **mphys_v2**: physical address flag (1 bit)
    This signal is asserted if the address pointer is of type physical.
  - **mswa_v2**: destination address (51 bits)
    The address is only 51 bits since the three least significant bits of the original 54 bit address are dropped. They are dropped since addresses are byte-addressed.
- Return
  - **mRtnClst_v2**: cluster to return value to (2 bits)
    This is computed by adding the current cluster number (which is hardwired) to **mDh**.
  - **mTslotId_v2**: return V-thread slot (3 bits)
    Note that this is *not* one hot encoded.
  - **mRtnRf_v2**: return register file: integer or float (1 bit)
  - **mRtnRegId_v2**: return register identifier (4 bits)
  - **mRtnCC_v2**: return condition code register (4 bits)
- Data
  - **msync_v2**: data synchronization bit (1 bit)
    This bit is passed from the input and should not be modified.

- **mswd_v2**: data (65 bits)

The following page contains a table indicating how each of the fields should be set for each of the memory operations. Note that when an errval needs to be passed to the memory system, **mReqCfg** is set to the value it would have been set at had there been no need to pass an errval.

	M-SWITCH OUTPUT														
OP	*Handshake*		*Transfer*			*Address*			*Return*					*Data*	
# bits	mReqCfg (1)	mbank (2)	mAccType_v2 (6)	mpreC_v2 (2)	mpostC_v2 (1)	mcfg_v2 (1)	mphys_v2 (1)	mswa_v2 (51)	mRtnClst_v2 (2)	mTslotId_v2 (3)	mRtnRf_v2 (1)	mRtnRegId_v2 (4)	mRtnCC_v2 (4)	msync_v2 (1)	mswd_v2 (65)
ST	cfg req?	mDestV[4:3]	100000	X	X	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	X	mDataV	
FST	cfg req?	mDestV[4:3]	100000	X	X	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	X	mDataV	
STS	cfg req?	mDestV[4:3]	101000	preC	postC	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	X	mDataV	
FSTS	cfg req?	mDestV[4:3]	101000	preC	postC	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	X	mDataV	
STSU	cfg req?	mDestV[4:3]	101100	preC	postC	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	mDCCid	mDataV	
FSTSU	cfg req?	mDestV[4:3]	101100	preC	postC	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	mDCCid	mDataV	
STSCND	cfg req?	mDestV[4:3]	101010	preC	postC	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	mDCCid	mDataV	
FSTSCND	cfg req?	mDestV[4:3]	101010	preC	postC	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	mDCCid	mDataV	
SRS	cfg req?	mDestV[4:3]	100001	X	X	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	X	mDataV	
FSRS	cfg req?	mDestV[4:3]	100001	X	X	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	X	mDataV	
LD	cfg req?	mDestV[4:3]	110000	X	X	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	mDRF	mDregId	X	X	X
LDS	cfg req?	mDestV[4:3]	111000	preC	postC	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	mDRF	mDregId	XX	X	X
LDSU	cfg req?	mDestV[4:3]	111100	preC	postC	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	mDRF	mDregId	mDCCid	X	X
LDSCND	cfg req?	mDestV[4:3]	111010	preC	postC	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	mDRF	mDregId	mDCCid	X	X
LUC	cfg req?	mDestV[4:3]	110001	X	X	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	mDRF	mDregId	X	X	X
SPECIAL OPS (INCLUDING CONFIG SPACE)															
FLNE	0	mDestV[4:3]	000010	X	X	0	mDestV=phys?	mDestV	clst + mDh%4	vslotId	X	X	X	X	X
FCA	0	0	000011	X	X	0	0	X	clst + mDh%4	vslotId	X	X	X	X	X
GETCSTAT	0	mDestV[4:3]	000101	X	X	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	mDRF	mDregId	X	X	X
PUTCSTAT	0	mDestV[4:3]	000100	X	X	mDestV=cfg?	mDestV=phys?	mDestV	clst + mDh%4	vslotId	mDRF	mDregId	X	mDataV	
HFORK	1	X	010000	Fork Dest	X	1	X		clst + mDh%4	vslotId	X	X	mDCCid	mDataV	
HEXIT	1	X	010001	X	X	1	X	X	clst + mDh%4	vslotId	X	X	X	X	X
Pass *errVal*	depends on op	mDestV[4:3]	011111	X	X	0	0	X	clst + mDh%4	vslotId	mDRF	mDregId	X	X	*errVal*
Pass *errVal*with CC	depends on op	mDestV[4:3]	011110	X	X	0	0	X	clst + mDh%4	vslotId	mDRF	mDregId	mDCCid	X	*errVal*
ABORT	X	X	000000	X	X	0	0	X	X	X	X	X	X	X	X

Note: Terms mDataV and mDestV refer to sz_s3_L_v1 and sz_s1_L_v1 respectively.

The phrase "cfg req?" is equivalent to "(mDestV=cfg?) & mDestV[53]".

2.5.2 From MSW

This sections presents the signals sent to the memory unit from the M-Switch.

Flow Control	
mGrnt	mShGrnt
1	1

- **mGrnt**: regular path grant signal from M-Switch arbiter (1 bit)
  This signal indicates whether or not the regular path can send a memory system request over the M-Switch in the current cycle.
- **mShGrnt**: shadow path grant signal from M-Switch arbiter (1 bit)
  This signal indicates whether or not the shadow path can send a memory system request over the M-Switch in the current cycle.

2.6 Exception Interface

There is only one signal involved with exceptions: **mxd_v1**. This one bit signal is asserted by the memory unit if an exception has occurred. In this case, **mwb_v1[63:54]** will indicate what kind of exception occurred, and this value will be sent to register 5 of the exception thread.

3 MU Datapath

The datapath of the memory unit is similar to that of the integer unit, except that there is no shifter, selector, send unit, zero detect, or add mux. See the integer unit datapath document. The memory unit datapath does have a formatter. See figure 3 for a high level block diagram of the memory unit datapath.

There is a path through the datapath for each instruction. The add/sub operations go through the adder and result mux to the output. The boolean operations go through the boolean units and result mux to the output. All of the load, store, and special operations go through the formatter. In addition, some of the load, store, and special operations are postincrementing operations, meaning those operations also use the adder and segment checker as well, as if an LEA was being executed.

LEA involves the adder and the segment checker. Here, an address is added to some immediate to give a new address. Both the old address and new address are sent to the segment checker, along with a mask generated by the mask generator. The mask is used to help check for segment violations.

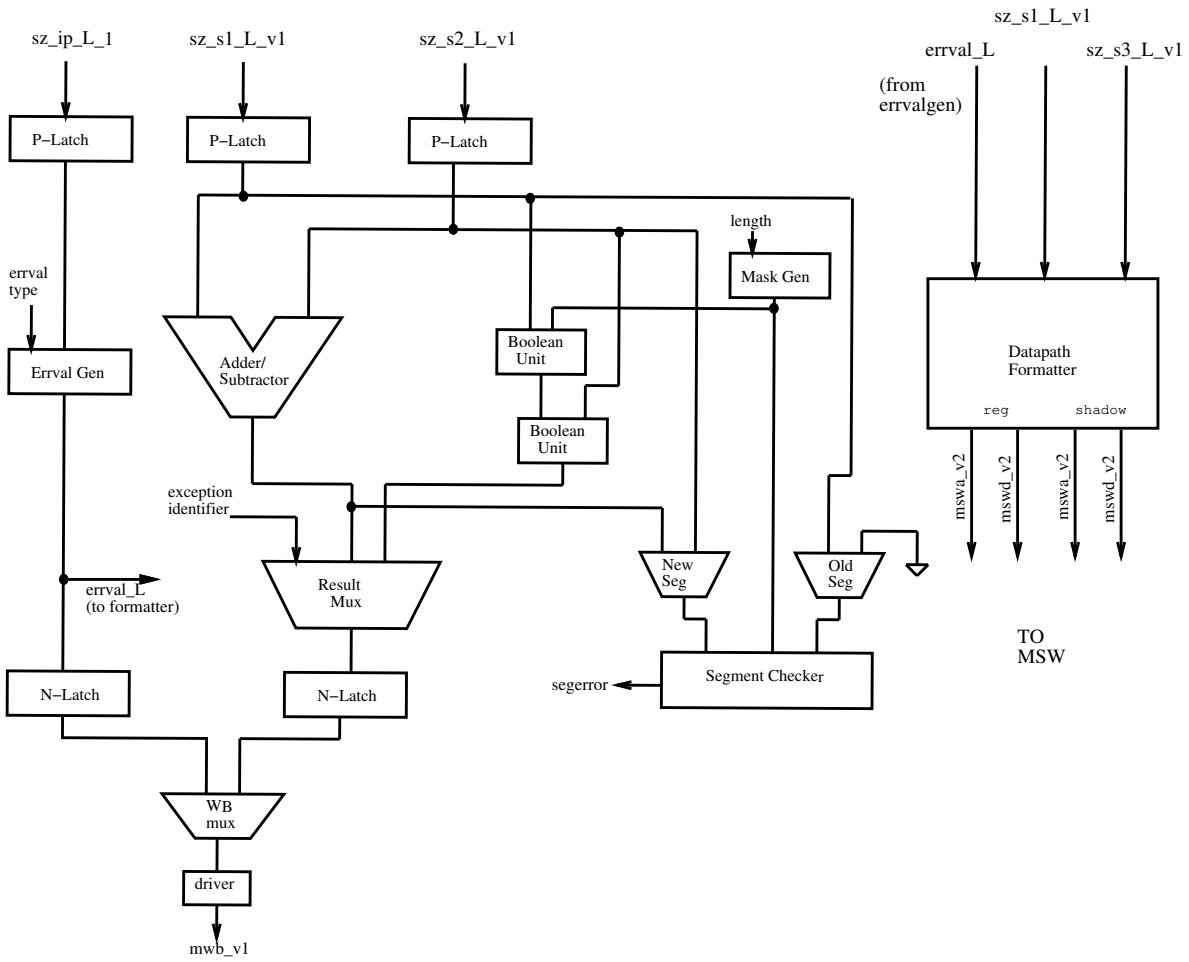

Figure 3: Memory Unit Datapath

LEAB involves the boolean units and the segment checker. In this instruction, an immediate is supposed to be added to the base of the segment in which the address (the first operand) is located. To come up with this new address, the mask is used in conjunction with the original address to generate the segment base. This is done in the first boolean unit. Then this new base is or'd with the immediate in the second boolean unit to yield the new address. A segment check is done to make sure there is no segment violation.

There are also cases where an exception identifier, errval, or an operand need to be passed to the output. The errval is generated at the errval_generator and has a path to the output. The exception identifier is muxed in at the result mux. An operand can be passed to the output through the boolean units.

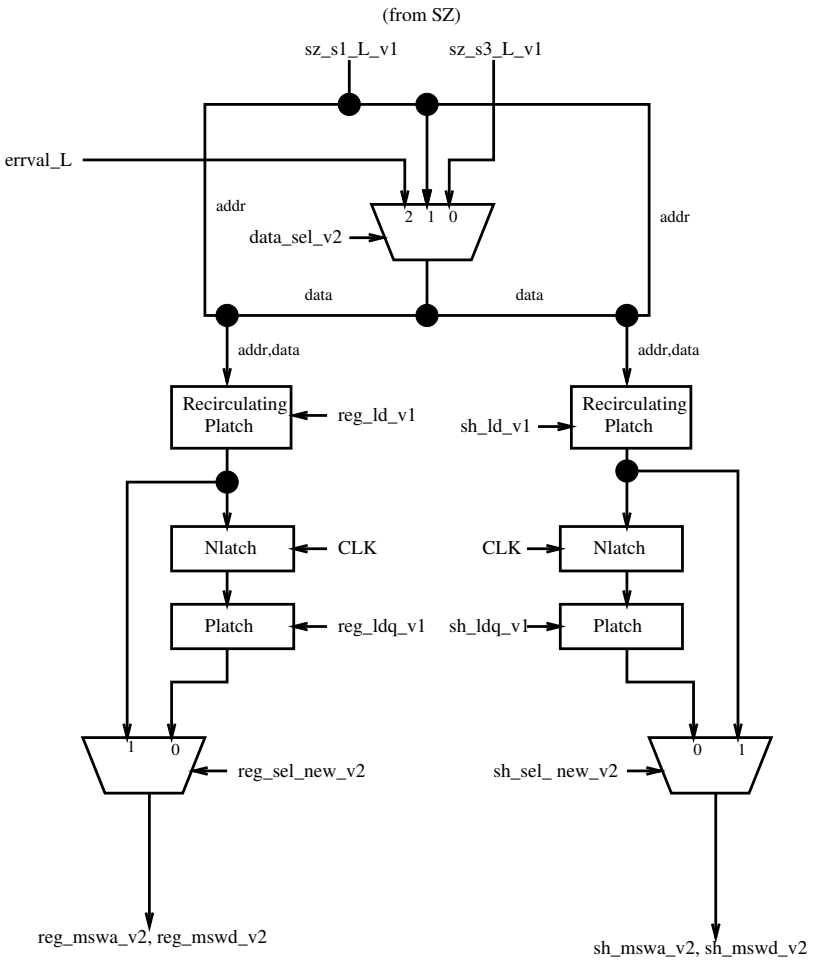

Figure 4: Memory Unit Datapath Formatter

3.1 Module Description

This section describes the function of the various modules within the MU datapath.

3.1.1 Formatter

The formatter's job is to format part of the request packet that will go to the memory system. Since it is possible for a request not to be granted in a particular cycle, the formatter has to be able to hold a request indefinitely. Due to a timing issue with the reception of the grant signals from the M-Switch, the SZ stage will not find out in time for the next cyclewhether or not a path (regular or shadow) is busy or not. Therefore, it is possible that a regular path request (say) is not granted and a new regular path operation is sent on the next cycle. This means the formatter has to be able to hold two requests. The non-formatter part of

the datapath does not have to remember past operations, since anything that goes through there takes one cycle to complete regardless of whether there was a grant or not. See figure 4 for a diagram of the internals of the memory unit datapath formatter. Note that the control signals have been annotated with timing information that reflects the kind of latch driving the signals.

The data inputs to the formatter are **errval_L**, **sz_s1_L_v1**, and **sz_s3_L_v1**. The control inputs include signals to select the proper signal to serve as the data, load signals, queue load signals, mux selects, and drive signals. The input that will serve as data is selected based on whether an errval needs to be passed or not. If no errval needs to be passed, then **sz_s3_L_v1** is selected. If a newly generate errval needs to be passed, then **errval_L** is selected, else **sz_s1_L_v1** is selected. The load signals are asserted to allow new values to enter the formatter. The queue load signals are used when values need to be sent into the second storage spot on each path of the formatter. The mux selects determine which data in the queue is the "live" data. Finally, the drive signals determine which path can send its request to the M-Switch. The outputs include a 51-bit address and a 65-bit data value.

The worst case delay from the input of the mux to the driver output is 2.3 ns.

3.1.2 Adder/Subtractor

The adder is a hybrid carry lookahead / carry select adder that has the capability to do a subtract as well by inverting the second operand and setting the carry input to be one. It works by computing local carries for each pair of operand bits based on carry ins of 0 and 1. In parallel, carry signals for each group of 8 bits is computed. Based on the carry signals for each group of 8 bits, the correct local carries are selected. When these selected carries are xor'd with propagate bits, the sum bits are the result. See the integer unit datapath document for a more detailed description of how the adder works. The data inputs to the adder are two 64-bit operands. The output is the 64 bit result of the addition or subtraction. The only control inputs are **sub** and **carryin**. Both of these signals are asserted in the case when a subtract needs to take place. The worst case delay through the adder is 3.7ns.

3.1.3 Boolean Unit

There are two boolean units in the MU datapath, each of which takes in two 64-bit operands as well as a 5 bit function signal. This function signal indicates what boolean operation should be performed. The output is the 64-bit result of the selected boolean operation. The delay through the boolean unit is .5ns in the worst case. The reason for having two boolean units has to do with the LEAB instruction, as described earlier. Not that the NOT operation inverts the operand coming in as the second source input to the memory unit.

3.1.4 Mask Generator/Segment Checker

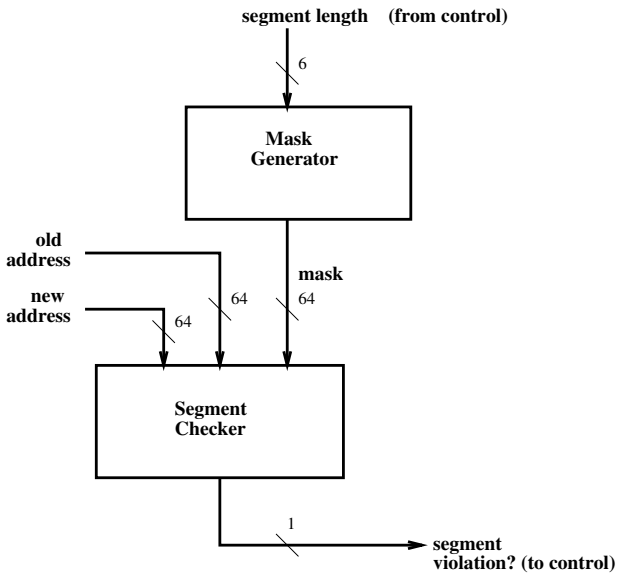

Figure 5: Memory Unit Mask Generator/Segment Checker Interface

The mask generator and segment checker are used together to determine if there has been a segment violation. The mask generator takes in the length of the segment in question (6 bits) and outputs a 64 bit mask. This mask indicates which bits of a new address can change with respect to the old address without causing a segment violation. The segment checker takes in this mask along with the old and new addresses and determines if the new address is in the same segment as the old address through use of the mask. The output of the segment checker is one bit which is asserted if there is a segment violation. The delay through the mask generator is 1.55ns and the delay through the segment checker is 1.75ns. See figure 5 for a block diagram of this interface.

3.1.5 Errval Generator

The errval generator simply takes in the lowest bits of the IP and 10 bits from control that indicate what the nature of the errval is. It outputs these bits as a 64 bit value which may eventually be the output of the memory unit when an errval needs to be generated. The internals of the errval generator just consist of buffers, leading to a delay of 300ps.

3.1.6 Result Mux

The result mux is a 3-input mux that takes in the adder result, boolean unit result, and the exception code. The exception code is actually 10 bits. The remaining 54 bits are grounded. Based on the operation to perform, the correct value to pass on is selected. The output is one of the 64-bit inputs. The delay is about 430ps.

3.1.7 WBmux

The writeback mux selects between the errval and the result mux output. If an errval needs to be passed, then the errval is selected, else the result is selected. The delay through this module is 800ps due to the buffering in this cell.

3.1.8 Segment Checker Muxes

There are two muxes that supply the old and new addresses to the segment checker. Both of these muxes are 2-input muxes that have a worst case propagation delay of 400ps.

3.1.9 Driver

The driver just consists of buffering to drive the writeback line, which has a large capacitive load on it. The delay through the driver is 600ps.

3.1.10 P and N Latches

The P latches have the capability to recirculate their outputs, though this feature is not used in the memory unit. The delay through the P latches is 880ps mostly due to buffering. The delay through the N latches is about 380ps.

3.2 Timing Issues

There are no paths through the MU datapath that are close to not meeting timing constraints. The worst case path is the LEA path which can take up to 8ns to complete, leaving 2ns of time to spare. The only other potential problem may be with the delay from the time a grant signal is received to the time that the memory system receives the request data packet from the memory unit. The delay from reception of a grant signal to the time the request packet reaches the memory system is about 800ps. The original specs indicate this should take 500ps, but this probably is not possible.

3.3 Floorplan

In deciding on a floorplan for the memory unit, the goal was to minimize the number of M3 tracks that would be needed and also to minimize the length of wires running between modules. In running simulations on the MU, wire load was simulated. Estimates for this wire load were based on the floorplan shown in figure 6. The size of each module was estimated, and then the length of the wires could be estimated. The floorplan shown in figure 6 uses up six M3 tracks.

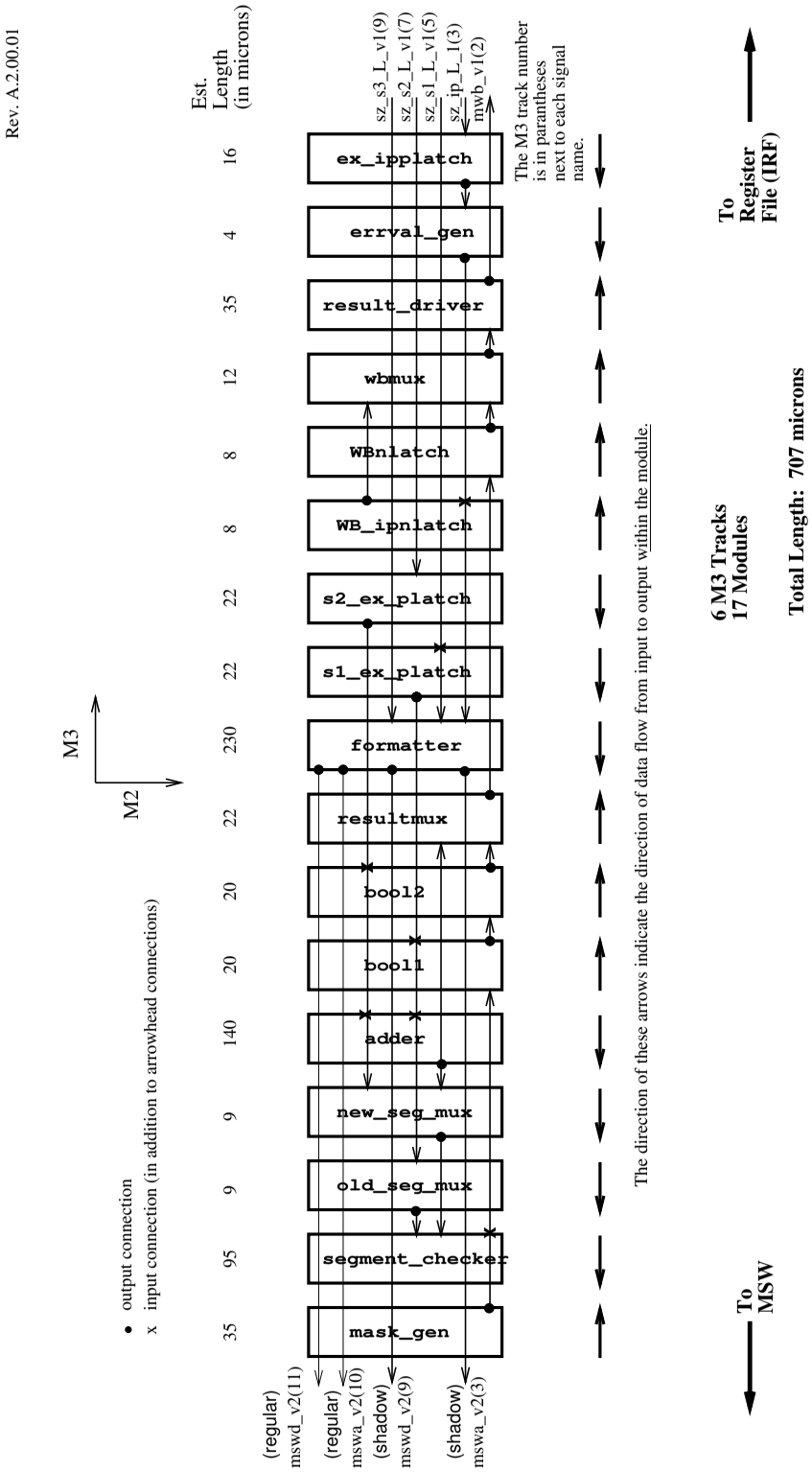

Figure 6: Suggested Memory Unit Floorplan

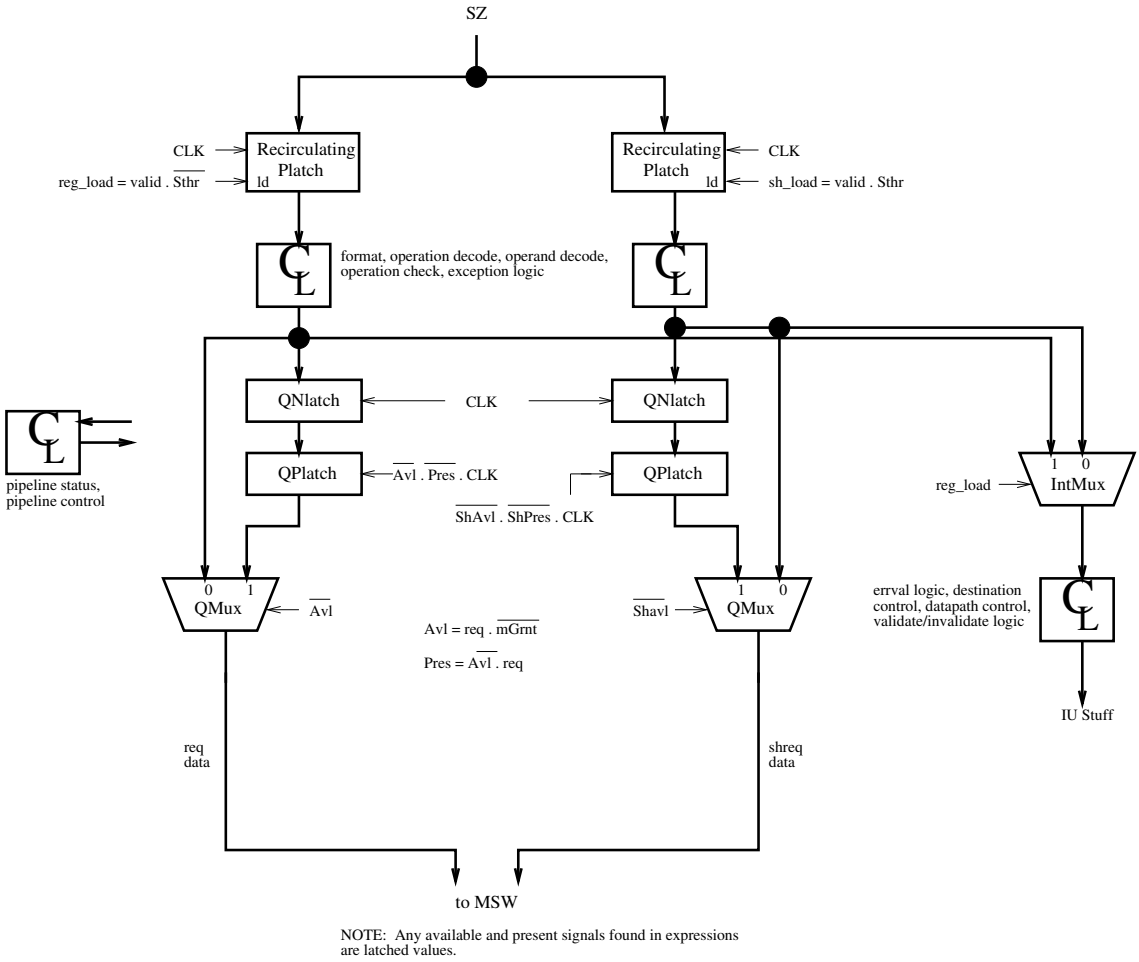

Figure 7: Memory Unit Control Architecture

3.4 Testing

Functional verification of the memory unit datapath was done by running randomized input vectors on an extracted netlist and on a behavioral model of the datapath, making sure that the output of the two models match. See the appendix for the datapath RTL that was used as the behavioral model.

Timing verification was done using Hspice. All paths through the memory unit were simulated. All paths easily meet timing constraints and all rise and fall times are reasonable (less that .5ns).

4 MU Control

The control of the memory unit is mostly straightforward, consisting of mainly boolean

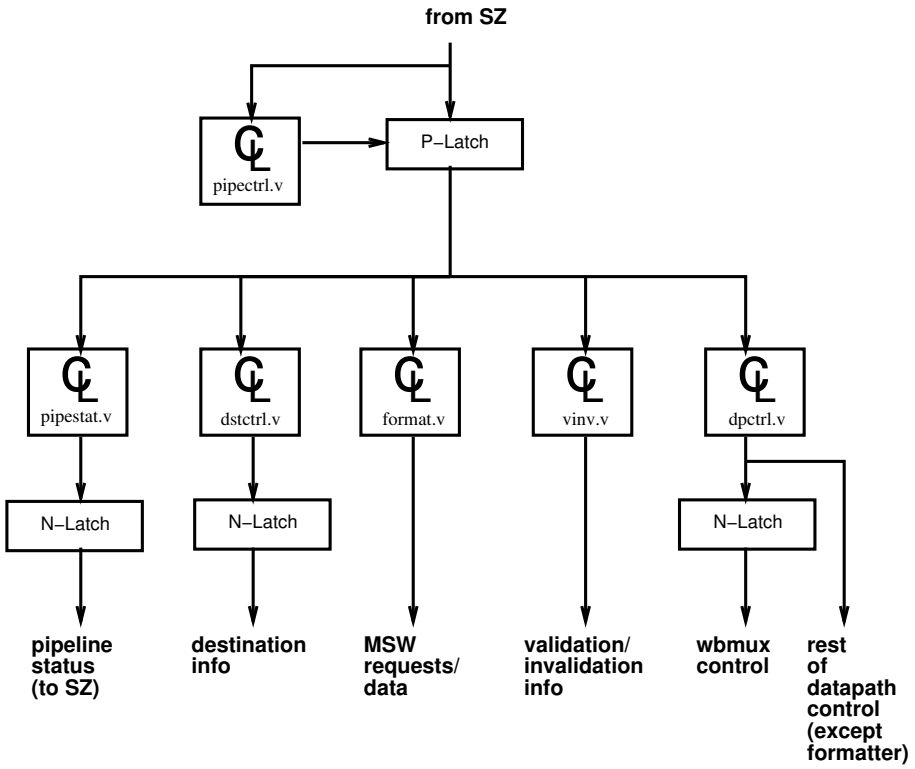

Figure 8: Memory Unit Control Latch Placement

equations. The most difficult part was getting the control for the formatter correct. See figure 7 for a high level view of the memory unit control architecture. See also figure 8 for a diagram illustrating the placement of latches within the memory unit control. Each of the different modules has a particular function.

As the figure shows, there are two paths through the control, one for the regular path and one for the shadow path. Also, there is space in each path to hold two instructions at once, as in the datapath formatter. Some of the modules, such as the operation decode and exception logic, are replicated on both paths, while other modules, such as the datapath and destination control, are shared by both paths. Apart from the two paths, there is a path that goes through a module called INTMUX. The signals that go here are used to control the datapath as well as result information, validation/invalidation information, and errval generation.

The following sections present timing constraints on the control, descriptions of each of the modules that compose the control, and testing methodology.

4.1 Constraints

There are many constraints on the memory unit that affect control. This section describes these constraints. First of all, a request for use of the M-Switch must be issued as early as possible in the first half of the clock cycle. During this time, it must be determined if there are any exceptions or cause for executing nops that would cause the request to be nullified. If there are no exceptions, etc., and the current instruction is a new memory operation or an old ungranted memory operation, then a request for the M-Switch should be issued without nullification.

All exceptions must be detected in the first half of the clock cycle, but since the issuing of requests depends on exceptions being detected in the first quarter of the clock cycle, meeting this constraint is no problem.

Next, the validation/invalidation signals need to be v1 signals. Also, the result control signals must be v2 signals, indicating that these should be driven by an n-latch.

4.2 Module Descriptions

The control unit was designed as a bunch of interconnected modules, each of which has a particular function. The following sections describes the function of each of these modules. The lower level modules are discussed first. Note that MUCTRL is the top level control module, while CTRLPATH is the module that represents a path through the control, where the two paths are the regular and shadow paths. The appendix contains a complete listing of the control RTL.

4.2.1 Control Input Latch

At the interface between the SZ stage and the MU control, a recirculating p-latch is used to latch in the values on each path. A recirculating latch is used since it is possible for instructions to be held in the memory unit over multiple cycles if the instruction is a memory operation that has not been granted used of the M-Switch. This input latch, which is called MU_SZ_PLATCH, takes in all of the control signals which the SZ sends to the memory unit in addition to the pointer and segment information from the operands and IP. It only loads a new instruction if the instruction is from the correct thread, otherwise it holds the current instruction. If reset is asserted, then the MU_SZ_PLATCH outputs all zeros. It is assumed that the SZ stage will not send an instruction to a particular path if the path is not available (due to an instruction waiting for the M-Switch).

The internals of the MU_SZ_PLATCH consist of the base latch cell called MU_SZ_LATCH, which itself consists of a 3-input mux followed by a p-latch with an n-latch feeding back to the mux. See figure 9 for an illustration of this.

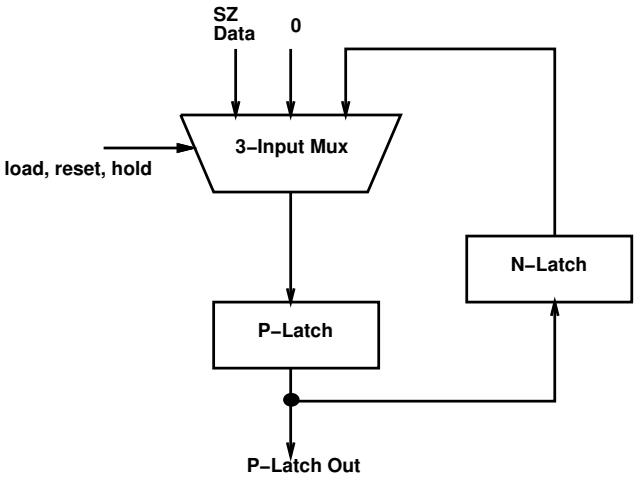

Figure 9: Memory Unit Input Recirculating P-Latch

4.2.2 Queue Muxes and Latches

Within each of the two paths through the memory unit, there exists resources to hold two instructions. An n-latch, p-latch, and 2-input mux are used to implement this, along with the input latch discussed in the previous section.

The n-latch, called M_QNLTCH, just consists of standard cell n-latches. It simply loads its inputs once every clock cycle.

The p-latch, called MU_QPLTCH, just consists of standard cell p-latches. It takes in various control signals associated with issuing a request to the memory system and to the M-Switch. It passes these values to the output when told to load, else it holds its value. Generally, this latch loads if an instruction that was loaded into the memory unit in the previous cycle does not get a grant of a request for use of the M-Switch. That instruction is then held until a grant is issued.

The mux, called MU_QMUX, consists of standard cell 2-input muxes. The inputs to this mux are the two positions in which memory operations can be stored. Depending on where the active instruction is located, the mux will select the proper request to go to the M-Switch drivers.

4.2.3 MSW Driver

Once a request for use of the M-Switch is granted, the actual memory system request has to be driven to the M-Switch. That is the job of MSWDRIVER, which consists of a bunch of tri-state drivers called MSWDRIVE. This tri-state driver is special in that it takes in two data inputs and two drive signals. It is assumed that both drive signals will not be asserted at the same time. The two data inputs are from the regular and shadow paths, and the drive

signals are effectively the grant signals from the M-Switch arbiter. If one of the drive signals is asserted, then the proper input is driven to the output. If neither drive signal is asserted, the output will be at high impedance.

4.2.4 Integer Mux

On any particular cycle, at most one new instruction can be accepted by the memory unit, either on the regular path or the shadow path. Since both paths share the integer part of the datapath (everything except the formatter), only one instruction can have some kind of integer operation taking place at a time. It is the job of INTMUX to pass on the control information for the "current" instruction such that datapath control, destination control, and validation/invalidation will be done for the correct instruction. INTMUX just consists of standard cell 2-input muxes.

4.2.5 Destination Control

This module, called DSTCTRL, is responsible for deciding where the writeback value will go, if anywhere, and what the writeback sync and pointer bit should be. The destination register is set based on the type of operation and on whether there was an exception or not. If there was an exception, the result will go to register 5 of the exception thread. Otherwise, if the current instruction is a postincrement operation, the result goes to **mPincId**, else it goes to **mDregId**. As for the destination thread slot, if the current instruction needs to write back a value and there was no exception, the result goes to **vSlotId**, else if there is an exception, the result goes to the exception thread. Otherwise, no result is to be written back and the thread slot is set to all 0's to indicate this.

The result's sync bit is copied from the sync bit of **sz_s1_L_v1** in the case of a MOV instruction, else the sync bit is copied from **sz_s3_L_v1**. The pointer bit of the result is set to 0 for all arithmetic ops that do not involve a segment check, else the pointer is set to 1. A pointer bit of 0 indicates that the result is an integer.

Finally, all of the outputs are sent through an n-latch so their values do not change until the second half of the next clock cycle.

4.2.6 Validation/Invalidation

This module, called VINV, is responsible for deciding what, if anything, needs to be validated or invalidated. Any operation which places a result in a condition code register should invalidate the CC destination. The destination should be invalidated for non-CC load operations and for getcstat. The only operations that should validate a destination are the single cycle arithmetic ops, boolean ops, and operations which cause an exception, in which case the exception destination should be validated. The exception destination is the exception thread and register 5 of the integer register file.

The slot to be validated or invalidated is given by **vSlotId**, unless there is an exception in which case the target thread is the exception thread. If nothing is to be validated or invalidated, then the destination thread is set to all 0's. The register file ID, given by **mIVRFId** is copied from **mDRF** unless there is an exception in which case the exception identifier should go to the integer register file. **mIsCC** is set to 1 if an operation with a condition code destination is being executed. The register ID to validate/invalidate, given by **mIVRegId**, is set to **mDregId** unless there is an exception or an operation with a CC destination. If there is an exception, the register ID is set to register 5. If the destination is a condition code register, the register to invalidate is given by **mDCCId**. Whether a validation or invalidation is being done is determined by **mValInv**, which is set to 1 if there is an exception or an operation which requires a validation, else it is set to 0. Finally, **mbInc** is asserted if there is no exception or nop, a request for the M-Switch is issued, and **in_mb** is asserted.

4.2.7 Datapath Control

This module, called DPCTRL, is responsible for controlling all of the datapath except the formatter, whose control is handled by PIPECTRL. Implementation of this module was straightforward. Depending on the operation, different signals will be asserted. The result mux will only select the exception input if there is an exception. The boolean output will be selected if there is no exception and if the current instruction is a boolean op, LEAB, or if one of the operands needs to be passed through the memory unit, such as when one of the operands is an errval. The WBmux will select the errval path only when a newly generated errval needs to be passed. The control signal is passed through a n-latch, since the WBmux is located after the n-latch in the memory unit datapath. The boolean operations, subtract signal, old seg select signals, and new seg select signals are set based on the operation.

4.2.8 Operation Check

OPCHECK simply decides if the new instruction is valid or not, based on **mOpCnd** and **mOpCCV** as described in the interfaces section of this document. It also makes sure **mVld** is asserted, otherwise a signal is asserted that indicates that a nop should be executed. This module is replicated on both the regular and shadow paths.

4.2.9 Operation Decode

OPDEC, based on **mGrp** and **mType**, outputs signals indicating what the instruction is and also what class of instructions the new instruction falls in. There is an output signal for each instruction as well as an output signal for classes of instructions, such as memory operations, privileged operations, and postincrement operations. This module is replicated on both the regular and shadow paths.

4.2.10 Operand Decode

OPRNDDEC takes in the pointer and permission bits of the operands and the IP and outputs signals indicating whether these values are pointers, and, if so, what kind of pointers they are. This module is replicated on both the regular and shadow paths.

4.2.11 Format

FORMAT outputs the signals that go to the M-Switch, which includes the request for use of M-Switch signal as well as the actual data signals that go to the memory system when use of the M-Switch is granted. These signals are set according to the table given in the interfaces section of this document. There is one non-obvious thing that is done in this module. A request should not be issued if there is an exception (given by the signal called **reqok**), a nop is being executed (given by **execnop**), or if the operand for FLNE is of the wrong type. This module is replicated on both the regular and shadow paths.

An alternate way of generating the MSW request signal is possible that would allow the request signal to come out faster. However, the chances go up of having to nullify the request once the request has been sent. Instead of checking for exceptions before a request is sent out, a request can be sent out based solely on the instruction to execute and the validity of the instruction, as detected by OPCHECK. If it is determined that an exception has occurred, then **mAccType_v2** can be set such that the request will be nullified.

4.2.12 Errval and Exception Detection and Generation

ERRVAL and EXCPT detect conditions where an errval or exception should be generated and then synthesizes the errval or exception code that should be sent out to the datapath. See the appendix for a listing of all of the possible causes of errvals and exceptions for each instruction. ERRVAL outputs signals indicating whether either of the sources should be passed through the datapath, whether an errval should be passed out of the datapath, and the actual errval code that goes to the errval generator in the datapath.

EXCPT outputs signals indicating if there was an exception, whether to select the errval, data, or the address as data input to the datapath formatter, and the exception code that will go to the exception thread in the case of an exception. If there is a memory operation which requires that the errval given by **sz_s1_L_v1** be passed to the data output of the formatter, **s1_errval** is signalled. If a newly generated errval needs to be passed to the memory system, then **sel_err_to_msw** is selected. Otherwise, **pass_s3** is asserted. Though these control signals have nothing to do with exceptions, they are in the EXCPT module since they need to be generated by both paths and EXCPT is replicated on both paths. ERRVAL is not replicated twice.

4.2.13 Pipeline Status

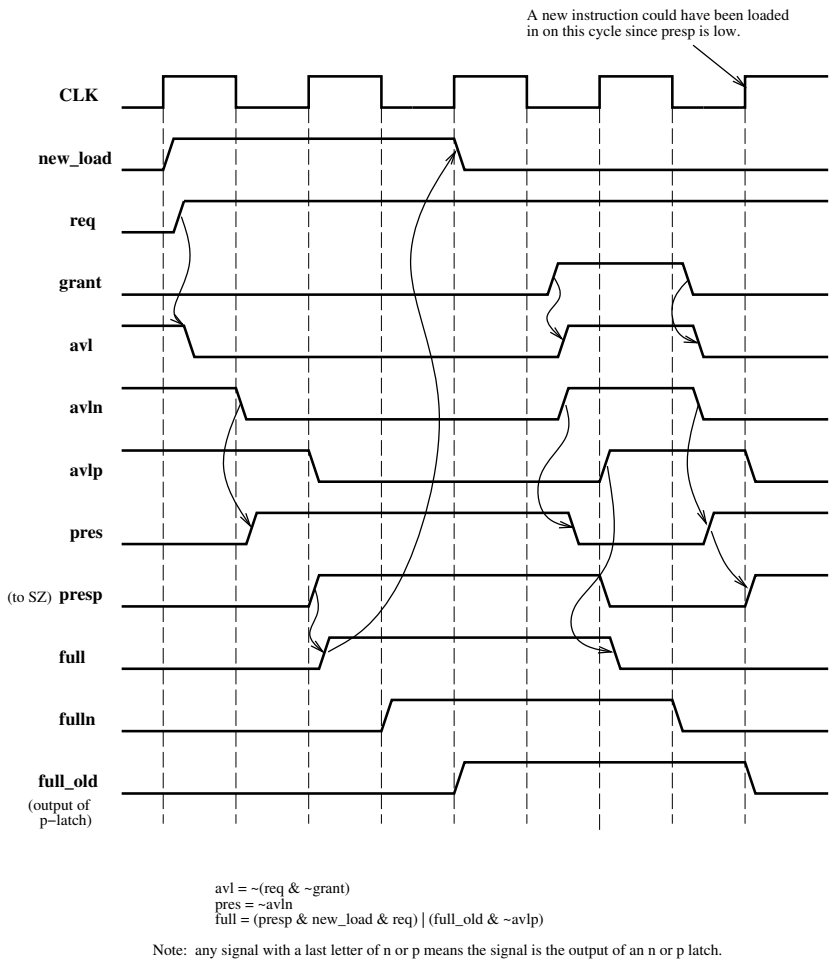

Figure 10: Memory Unit Pipeline Status Timing

This module, called PIPESTAT, along with PIPECTRL, is the most difficult part of the memory unit control. PIPESTAT is responsible for maintaining the status of the memory unit pipeline. It generates availability signals, presence signals, and formatter full signals. Available signals indicate if there was a request for the M-Switch without a grant from the M-Switch arbiter. The presence signals are basically the inverse of the available signals, indicating if something is present in the formatter queue. The full signals indicate if there are two instructions sitting in the formatter of a particular path. A path is full if something is present in the queue, there is a new load, and the new instruction is a memory operation that wants to make a request. Additionally, a path is also full if it was full last cycle and there was no grant by the M-Switch. This module handles status signals for both the regular and shadow path. See figure 10 for a timing diagram illustrating the derivations of some of these signals.

4.2.14 Pipeline Control

PIPECTRL handles the control for the formatter and input p-latches. A new instruction is loaded into a particular path if it is a valid operation and if it is from one of the threads that the path accepts. A request is driven to the M-Switch if the appropriate grant signal is asserted. A new request is loaded into the queue in the formatter if there is nothing present there and if there was no grant. Finally, the new instruction is selected to go to the MSW drivers by the MSWDRIVER if the formatter is available. If the formatter is available, this will mean that the current instruction is always in the slot following the input p-latch. When the formatter is not available, the current instruction will be found after the second p-latch in the formatter.

4.2.15 Control Path

CTRLPATH ties together a bunch of modules that are replicated on both the regular and shadow paths. It also has some random logic in it. These pieces of logic have to do with the fact that stale instructions can be sitting in the memory unit if nothing new is loaded into a path. The control has to be sure not to execute these instructions twice. This random logic uses a signal called **new_load** to ensure that instructions are not executed twice. If there was not a new load, then a nop should be executed. Executing a nop entails not writing anything, not validating or invalidating anything, not making a new request, not signalling an exception, and not reincrementing the memory barrier.

4.2.16 Top Level Control

MUCTRL is the top level memory unit control module that puts together the various pieces of the control, include a regular path, shadow path, pipeline status and control modules, and datapath control modules. There is no extra logic at this level.

4.3 Testing

Verification of the memory unit control was done by running many hand crafted cases for each instruction and then manually checking the outputs. There were also separate tests for testing the formatter control.

5 Integration Testing

The control and datapath integration testing again was done using handcrafted cases for each instruction. Then a randomized simulation was run that would generate random inputs. Only the datapath outputs were checked versus a behavioral model. Over 6 million instructions were simulated in this way with there being no errors.

A Datapath Behavioral RTL

B Control RTL

C Integrated RTL

D MU Constants

E Errvals and Exceptions