DRAFT: Do not distribute

MIT Concurrent VLSI Architecture Memo ??

Massachusetts Institute of Technology
Artificial Intelligence Laboratory

The Architecture of the Integer Execution Stage of the MAP Pipeline
Draft Version

Andrew Chang

February 14, 1995

Abstract

This document describes the operation, architecture, and implementation of the Integer Execution stage (EX) in the multi-ALU processor (MAP) pipeline. This document should be read in conjunction with *M-Machine Architecture v1.1* and *M-Machine Microarchitecture v1.0*.

Contents

1 Functional Overview **2**

1.1 Preliminaries . . . 2

1.2 Integer Unit . . . 2

1.2.1 Functions . . . 5

2 Interfaces **5**

2.1 IF/EX Interface . . . 6

2.2 SZ/EX Interface . . . 6

2.3 EX/WB Interface . . . 8

2.4 EX/CSW Interface . . . 8

2.5 CFG/EX Interface . . . 8

3 Operation **8**

3.1 Pipeline Control . . . 9

3.2 Synchronization . . . 9

3.3 Operation Execution . . . 9

3.4 ISENDs . . . 11

3.5 Operation Predecode . . . 11

3.6 CFG Pseudo Operations . . . 11

3.7 Bypassing Operands . . . 11

3.7.1 Data Hazards . . . 11

3.7.2 Control Hazards . . . 16

3.8 Exception Handling . . . 16

3.9 Arithmetic Operations . . . 17

3.10 Error Value Creation . . . 18

4 Implementation **18**

4.1 Timing . . . 19

4.2 Circuits . . . 19

4.3 Area Estimate . . . 19

4.4 Power Consumption . . . 19

5 OPEN ISSUES **22**

1 Functional Overview

The Integer EX stage of the MAP pipeline performs all integer operations except multiplication and division. Figure 1 shows an overview of the MAP chip pipeline.

The each cluster in the MAP chip possess three seperate EX pipelines: Integer, Memory, and Floating Point. All of the execution units are responsible for performing the following operations:

- Control multiplexing between normal and Bypass sources.
- Detect Conditional Nullification.
- Initiate C-Switch Arbitration.
- Generate register invalidation to SZ and RR stages.
- Generate register validation to SZ and RR states for Bypass results (ie. not in Register File).
- Bypass data to OPSYNC register, Instruction Buffers, and EX register.
- Provide new Instruction Pointer to IF stage during branches or jumps
- Detect Exceptions and initiates handling
- Create Error Values.
- Signal "Not Available" when arbitration is lost for either the CSW or MSW.

The MAP performs *in-order issue* and allows *out-of-order completion*. Out-of-order completion improves performance in the presence of long latency operations. No operation is issued if it has an unresolved data dependency. Several bypass pathes are provided to allow back-to-back operations to use the destination of the first operation as a source for the second.

1.1 Preliminaries

All data is *internally* represented in the MAP chip in *two's complement* format.

The byte ordering in the MAP chip defaults to *Little-endian* format. Alternately, an additional bit can be added to the MAP global state to control byte-ordering on Loads and Stores.

1.2 Integer Unit

Figure 2 shows the high level logical block diagram of the Integer EX stage.

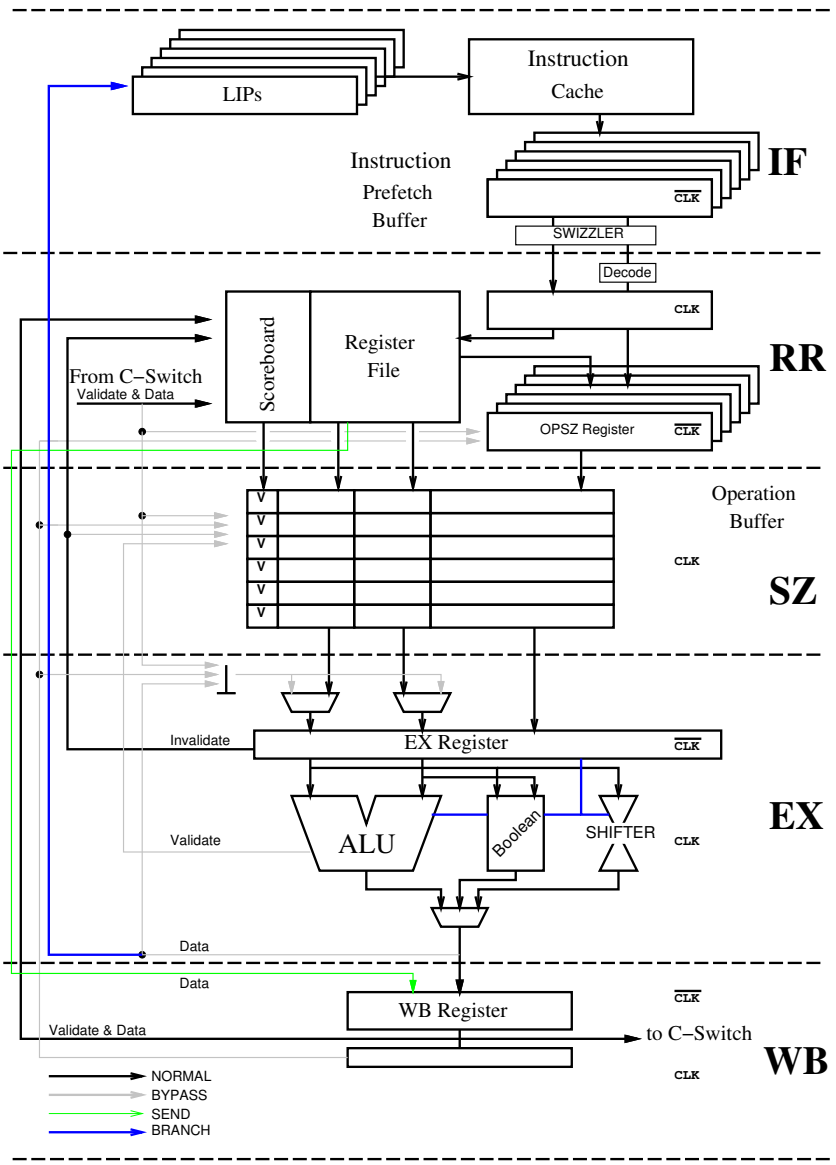


Figure 1: *The MAP Pipeline*

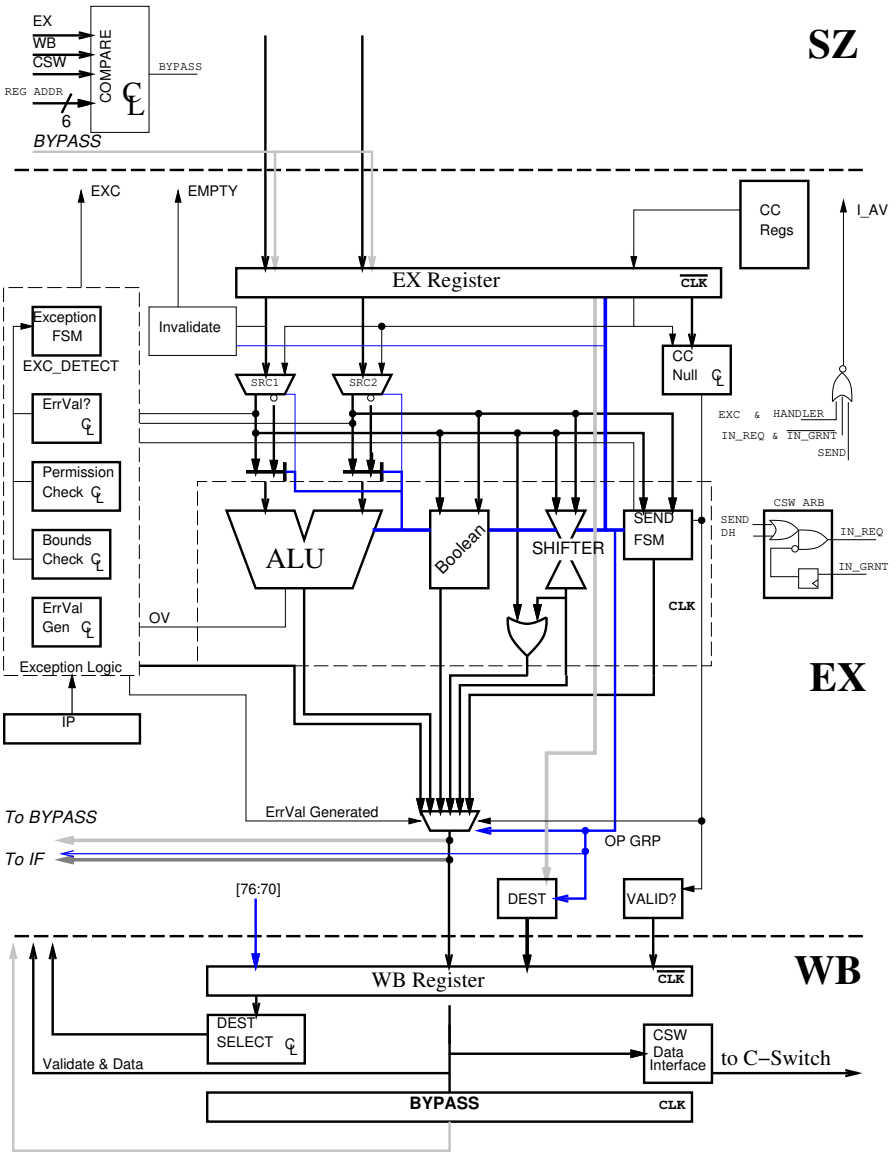


Figure 2: *IU Logical Block Diagram*

1.2.1 Functions

The integer unit (IU) of the MAP performs 43 operations. All operations besides the five ISENDs, complete in one (1) cycle. Please refer to *The MAP Instruction Set Reference Manual v1.1* for details on the specific integer operations supported.

In general, the MAP integer unit:

- Execute all integer operations except for IMUL and IDIV
- Execute ISEND

During operation execution the following tasks are performed *in parallel* by the EX stage:

- Test for conditional nullification
- Begin execution of requested operation
- Arbitrate for CSW if needed
- Perform pointer bounds check for LEA, LEAB, BR, and JMP
- Perform permission check for JMP
- Check for Operands for Errval's

2 Interfaces

The Integer EX stage has five interfaces which are itemized below and detailed in the subsequent figures.

1. Instruction Fetch Stage (IF/EX) - IP for Branches and Jumps
   63-bits
2. Synchronization Stage (SZ/EX) - Operation issue
   236-bits
3. Write Back Stage EX/WB - Data value
   75-bits
4. Cluster Switch (EX/CSW) - Request CSW
   TBD bits
5. Configuration Space FSM (CFG/EX) -
   TBD bits

MAP INTERFACES: IF/EX

INT

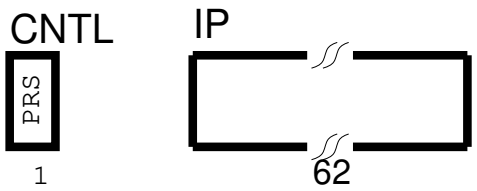


Figure 3: *IF/EX Interface*

2.1 IF/EX Interface

Figure 3 presents the bit fields for the IF/EX interface. During taken branches and jumps the EX stage signals the IF stage that an IP is present and drives the 62 bits of the new IP directly to the IF stage.

2.2 SZ/EX Interface

Figure 4 presents the bit fields for the SZ/EX interface. The EX stage receives seven types of input from the SZ stage:

1. Condition Code and designated CC register
2. Instruction Pre-decode - Group and Type
3. Source Operand Values
4. Destination Register Name
5. Send Length
6. Current IP
7. Pipeline Control including Present, Valid and Illegal Operation Flags

The EX stage generates I_AV and EXC for the SZ stage.

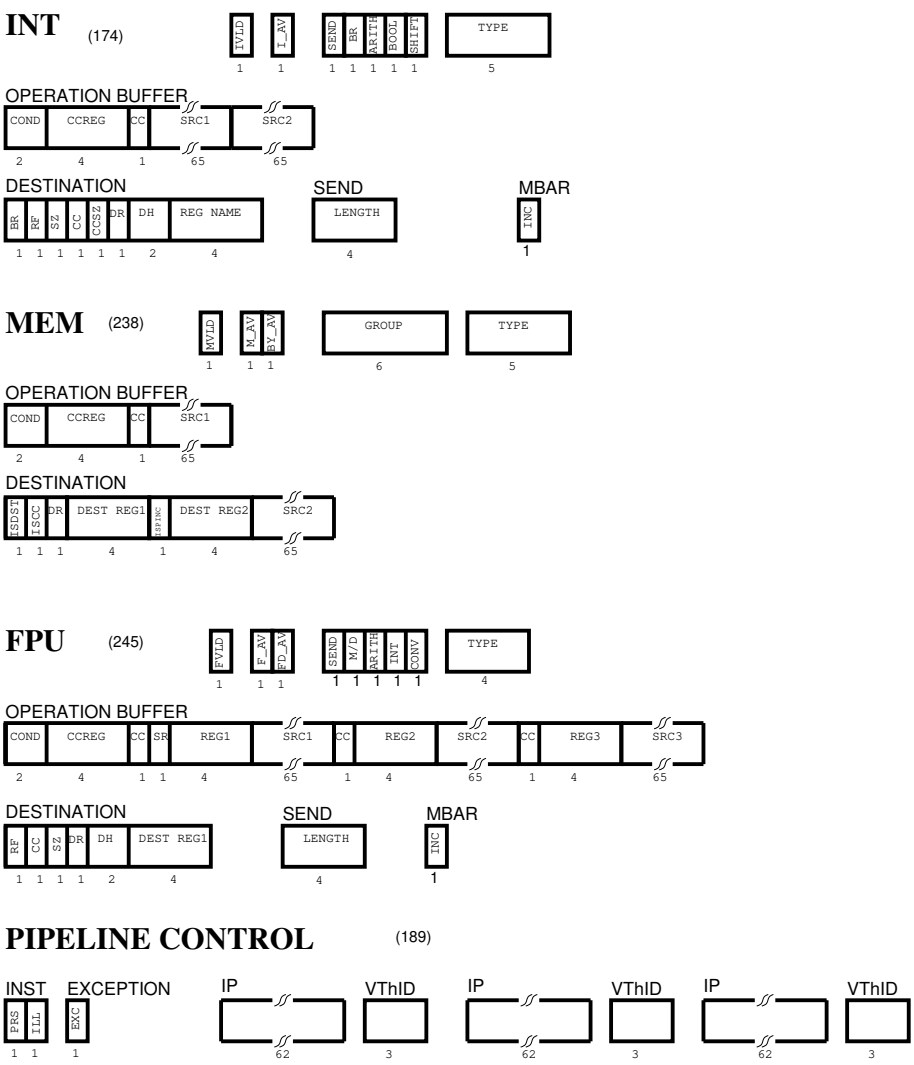


Figure 4: *SZ/EX Interface*

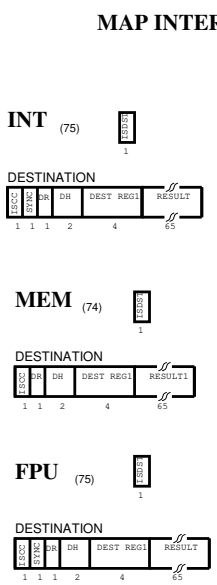


Figure 5: *EX/WB Interface*

2.3 EX/WB Interface

Figure 5 presents the bit fields for the EX/WB interface. The EX stage provides the destination type, the register name and the result data to the WB stage.

2.4 EX/CSW Interface

The bit fields for the EX/CSW interface remain *to be defined.*

2.5 CFG/EX Interface

The bit fields for the CFG/EX interface remain *to be defined.*

3 Operation

This section describes the operation of the MAP pipeline and explores the following functions in detail: pipeline control and sequencing, synchronization, operation execution, exception detection, operation pre-decode, pipeline hazards, exception handling, and error value generation.

3.1 Pipeline Control

Stalls: The MAP Pipeline stalls under the following conditions:

- Loss of CSW Arbitration stalls *both* the Integer and Floating Point execution pipelines
- Loss of MSW Arbitration stalls the memory unit execution pipeline.
- ICache Miss *ONLY* stalls instruction issue for the relevant thread.
- Exception handling stalls EX and WB stages for *USER* threads only.

3.2 Synchronization

All register invalidation/validation must complete in the *first half* of the cycle. Since all integer operations (except ISENDs) complete in one cycle, *no* explicit invalidation is performed.

3.3 Operation Execution

Conditional Nullification All operations are conditionally executed. The four conditions are:

- *UA* Execute Unconditionally Always
- *UN* Execute Unconditionally Never
- *CF* Execute conditionally if False
- *CT* Execute conditionally if True

In parallel with the start of execution, the EX stage tests the condition specified with the instruction against the value of the indicated condition code register and potentially nullifies the result of the operation. Note, in all cases except ISEND, the operation is allowed complete in the execution unit. The results are prevented from changing state up setting the "valid" bit on the input to the WB stage to False. Therefore, an operation issue slot is wasted for a conditionally nullifed instruction but no state is modified.

In the case of ISEND, the CSW request is aborted and I_AV is re-asserted after one cycle.

The values of the condition code registers are bussed across the EX unit and are, therefore, readily available for testing.

Figure 7 presents the logic to perform CC nullification checking.

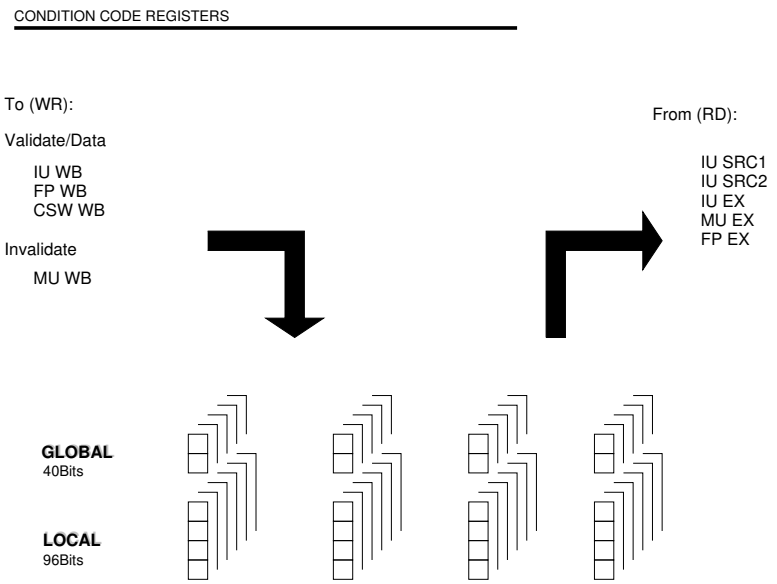


Figure 6: *Condition Code Registers*

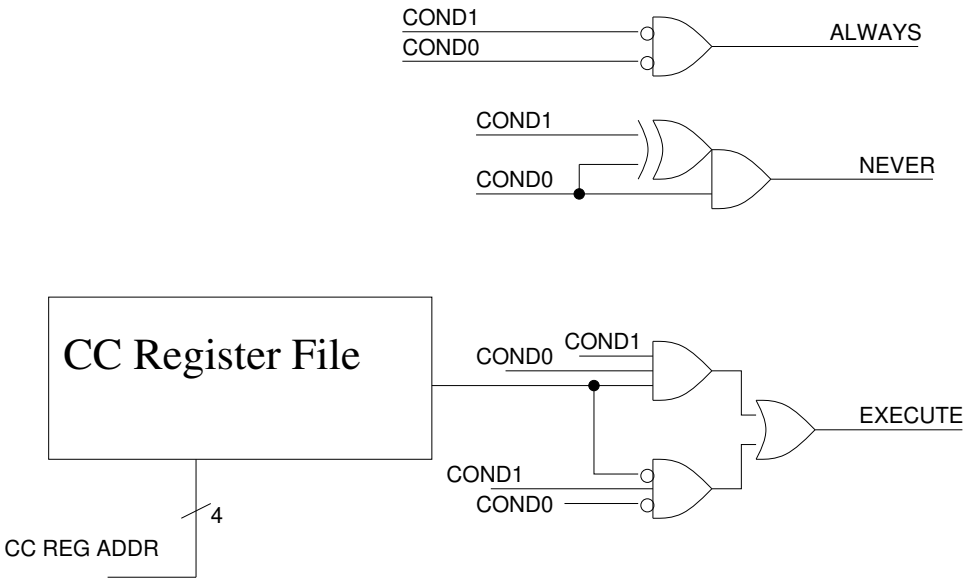


Figure 7: *Condition Nullification Check Logic*

3.4 ISENDs

The integer unit is responsible for executing an ISEND. Figure 8 shows both the logical state transition diagram and a physical implementation requiring only the SEND bit from the pipeline register, a 2-bit shift register, and two 4-bit counters.

The 2-bit shift register is used to sequence the transmission of the SRC1 and SRC2 operands before accessing the integer register file for Message composition register file values.

3.5 Operation Predecode

The RR stage "predecodes" the operation code into group and type to expedite execution in the EX stage.

Figure 10 presents the group and type fields "predecoded" by the RR stage for integer operations.

3.6 CFG Pseudo Operations

The CFG space interface and the implementation of CFG space pseudo operations remains *to be defined.*

3.7 Bypassing Operands

All operations in the integer unit, except for ISEND, are single cycle operations and thus, do not invalidate/validate their destination registers. However, several data bypass pathes must be provided:

1. Output of the EX stage to input of EX stage (latched $\overline{clk}$)
2. Output of WB stage to input of EX stage (latched $\overline{clk}$)
3. Output of WB stage to Instruction Buffers (latched clk)
4. Output of WB stage to OPSYNC Buffers (latched $\overline{clk}$)

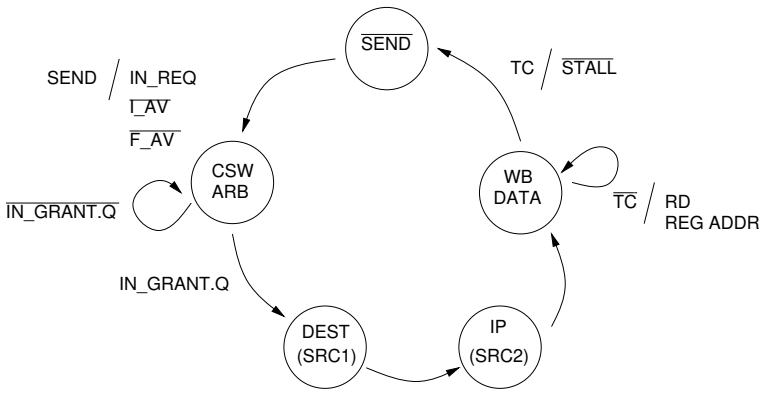
Figure 11 shows an example using the described bypass pathes.

3.7.1 Data Hazards

All multi-cycle latency operations mark their destination registers invalid upon successful issue (ie no conditional nullification or exception).

The destination registers of all single cycle operations are always valid.

LOGICAL



IMPLEMENTATION

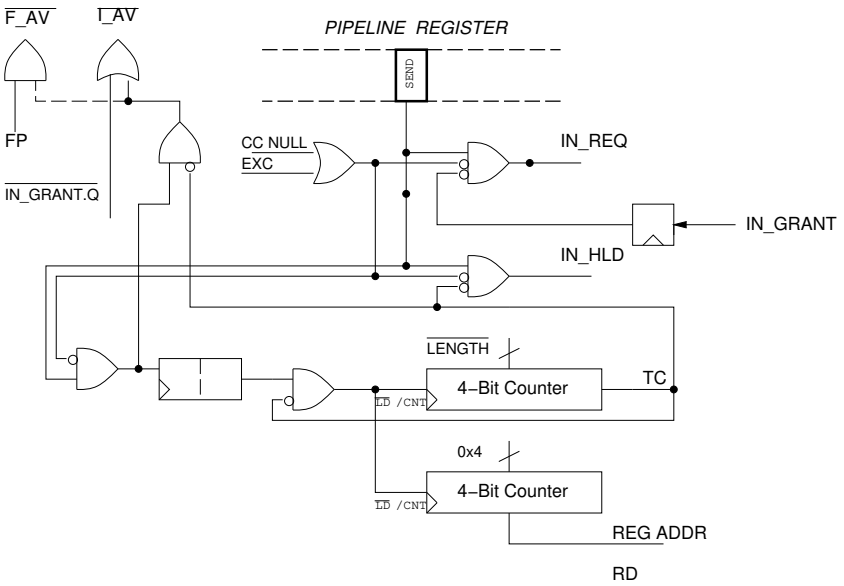


Figure 8: *Send State Diagram*

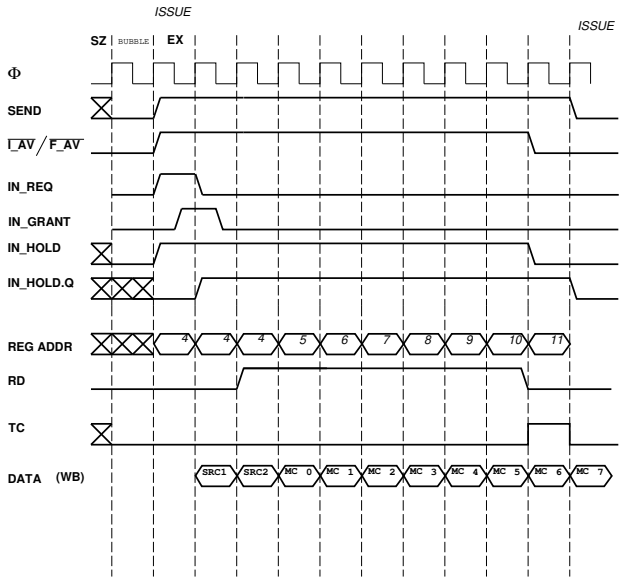


Figure 9: *8 Word Send Timing*

RAW The most common data dependency is "read after write" in which the result of the previous instruction is used as a source operand in the current instruction

Example instruction sequences:

Single cycle operation

```
ADD   i5 i6 i7
SUB   i7 i8 i7
```

The data result of ADD is *bypassed* to the beginning of the EX stage to "meet up" with the SUB operation. Since ADD is a single cycle operation its destination register is always valid.

Multi-cycle operation - in Memory Unit or Floating Point Unit

```
LD    i5 i6 i7
ADD   i7 i8 i7

FMUL   f5 f6 f7
FADD   f7 f8 f7
```

After successfully issuing, the IMUL marks i7 invalid. The ADD is prevented from issuing until the IMUL completes. The IMUL validation, over the CSW, is bypassed to the SZ stage Instruction Buffer (latched on clk) and the data result is bypassed to both the Instruction Buffer and the beginning of the EX stage (latched on clk).

INT

GROUP

SEND	BR	ARITH	BOOL	SHIFT

TYPE

SEND — 5

X	[GTLB]	[Pri]	[Phys]	[~INC]

BR — 2

X	X	X	[TRIP]	[JMP]

ARITH — 11

PTR	NEQ	SIGN	EQ	$\overline{SRC2}$		
0	0	1	0	0	ADD	
0	0	0	0	0	ADDU	
0	0	1	0	1	SUB	
0	0	0	0	1	SUBU	
0	0	1	0	1	ILT	**CC DEST**
0	1	1	1	1	ILE	
0	0	0	0	1	ULT	
0	1	0	1	1	ULE	
0	0	1	1	1	IEQ	
0	1	1	0	1	INE	
1	0	1	0	0	LEA	
1	0	1	0	1	LEAB	

BOOL — 12

MOV	PTR	XOR	NOT	AND/$\overline{OR}$		
0	0	0	0	0	OR	
0	0	0	0	1	AND	
0	0	0	1	0	NOT	
0	0	1	0	0	XOR	
0	0	0	0	0	CCOR	**CC DEST**
0	0	0	0	1	CCAND	
0	0	0	1	1	CCNAND	
1	0	0	0	0	MOV, EMPTY, CCEMPTY	
0	1	0	0	0	SETPTR	
0	1	0	0	1	ISPTR	**CC DEST**
0	1	1	0	0	ISERR	

SHIFT — 9

OR	H/$\overline{B}$	W	ROT	LG/AR	
0	0	1	0	0	ASH
0	0	1	0	1	LSH
0	0	1	1	1	ROT
0	0	0	0	1	EXTB
0	1	0	0	1	EXTH
1	0	0	0	1	INSB
1	1	0	0	1	INSH
1	0	1	0	1	SHORU
1	0	1	0	1	IMM

Figure 10: *Integer Unit Operation Predecode*

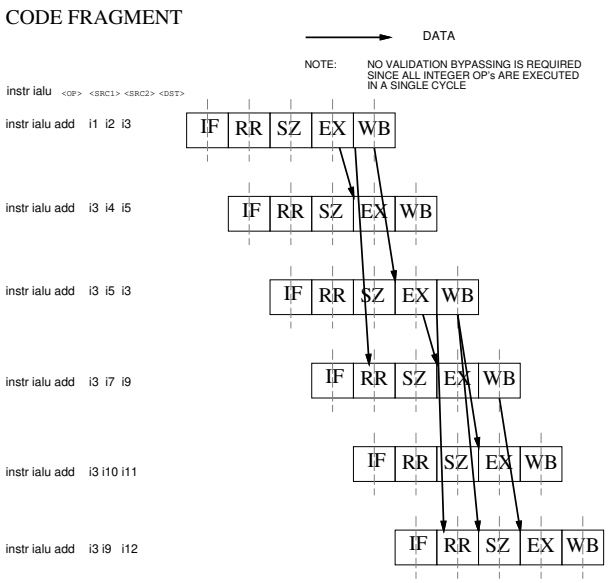


Figure 11: *Bypassing Example*

WAR The M-Machine architecture guarantees *In-Order Issue* and thus prevents the results of subsequent operations from overwriting source values of previous operations.

```
LD    i5 i6 i7
SUB   i4 i8 i5
```

WAW The M-Machine architecture supports *Out-of-order* operation completion, and therefore provides *NO* guarantees regarding the order of writes.

The following instruction sequence is a race condition and is defined to be a programming error:

```
LD   i5 i6 i5
ADD  i6 #1 i5
LD   i5 i7 i8
```

The value of i5 is non-deterministic. If a programmer is concerned about the exact ordering of writes, the following sequence may be used:

```
LD   i5 i6 i5
MOV  i5 i5
ADD  i6 #1 i5
LD   i5 i7 i8
```

The additional MOVs guarantee that the IDIV will complete before the ADD is issued.

3.7.2 Control Hazards

Branches The IF stage double buffers the IP to correctly support the case of a branch during an ICACHE miss for the instruction in the third delay slot.

For either a taken branch or a jump, the integer execution stage supplies the new IP directly to the IF stage. It is the IF stage's responsibility to check for ErrVal in the provided pointer.

The M-Machine architecture has *three (3)* branch delay slots.

3.8 Exception Handling

The M-Machine architecture categorizes interrupts into two clases: exceptions and events.

Exceptions are synchronous and local to an individual cluster. Exceptions are handled by the dedicated X thread in each cluster. Each cluster may only have one exception active at any single instant.

Events are asynchronous and occur outside the clusters and require system software intervention. Memory synchronization faults and block status faults are examples of events. Events are queued in a hardware event queue and resolved by software event handlers.

The integer execution unit can generate an exception under the following four conditions:

1. ErrVal is a source operand
    - For compare operation
    - For BR or JMP
    - For CCMOV or MOVCC
    - For ISEND
    - For SHIFT operation
    - For SHIFT operation
2. Permission violation
    - For JMP
    - For ISEND
    - For LEA and LEAB
3. Non-Pointer
    - For SRC1 $\neq$ Pointer in ISEND
4. Illegal instruction (ILL)

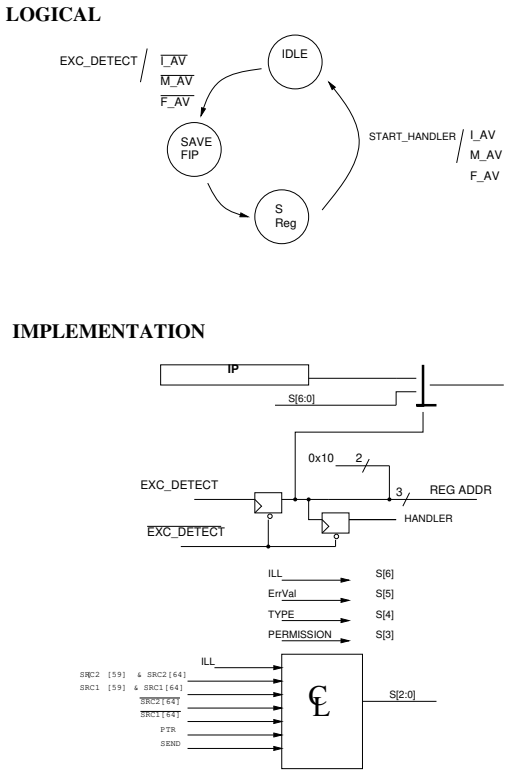


Figure 12: *Exception Handler FSM*

Figure 12 shows the state transition diagram for the exception finite state machine (FSM) . The integer unit exception FSM performs the following four tasks after an exception is detected.

1. Set I_AV, M_AV, and F_AV to false
2. Write the IP into i4
3. Create the exception vector and write it to i5
4. Assert I_AV, M_AV, and F_AV to true

3.9 Arithmetic Operations

By convention, comparison operations are the *only* integer arithmetic operations that can generates an exception.

ADD and SUB Overflow during integer ADD, SUB return the low-order bits of the true two's complement result.

IMUL Note, IMUL and IMULU are performed by the Floating Point execution pipeline.

Overflow during signed integer multiplication (IMUL) generates the lower bits of the true two's complement result.

Overflow during unsigned integer multiplication (IMULU) also produces the lower bits of the true two's complement result.

Integer Division Note, IDIV and IDIVU are performed by the Floating Point execution pipeline.

Overflow during signed integer division (IDIV) produces an error value (ErrVal).

Unsigned integer division (IDIVU) can not overflow.

Divide by zero for either type of integer division produces an error value.

Conversion Note, FTOI and FTOIU are performed by the Floating Point execution pipeline.

Overflow during signed floating-point to integer conversion (FTOI) produces an error value.

Overflow during unsigned floating-point to integer conversion (FTOIU) also produces an error value.

3.10 Error Value Creation

The creation and usage of ErrVal's allows the MAP processor to schedule and perform speculative operations and improve performance. ErrVals do not raise exceptions unless they are used in a ST operation. Arithmetic operations on Errvals return the SRC1 Errval. Figure 13 shows the logical implementation of ErrVal creation hardware for the integer execution unit.

In the Integer execution pipeline error values can only created by under two conditions:

1. BR, LEA, LEAB, beyond segment
2. ErrVal used as one of the source operands for non-compare operation

4 Implementation

This section describes and discusses the implementation issues for the described logical design.

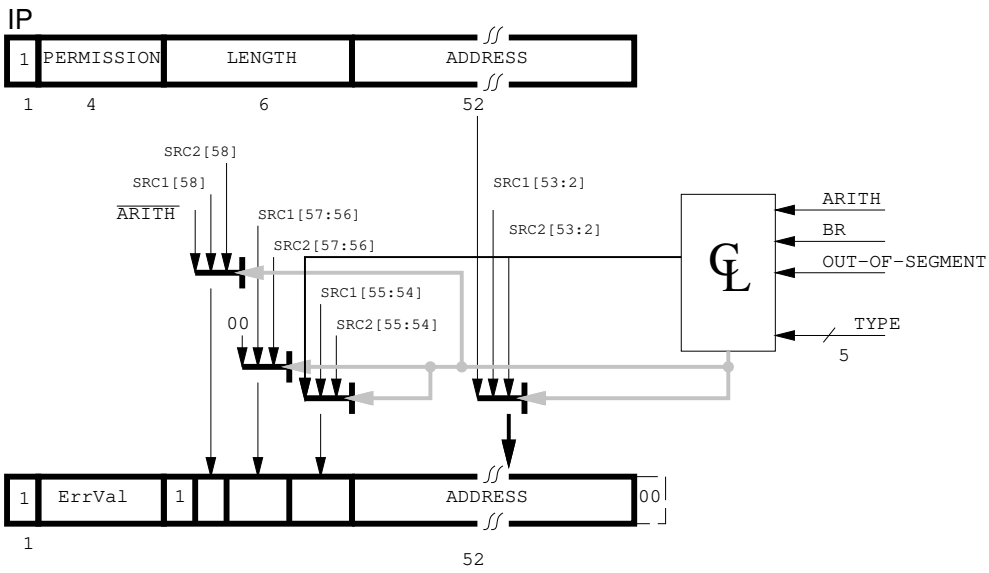


Figure 13: *Integer Unit Errval Creation*

4.1 Timing

Figure 14 shows the expected timing for a normal operation. All protection, exception, and conditional nullification checks are performed in parallel with operation execution. A final multiplexor selects the output supplied to the WB stage.

4.2 Circuits

A circuit study is currently in progress.

4.3 Area Estimate

An estimate of chip area is currently in progress.

The anticipated area for the EX register is $2\text{M}\lambda^2$. The current area estimate for the complete Integer EX stage including pipeline registers, 64-bit adder, shifter and boolean array is $20\text{M}\lambda^2$. Figure 15 shows the preliminary floorplan.

4.4 Power Consumption

Also, an estimate of chip power is currently in progress.

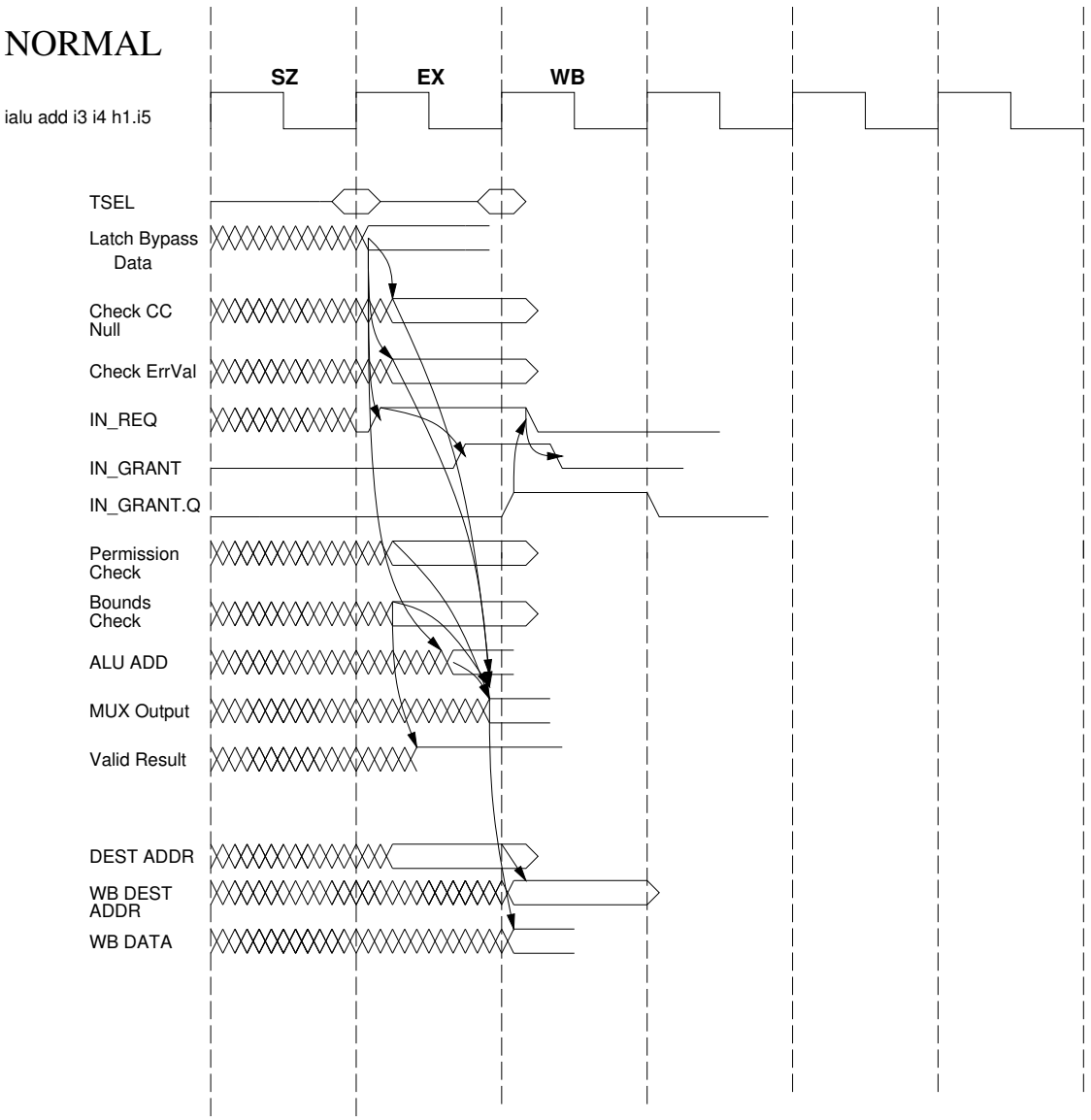


Figure 14: *Integer Unit Timing Diagram*

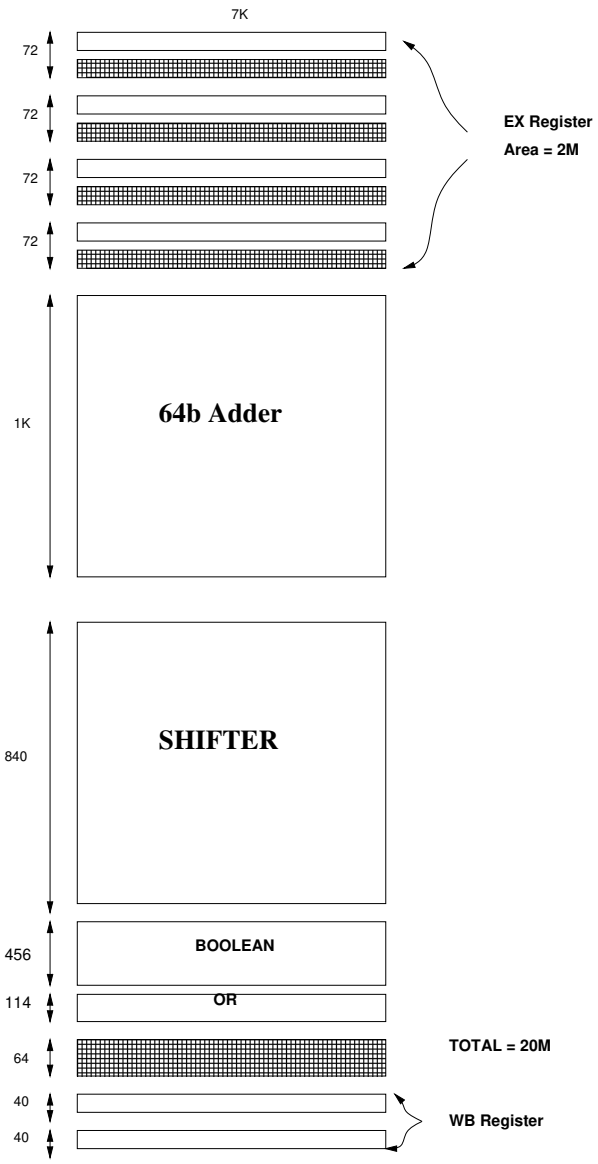


Figure 15: *Integer Execution Stage Floorplan*

5 OPEN ISSUES

The following issues remain to be resolved:

1. The implementation of CFG Space Pseudo Operations
2. Physical location of Local and Global CC Registers
3. Signal distribution for Local and Global CC Registers
4. The definition of the EX/CSW interface
5. The definition of the CFG/EX interface