Table of Contents

February 2, 1998 2:49 pm

TGen

Programmer's Reference Manual

D. A. B. Weikle, Wm. A. Wulf
University of Virginia
{daw4q@virginia.edu, wwulf@nae.edu}

1.0 Introduction

The TGen (Trace Generation) Processor is a VLIW (Very Long Instruction Word) machine designed to generate general memory references to an arbitrary memory system. The first version, called STAG (Smc TrAce Generation) in the Electrical Engineering Department, is being implemented with an SMC (Stream Memory Controller) and a path for simulating memory accesses to a generic hierarchical caching system. See Appendix C for a more complete description of the SMC Implementation.

Specifically, TGen can be programmed to generate a memory reference every cycle to an arbitrary memory system. The design is general enough to simulate traces from most current microprocessors and can be included on chip as appropriate for certain memory system designs. Because the processor is not actually performing significant computation, it is also possible to run it at fairly high-speeds without using heroic silicon design, layout, and fabrication. Currently the interface to the memory system is very simple. In fast mode, TGen executes an instruction every clock cycle and, if specified by the instruction, generates a memory reference. If the Hold line indicates the memory system needs time to process the memory request, TGen waits until the Hold line indicates it should continue. Since running at high-speed requires running from a program on-chip, TGen has both a Fast and Slow Mode. It automatically comes up in Slow Mode, executing from a program off chip in External Instruction Storage (EIS). The on-chip instruction memory is loaded from this memory and then control is switched to Fast Mode. When execution of the Fast Mode instructions is complete, control can be switched back to Slow Mode to refill the internal memory. Since this could happen in the middle of a simulated program run, TGen has a Freeze Line it can assert for the memory system. This line indicates that the memory system should hold its current state if at all possible and simulate time “standing still” until TGen has reloaded its internal memories and deasserts the Freeze Line.

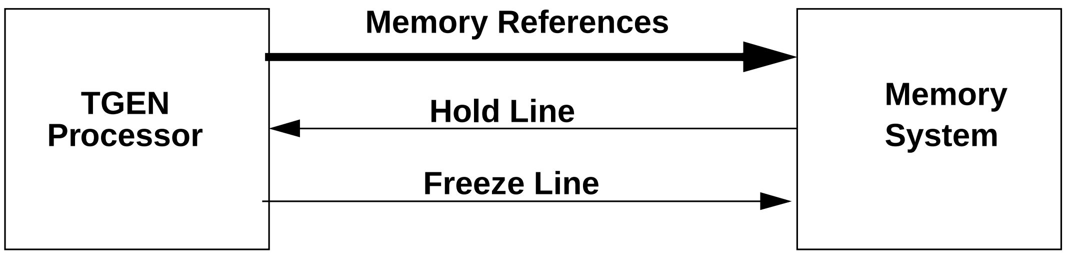

Figure 1: High-level System Concept

The overall architecture of TGen is described in the next section. Its design and implementation are guided by the organization of the TSpec (Trace Specification) Language. TSpec is a simple language designed specifically for the purpose of describing reference traces. See the TSpec specification for details. Although it is not within the scope of this document to cover TSpec in detail, it is useful to realize this connection and know that

TSpec is considered the high-level language of choice for TGen.

2.0 General Architecture

As mentioned in the previous section, TGen is a VLIW machine (See Figure 2). Its instruction word is 64 bits in length, although, all these bits are not always used. The processor runs in two modes, called Slow Mode and Fast Mode. The processor automatically comes up in Slow Mode, executing instructions from its External Instruction Store (EIS). The purpose of Slow Mode is to load the internal memories of TGen so that it can begin execution in Fast Mode. Once it is in Fast Mode, TGen can generate a memory reference every clock cycle. To perform a simulation it may be necessary to run several Fast Mode programs, freezing the memory system in between each to reload the internal memories with Fast Mode Data.

As shown in Figure 2, TGen consists of 5 execution units. One of these, the Slow Mode Execution Unit, provides the interface to the EIS and the means by which the internal memories of each of the other four execution units are loaded. The other four execution units are the VLIW portion of the machine and perform all Fast Mode operations. The ICIS execution unit provides the control for the other three Fast Mode units and contains both the Internal Instruction Store (IIS) and the Internal Instruction Register (IIR). The IIS is where the Fast Mode program is stored, and the IIR holds the VLIW instruction currently being executed by the Fast Mode units. The Memory Address Register Unit (MAR Unit) generates addresses for the non-SMC portion of the external memory under test, while the SMC Unit provides control for the SMC portion. The Counter Unit provides a mechanism for counting loop iterations and provides an indicator to the ICIS execution unit when a particular count has been reached.

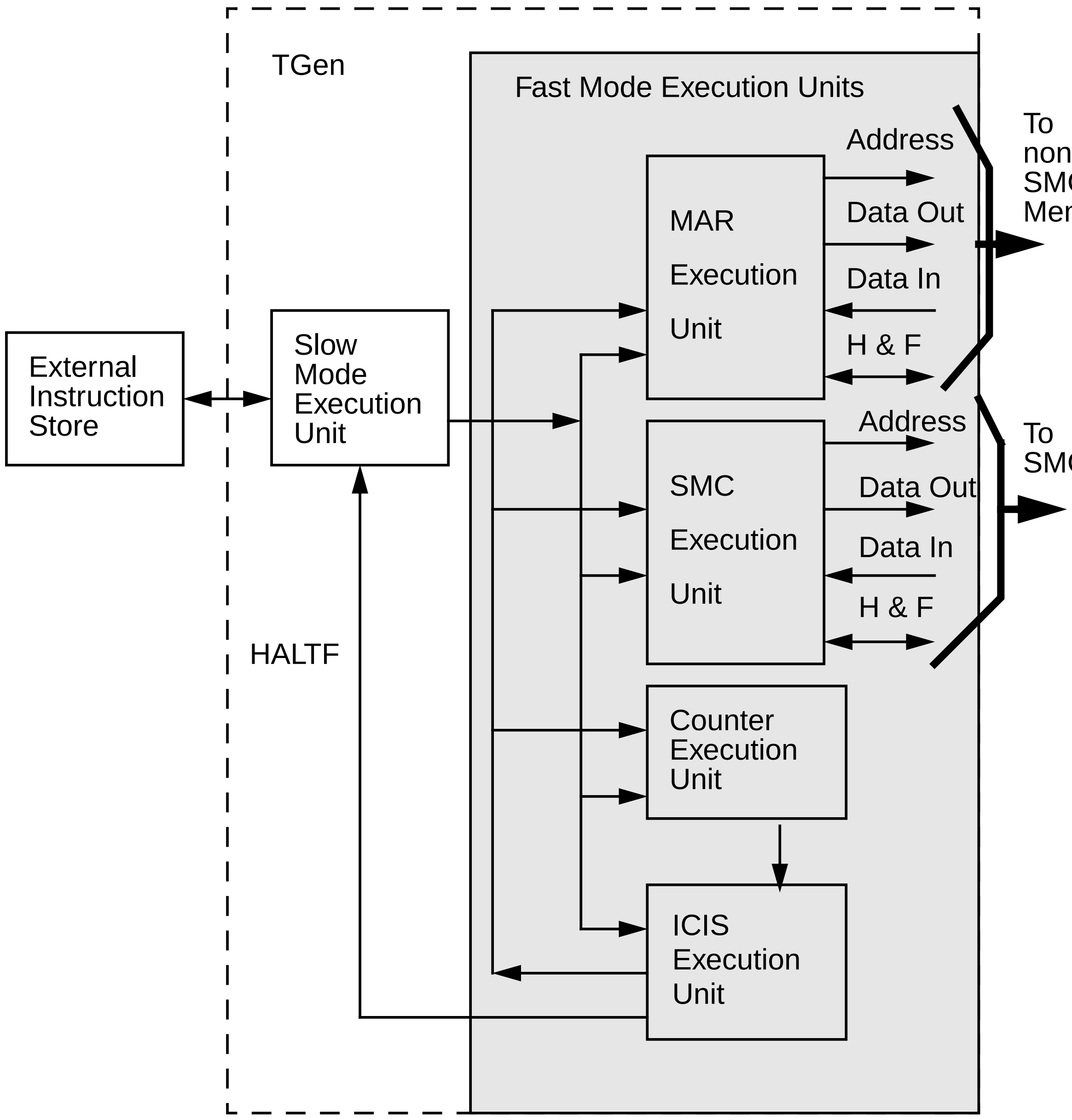

Figure 2: TGen Block Diagram

3.0 Slow Mode Formats

On startup the processor comes up in Slow Mode. In this mode, instructions are executed from the External Instruction Store (EIS) beginning at address 0x0 and the Freeze Line to the memory system being tested is always asserted. The words in the EIS can serve the purpose of either instructions or data. The instructions are used to load TGen's internal memories and the data are what is loaded into the internal memories with these instructions.

3.1 Slow Mode Instruction Format

Figure 3 shows a diagram of the TGen instruction word for Slow Mode. As the instruction format indicates, Slow Mode instructions are not VLIW. The VLIW functionality of TGEN is present only in Fast Mode.

Instruction Format:

63 61	60 43	42 35	34 30	29 22	21 0
Opcode	External Mem Source Address	# of words to move	Internal Memory	Internal Mem Destination Address	Unused

Figure 3: Slow Mode Instruction Format

3.1.1 NOP

Assembly Language Syntax: NOP

Description: Perform no operation.

Instruction Format:

63		61	60 0
0	0	0	Unused

Figure 4: NOP Instruction Format

Example: NOP

3.1.2 RUN

Assembly Language Syntax: RUN

Description: Begin execution in Fast Mode at the instruction pointed to by IP1.

Instruction Format:

63		61	60 0
0	0	1	Unused

Figure 5: RUN Instruction Format

Example: RUN

3.1.3 HALT

Assembly Language Syntax: HALT

Description: Stop processor execution.

Instruction Format:

63		61	60 0
0	1	1	Unused

Figure 6: HALT Instruction Format

Example: HALT

3.1.4 BMV - Block Move

Assembly Language Syntax: BMV 0xExtAddr, 0xCount, MEM(0xIntAddr)

Description: Block move the number of 64-bit words indicated by Count from the External Instruction Store starting at the address indicated by ExtAddr into the internal memory indicated by MEM at the address indicated by IntAddr. Note that each 64-bit word in the External Instruction Store is assumed to contain only one word for the internal memory. This means that if the internal memory is only 8-bits wide, the 64-bit quantity is moved into the processor but only the least significant 8-bits are moved into the internal memory. The higher-order bits are dropped.

Instruction Format:

63		61	60 43	42 35	34 30	29 22	21 0
1	0	1	ExtAddr	Count	MEM	IntAddr	Unused

Figure 7: BMV Instruction Format

ExtAddr - Hexadecimal address of the location in the 64-bit External Instruction Store from which the block move will start. Note that these address 64-bit quantities, so 0x0 is the first 64-bit word in the memory and 0x1 is the second.

Count - Hexadecimal number indicating the number of 64-bit words to move from the External Instruction Store. Count +1 words are actually moved.

MEM - Indicates which internal memory is the destination of the block move. Can have one of the following five designations.

ACM -- 10000 -- Address Control Memory in MAR group, width 56 bits
SCM -- 01000 -- SMC Control Memory in SMC group, width 37 bits
IIS -- 00100 -- Internal Instruction Store in ICIS group, 54 bits
ICM -- 00010 -- Instruction Control Memory in ICIS group, 8 bits
CCM -- 00001 -- Counter Control Memory in Counter group, 32 bits

IntAddr - The address in the internal memory designated by MEM of the destination of the block move.

Example: BMV 0x16, 0x4, ACM(0x0)
Moves 5 words from the EIS starting at address 0x16 into the ACM starting at address 0.

3.2 Slow Mode Data Formats

Each internal memory discussed in 3.1.4 above has a depth of 32, except for the IIS, which has a depth of 256, but each has a different width and each uses these bits for their own purpose. These internal memories are loaded in Slow Mode with the BMV instruction. The format of the data in each internal memory is described below.

3.2.1 ACM - Address Control Memory

The Address Control Memory is in the MAR Group and contains the initial values and increments of all of the variables to be used in this Fast Mode run. It is 56-bits wide and 32 elements deep.

Data Format:

63 56	55 40	39 32	31 0
Unused	Increment	Tag	Memory Address

Figure 8: ACM Data Format

Increment - Hexadecimal number that indicates the constant increment of the variable associated with this data.

Tag - Hexadecimal number providing the information normally on control lines for a memory access such as read/write, supervisor/user, memory/io, data/code. Also provides control for simulations of cache misses. Add detailed explanation of tag bits here! *** Note: Probably want to have mnemonics for the tags, and default value that makes sense.

Memory Address - Hexadecimal number that indicates the initial value of the variable associated with this data.

Example: DATA ACM 0x0, 0x0, 0x4 ****

3.2.2 SCM - SMC Control Memory

The SMC Control Memory is in the SMC Group and contains data used to initialize the SMC and its FIFOs. Two data words are required to set up each FIFO. The first word is in data format A and contains the base address for the FIFO, and the size of the data being held in the FIFO. The second word is in data format B and contains the depth, stride and length for the FIFO. Both control words must be written out to the SMC from the SCM using Fast Mode instructions to initiate work at an SMC. The word in data format A must be written first, and the word in data format B second to insure proper operation of the SMC. Please see Section 4.4 SMC Group on SMC Fast Mode Instructions for more details.

Data Formats:

63 37	36 35	34 32	31 0
Unused	Unused	Size	Base Address

Figure 9: SCM Data Format A

Size - The size of the data item to be stored in the FIFO.

Single (32 bits) - 001
Double (64 bits) - 010
Quad (128 bits) - 100

Base Address - the physical memory address of the first data item to be placed in the FIFO.

63 37	36 32	31 16	15 0
Unused	Depth	Length	Stride

Figure 10: SCM Data Format B

Depth - The number of data items this FIFO should be set up to hold. Possible depths are measured in elements and can have the values specified below.

8 elements	00001
16 elements	00010
32 elements	00100
64 elements	01000
128 elements	10000

Length - the total length of the vector to be held in this FIFO, specified in bytes.

Stride - the difference between the addresses of consecutive data items to be stored in the FIFOs, specified in elements. A stride of one indicates an address difference as specified in the Size field of Data Format A. ****Check this with Adam.

Example: DATA SCM 0x1, 0x20
DATA SCM 0x400, 0x80, 0x1

3.2.3 IIS - Internal Instruction Store

The Internal Instruction Store is in the ICIS Group and is used to store the Fast Mode program currently being run. It is 54 bits wide and 256 elements deep.

Data Format:

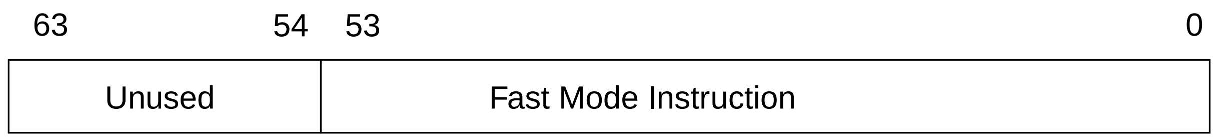

Figure 11: IIS Data Format

Example: DATA IIS Fast Mode Instruction Format

3.2.4 ICM - Instruction Control Memory

The Instruction Control Memory is in the ICIS Group and is used to store the initial values of Instruction Pointers (IPs) for subroutines and loop bodies in the IIS.

Data Format:

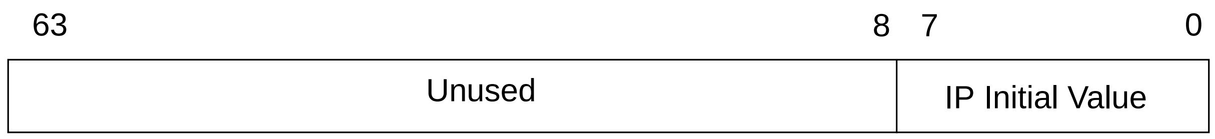

Figure 12: ICM Data Format

Note the range of values an IP can legally have is 0-255, as this is the range of legal addresses into the IIS.

Example: DATA ICM 0x0

3.2.5 CCM - Counter Control Memory

The Counter Control Memory is in the Counter Group and is used to store the initial values of counters that control the number of times a loop is executed. This corresponds to the number of repetitions of a pattern in the TSpec language.

Data Format:

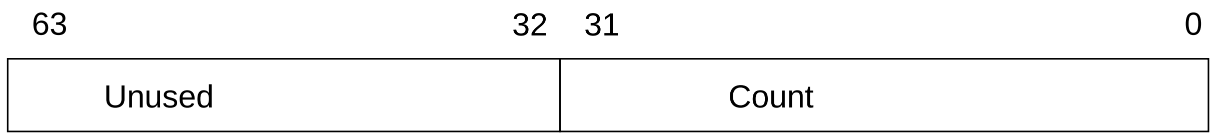

Figure 13: CCM Data Format

Example: DATA CCM 0x0

4.0 Fast Mode

To generate memory references, TGen must be in Fast Mode. In this mode, instructions are executed from the Internal Instruction Store (IIS). It is also in this mode that the VLIW aspect of the architecture becomes apparent. In general, any combination of operations can be run at the same time, but there are a couple of exceptions as documented in Appendix B. So, with those exceptions, any combination of F, ICIS, Counter, MAR, SMC, and H instruction can be run together. Below is a description of the entire fast mode instruction format. Each of the main instruction groups are explained in detail in the sections that follow.

Instruction Format:

63 54	53	52 37	36 26	25 10	9 1	0
Unused	F	ICIS	Counter	MAR	SMC	H

Figure 14: Fast Mode Instruction Format

F - Freeze bit, when set to 1 this bit asserts the external Freeze line and the memory system halts giving the processor time to reload its internal memories and continue a simulation. In Slow Mode, there is no Freeze Bit because the external Freeze Line is automatically asserted to freeze the memory system under test. Note: need to verify specifics on how many in a row, etc. - see Adam******

ICIS - Internal Control and Instruction Store Group, this group of instructions provides the control for the internal execution of the processor and is where the Fast Mode Program is stored.

Counter - This group of instructions provides the mechanism for counting loop iterations.

MAR - Memory Address Register Group, this group of instructions provides registers for initializing variables and their constant increments as well as generating immediate addresses to be output by the processor.

SMC - Stream Memory Controller Group, this group of instructions provides a special interface to the SMC system on chip with the processor, allowing it to be initialized and used as if it were connected to a normal processor.

H - Halt bit, when this bit is set execution in fast mode stops and control is transfered to the next instruction in the External Instruction Store and execution continues in Slow Mode. Note: This bit needs to be set for at least 4 Fast Mode Instructions to make sure a HALT is executed. Unfortunately it is undetermined which of these 4 instructions will actually initiate the transfer to Slow Mode, so they should be executed with NOPs in the other fields of the VLIW instruction.

4.1 ICIS - Internal Control and Instruction Store Group

The Internal Control and Instruction Store Group is the most complex of the execution units in TGen. It is the unit that provides instruction fetch and decode in Fast Mode and provides for execution of loops and subroutine calls in TSpec. Figure 15 shows a block diagram of the ICIS execution unit.

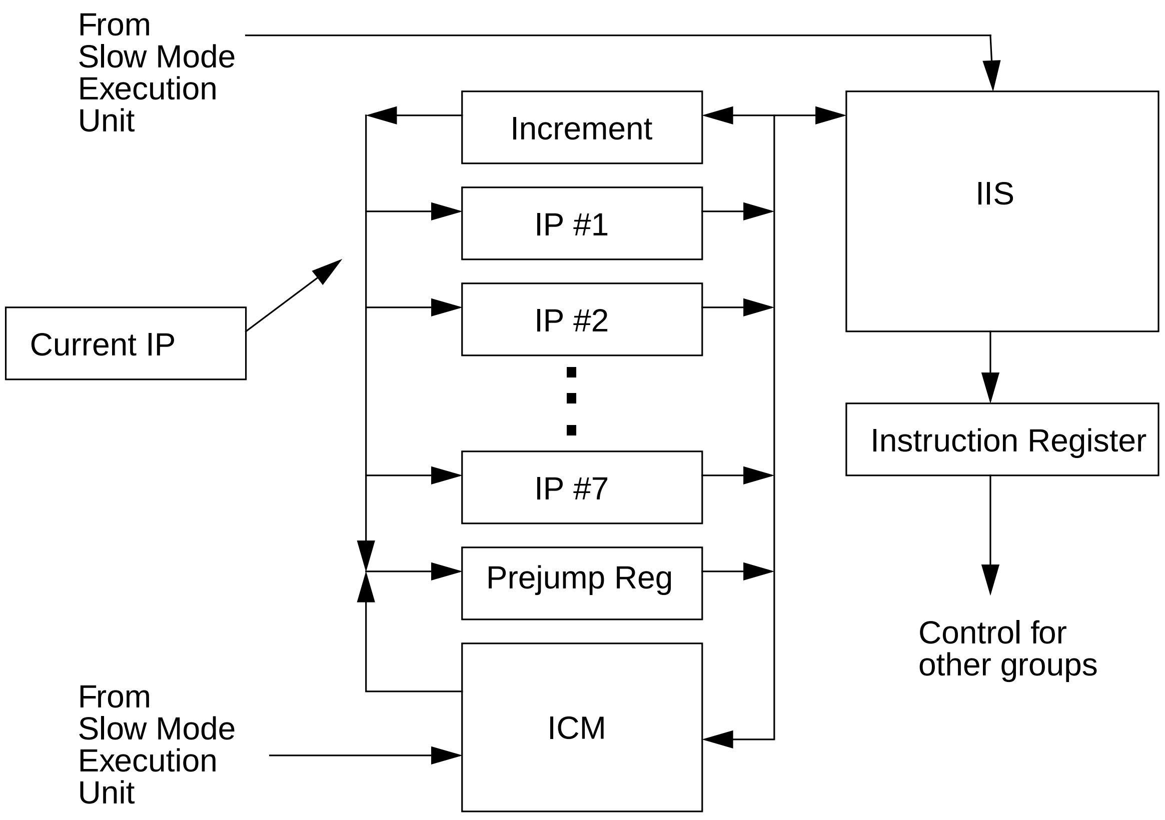

Figure 15: ICIS Execution Unit

The ICIS unit is composed of the Current IP (Instruction Pointer) pointer, seven IPs, a Prejump Register, an incrementer, the ICM, the IIS, and an IR (Instruction Register). The Current IP is a 3-bit pointer to which IP is currently active (ie which IP is being used to load the Instruction Register from the IIS). The seven IPs are used to implement a program call stack. The ICM (Instruction Control Memory) is 8 bits wide and 32 elements deep and is used to hold the initial values of IPs and loop beginnings. The IIS (Internal Instruction Storage) is 54 bits wide and 256 elements deep and contains the Fast Mode program currently being executed. The incrementer is used to increment an IP so that it can be used to fetch the next instruction from the IIS.

Fast Mode execution begins using IP1. When a CALL is executed, control is switched to the next greater IP. For the first CALL this would be IP2. When a RET (return) is executed, control is switched to the next lower IP. If one CALL and one RET have been executed, control would again be through IP1. Control is actually switched via use of the Prejump Register for speed improvements. Prior to the execution of a CALL, the Prejump Register is loaded from the ICM with the address of the subroutine to be called. When a CALL is executed, the Prejump Register is used to get the next instruction and the value of the Prejump Register is sent to the incrementer. The output of the incrementer is then loaded into the next greater IP, and the Current IP is incremented to point to this new IP. Execution then continues from this new IP until another CALL or RET is executed. When a RET is executed, the Current IP is decremented and the next instruction comes from this new IP.

Jumps also take advantage of the presence of the Prejump Register. Prior to executing a JMP, the Prejump Register is loaded with the address that is the destination of the JMP. When a JMP is executed, the next instruction is fetched with the Prejump Register and the value of the Prejump Register is sent through the incrementer into the current IP. A conditional jump (CJMP) is similar, except control is switched to the Prejump Register only if the Z-bit associated with the conditional jump is set. For an explanation of the Z-bit, see section 4.2. For details on specific control instructions see the subsections of 4.1 below.

The ICIS portion of TGEN has two execution units that operate in parallel. The first is the Load/Store (LS) Unit, which loads and stores values between the IP registers and the ICM. The second is the Call/Return/Jump (CRJ) Unit, which executes the control flow instructions. The only parallelism not allowed by the architecture between these two units is a LD of the IP that is currently being used for instruction fetch. (In other words, one cannot issue a LD to the IP currently in use for instruction fetch.) Since the execution units can operate in parallel, the instruction format has opcode and data fields for each unit. The LS Opcode, IPx and ICM Immediate fields are used by the LS Unit, and the CRJ Opcode and Zx fields are used by the CRJ Unit. Note that all instructions, except for CRJNOP that are executed by the CRJ Unit, have one delay slot and cannot be placed in the delay slot of another CRJ instruction. Also, because of the interleaving of the IIS, these CRJ instructions must be executed from an even address and their target must be an even address.

Instruction Format:

52 51	50 48	47 45	44 40	39 37
LS Opcode	CRJ Opcode	IPx	ICM Immediate	Zx

Figure 16: ICIS Instruction Format

LS Opcode - the load/store opcode. This opcode controls loads into the IPs from the ICM and stores from the IPs into the ICM. These operations cannot be done in parallel with each other, but can be done in parallel with operations controlled by the CRJ Opcode.

CRJ Opcode - the call/ret/jmp opcode. This opcode controls calls, returns, and jumps. These opcodes cannot be done in parallel with each other, but can be done in parallel with operations controlled by the LS Opcode. All instructions, except for CRJNOP, that use this opcode field have one delay slot and cannot be placed in the delay slot of another CRJ instruction. Also, because of the interleaving of the IIS, these CRJ instructions must be executed from an even address and their target must be an even address.

IP - Instruction Pointer x. The value of x determines the IP used in a load or store instruction. Values can range from 0-7 with 0 specifying the Prejump register.

ICM Immediate - the immediate address into the ICM. Indicates the source of a load, or the destination of a store.

Zx - Z-bit x. The value of x determines which Z-bit is used for a conditional jump.

4.1.1 LSNOP - Load/Store NOP

Assembly Language Syntax: LSNOP

Description: Do not perform a load or store in the ICIS unit.

Instruction Format:

52	51	50 48	47 40	39 37
0	0	CRJ Opcode	Unused	Zx

Figure 17: LSNOP Instruction Format

Example: LSNOP

4.1.2 ST - Store an IP

Assembly Language Syntax: ST IPx, ICM(Imm)

Description: Store the value of IPx into the ICM at address Imm.

Instruction Format:

52	51	50 48	47 45	44 40	39 37
0	1	CRJ Opcode	IPx	ICM Immediate	Zx

Figure 18: ICIS ST Instruction Format

IPx - Instruction Pointer x. Possibilities include instruction pointers 0-7, where 0 actually indicates the Prejump register.

ICM Immediate - the hexadecimal address of the location in the ICM where the value of the instruction pointer will be stored.

Example:
The ST instruction has an exposed 3-stage pipeline. This means it takes 3 Fast Mode instructions to actually accomplish a store. In the first of these three instructions the destination of the store in the ICM must be specified. The second instruction must have the entire ST instruction specified, and the third instruction must have the source IP speci-

fied.

```
        LSNOP   IPx,    ICM(0x2)
        ST      IP2,    ICM(0x2)
        LSNOP   IP2,    ICM(x)
```

This example will store the value of IP2 in location 2 of the ICM. Note that since in the first instruction the value of the IP is not specified, and in the last instruction the location in the ICM is not specified, consecutive stores can be overlapped so that two stores only take 5 instructions instead of 6.

4.1.3 LD - Load an IP

Assembly Language Syntax: LD ICM(Imm), IPx

Description: Load the value in the ICM at location Imm into IPx.

Instruction Format:

52	51	50 48	47 45	44 40	39 37
1	0	CRJ Opcode	IPx	ICM Immediate	Zx

Figure 19: ICIS LD Instruction Format

IPx - Instruction Pointer x. Possibilities include instruction pointers 0-7, where 0 actually indicates the Prejump register.

ICM Immediate - the hexadecimal address of the location in the ICM from which the value of the instruction pointer will be loaded.

Example: LD ICM(0x6), IP0
Loads the value at address 6 in the ICM into the Prejump (IP0) register.

4.1.4 CRJNOP

Assembly Language Syntax: CRJNOP

Description: Do not perform a change of control from the current IP.

Instruction Format:

52 51	50	49	48	47 45	44 40	39 37
LS Opcode	0	0	0	IPx	ICM Immediate	Unused

Figure 20: CRJNOP

Example: CRJNOP

4.1.5 RET - Return

Assembly Language Format: RET

Description: Return from a subroutine. The next instruction is fetched from the next lower IP and the Current IP is decremented by one. Execution is continued where it left off in the previous subroutine, once the one delay slot instruction has been executed. Note that this instruction should not be placed in the delay slot of another CRJ instruction. Also, because of the interleaving of the IIS, these CRJ instructions must be executed from an even address and their target must be an even address.

Instruction Format:

52 51	50	49	48	47 45	44 40	39 37
LS Opcode	0	1	0	IPx	ICM Immediate	Unused

Figure 21: ICIS RET Instruction Format

Example: RET <--- must be on even address boundary
-- delay slot instruction --

4.1.6 CALL - Call a subroutine

Assembly Language Format: CALL

Description: Fetch the next instruction using the value from the Prejump register, load the next higher IP with the output of the incrementer, and increment the Current IP by one. Execution is through the next higher IP once the one delay slot instruction has been executed. Note that the Prejump register should have been loaded with the address of the subroutine to call prior to executing the CALL instruction. Note that this instruction should not be placed in the delay slot of another CRJ instruction. Also, because of the interleaving of the IIS, these CRJ instructions must be executed from an even address and their target must be an even address.

Instruction Format:

52 51	50	49	48	47 45	44 40	39 37
LS Opcode	0	1	1	IPx	ICM Immediate	Unused

Figure 22: ICIS CALL Instruction Format

Example: CALL <--- must be on even address boundary
-- delay slot instruction --

4.1.7 JMP - Jump

Assembly Language Syntax: JMP

Description: Fetch the next instruction using the value from the Prejump register and load this IP (the one currently pointed to by the Current IP) with the output from the incrementer. Execution is from the same IP, but at a new location once the one delay slot instruction has been executed. Note that this instruction should not be placed in the delay slot of another CRJ instruction. Also, because of the interleaving of the IIS, these CRJ instructions must be executed from an even address and their target must be an even address.

Instruction Format:

52 51	50	49	48	47 45	44 40	39 37
LS Opcode	1	0	1	IPx	ICM Immediate	Unused

Figure 23: ICIS JMP Instruction Format

Example: JMP <---- must be on even address boundary
-- delay slot instruction --

4.1.8 CJMP - Conditional Jump

Assembly Language Syntax: CJMP Zx

Description: If the Z-bit of counter x in the Counter Group is set, perform a jump. To jump, fetch the next instruction using the value from the Prejump register and load this IP (the one currently pointed to by the Current IP) with the output from the incrementer. Execution is from the same IP, but if the jump is performed, execution is from a new location once the one delay slot instruction has been executed. Note that this instruction should not be placed in the delay slot of another CRJ instruction. Also, because of the interleaving of the IIS, these CRJ instructions must be executed from an even address and their target must be an even address.

Instruction Format:

52 51	50	49	48	47 45	44 40	39 37
LS Opcode	1	0	0	IPx	ICM Immediate	Zx

Figure 24: ICIS CJMP Instruction Format

Zx - Z-bit, or overflow bit for counter x. Indicates which bit should be checked before a jump is performed. Values can range from 0-7.

Example: CJMP Z0 <---- must be on an even address boundary
-- delay slot instruction --

4.2 Counter Group

The Counter Group provides the mechanism for generating the same address pattern multiple times. In TSpec, a pattern that appears multiple times has a repetition count associated with it. Keeping track of when this repetition count has been reached is the responsibility of the counter group. As can be seen from Figure 25 below, the Counter Group consists of 8 counters with their associated Z-bits, an incrementer, and the Counter Control Memory (CCM). The counters keep track of the number of times a pattern has been repeated and are 32-bits. The Z-bits are the overflow of the counter they are associated with, are one-bit, and are set when the count has been reached. The CCM is 32-bits wide and 32 elements deep and is where the intital values of the counters are stored during Fast Mode execution.

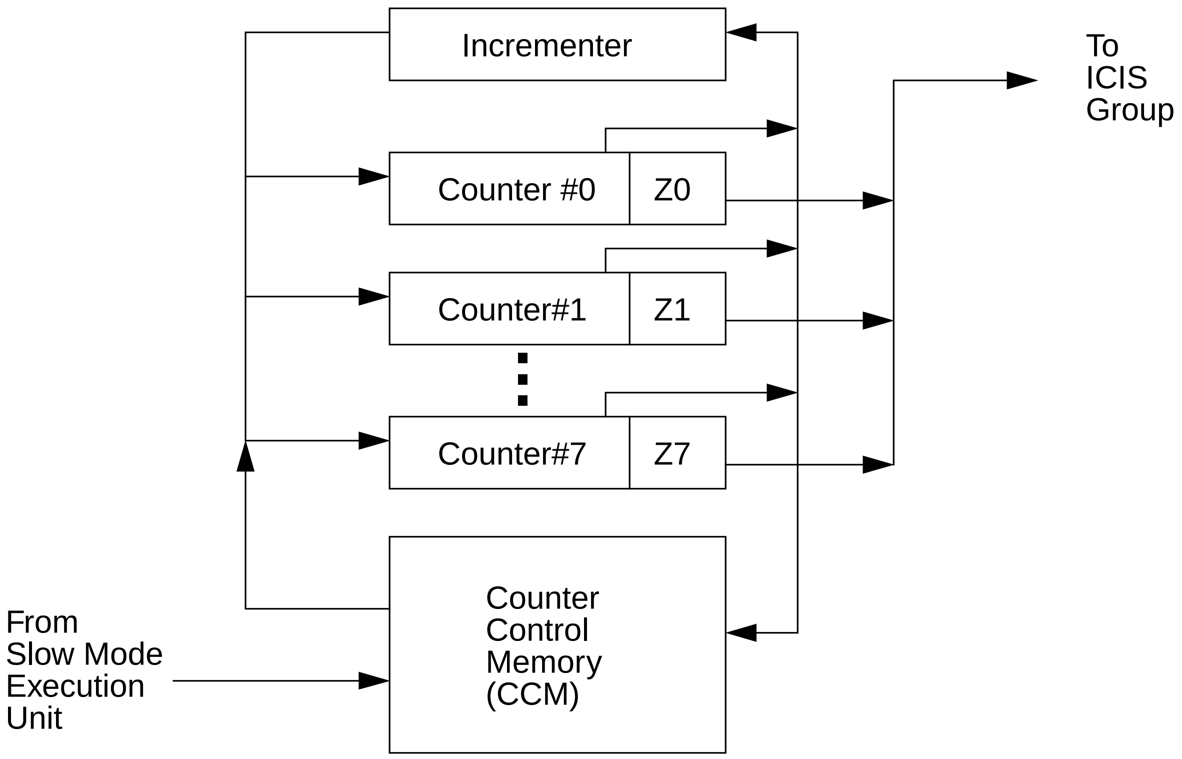

Figure 25: Counter Group Block Diagram

The process to use the Counter Group to keep track of a repetition count is as follows. First the value corresponding to the repetition count is loaded into the CCM in Slow Mode. Then when the address pattern is being set up, the value from the CCM is loaded into a specific counter. (Note this value is actually the max counter value - repetition count + 1) The code to generate the address pattern contains a single increment for every instance of the pattern it generates. When an increment is executed, the counter is incremented. Once the counter reaches its maximum value it overflows, setting the Z-bit associated with it. This Z-bit can then be used in a conditional jump instruction in the ICIS to determine if the pattern should be repeated or not.

The operations that can be performed in the Counter Group are loads and stores of the counters from and to the CCM, and incrementing a given counter. The general format of a Counter Group instruction is shown below and the details of the specific instructions are given in the subsections that follow.

Instruction Format:

36 34	33 29	28 26
Counter Opcode	CCM Immediate	Counter Number

Figure 26: Counter Group Instruction Format

Counter Opcode - inidicates which operation in the counter group will be performed for this instruction

CCM Immediate - the address in the CCM where a counter will be loaded from or stored to.

Counter Number - indicates which counter is being loaded, stored, or incremented. Values can range from 0-7.

4.2.1 CNOP

Assembly Language Syntax: CNOP

Description: Do not perform an operation in the Counter Execution Unit

Instruction Format:

36		34	33 26
0	0	0	Unused

Figure 27: CNOP Instruction Format

Example: CNOP

4.2.2 LD - Load a counter from the CCM

Assembly Language Syntax: LD CCM(Imm), Cx

Description: Load the value in the CCM at address Imm into counter x. Note that this value should be the max counter value (2^{32} - 1) - repetition count + 1.

Instruction Format:

36		34	33 29	28 26
0	1	0	CCM Immediate	Counter Number

Figure 28: Counter LD Instruction Format

CCM Immediate - The hexadecimal address in the CCM from which the load will be performed.

Counter Number - The number of the counter that will be loaded. Values can range from 0-7.

Example: LD CCM(0x2), C5
Loads the value in the CCM at location 2 into counter 5.

4.2.3 ST - Store a counter value in the CCM

Assembly Language Syntax: ST Cx, CCM(Imm)

Description: Store the value of counter x in the CCM at location Imm.

Instruction Format:

36		34	33 29	28 26
1	0	1	CCM Immediate	Counter Number

Figure 29: Counter ST Instruction Format

CCM Immediate - The hexadecimal address in the CCM to which the store will be performed.

Counter Number - The number of the counter that will be stored. Values can range from 0-7.

Example:
The ST instruction has an exposed 3-stage pipeline. This means it takes 3 Fast Mode instructions to actually accomplish a store. In the first of these three instructions the destination of the store in the CCM must be specified. The second instruction must have the entire ST instruction specified, and the third instruction must have the source Counter specified.

```
        CNOP    Cx,    CCM(0x2)
        ST      C2,    CCM(0x2)
        CNOP    C2,    CCM(x)
```

This example will store the value of C2 in location 2 of the CCM. Note that since in the first instruction the value of the Counter is not specified, and in the last instruction the location in the CCM is not specified, consecutive stores can be overlapped so that two stores only take 5 instructions instead of 6.

4.2.4 INC - Increment a counter

Assembly Language Syntax: INC Cx

Description: Increment counter x by one.

Instruction Format:

36		34	33 29	28 26
0	1	1	Unused	Counter Number

Figure 30: Counter INC Instruction Format

Counter Number - The number of the counter that will be incremented. Values can range from 0-7.

Example: INC C3
Increment Counter number 3.

4.3 MAR Group

The primary purpose of the MAR (Memory Address Register) Group is to implement the concept of a variable in TSpec. Variables in TSpec are used to represent sequences of memory references that have a constant increment. In a general trace, this sequence could have other types of references interleaved between those with a constant increment, but only those with a constant increment will be described by the variable. A variable consists of a starting address along with its associated tag bits, and an increment value. When a variable is initialized in TSpec, its value is set to the starting address. On each subsequent use of the variable it generates an address corresponding to its current value, and is incremented by its increment value. The MAR Group contains several registers to hold the current value of a variable (32-bits), its tag (8 bits), and its increment (16-bits). Also included is an adder to perform the addition of the increment to the variable each time it is used.

A secondary purpose of the MAR Group is to generate those memory accesses in a reference trace where no pattern has been found and no variable covers them. We think of these accesses as “random” accesses because no pattern has been discovered to describe them. There are two different methods in TGen to generate these type of addresses. The first provides for completely accurate generation of a trace, while the second provides an approximation of a trace. The first is a single FIFO that has stored in sequential order the exact address and tag values of the “random” accesses. This FIFO is accessed by executing a MEMAC from the MAR register mapped to the FIFO. In this way a memory access is generated with the address and tag that are at the head of the FIFO just like a memory access is generated for a variable. The second method of handling “random” accesses, is through two random generators that are mapped to two separate MAR registers. By executing a MEMAC from one of these registers, a random address is formed and a memory access is generated with this address and the tag bits associated with the mapped MAR register. (****Note need to ask Ken how to set up these tag bits - as in what instructions to use - also the random generator initialization and FIFO set up)

This secondary purpose of the MAR group is fulfilled by the two random generators, and the FIFO as shown in Figure 31. The other component of the MAR Group is the Address Control Memory (ACM) that contains the initial values of the variables and their associated tags and increments. The ACM is 32 elements deep and 56-bits wide.

Also present in the MAR Group, but not pictured in Figure 31 is a counter which increments with each write to non-SMC memory. The output of this counter is the data out for each non-SMC memory write.

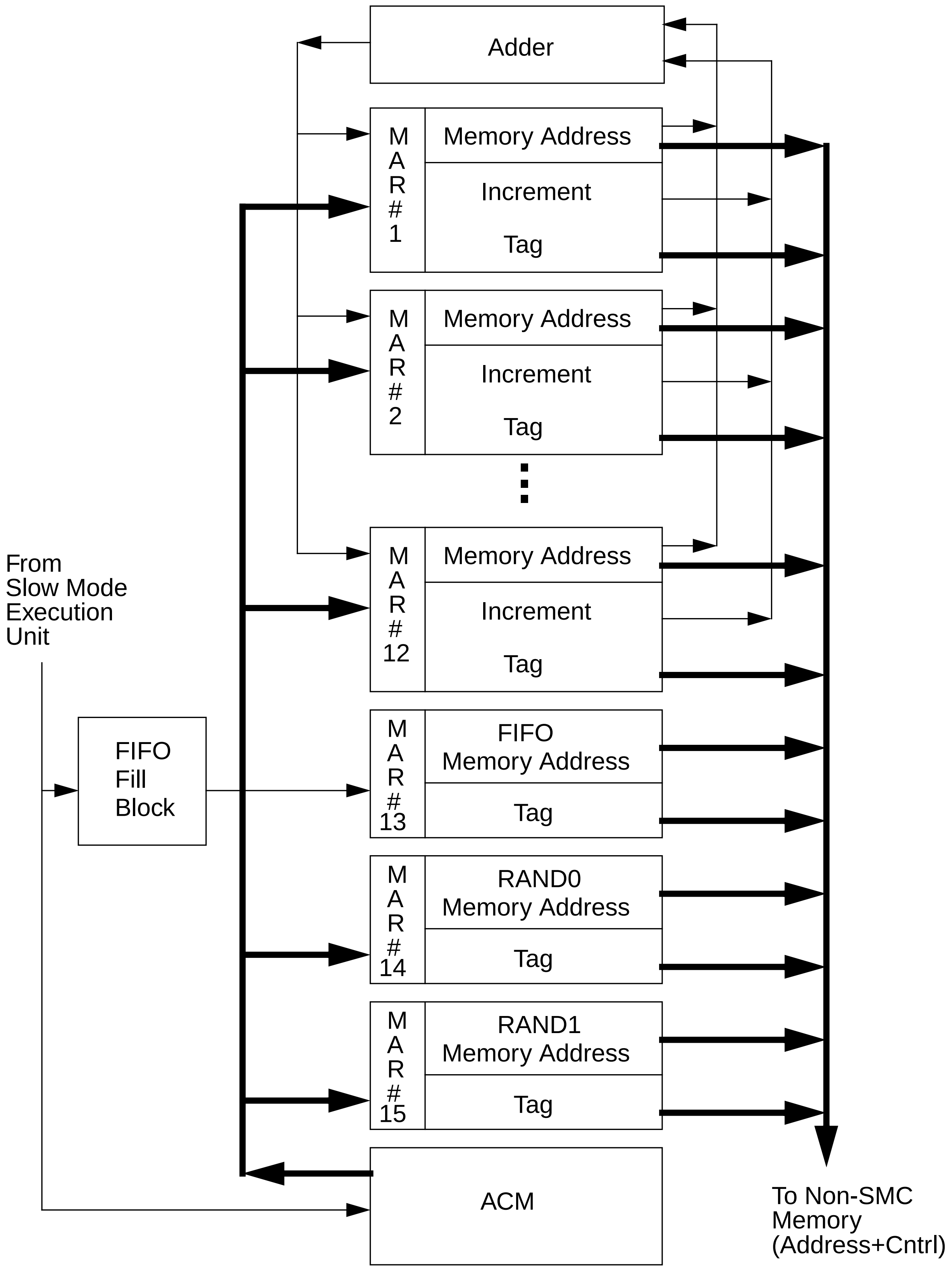

Figure 31: MAR Execution Unit

In the MAR Group, as in the ICIS Group, there are two basic categories of functionality that can take place in parallel. The first is the generation of a memory access to the non-SMC memory system. This can happen in parallel with a load or store of one of the MAR registers as long as the MAR register involved in the memory access is not the same as the MAR register involved in the load or store. The MAR instruction format below reflects this available parallelism.

Instruction Format:

25	24 23	22 18	17 14	13 10
MEMAC	LS Opcode	ACM(Imm)	LS Rx	MEMAC Rx

Figure 32: MAR Instruction Format

MEMAC - one-bit opcode that indicates if a memory access will be generated with this instruction.

LS Opcode - opcode for the load/store unit of the MAR Group.

ACM(Imm) - the hexadecimal address of the source or destination in the ACM of a load or store.

LS Rx - the number of the MAR register being used by the load/store unit.

MEMAC Rx - the number of the MAR register being used by the memory access generation unit.

4.3.1 MNOP - Memory Access NOP

Assembly Language Syntax: MNOP

Description: Do not generate a memory access.

Instruction Format:

25	24 23	22 18	17 14	13 10
0	LS Opcode	ACM(Imm)	LS Rx	Unused

Figure 33: MNOP Instruction Format

Example: MNOP

4.3.2 MEMAC - Memory Access

Assembly Language Syntax: MEMAC Rx

Description: Generate a memory access to the non-SMC memory using MAR x. Note the MAR being used should be initialized from the ACM prior to executing a MEMAC.

Instruction Format:

25	24 23	22 18	17 14	13 10
1	LS Opcode	ACM(Imm)	LS Rx	MEMAC Rx

Figure 34: MAR MEMAC Instruction Format

MEMAC Rx - indicates the number of the MAR register being used to generate the memory access to the non-SMC memory. Values can range from 1 -15. Value 13 indicates the FIFO head. Values 14 and 15 indicate RANDOM0 and RANDOM1 respectively.

Example: MEMAC R1

4.3.3 LSNOP - Load/Store NOP

Assembly Language Syntax: LSNOP

Description: Do not perform a load or store in the MAR Group.

Instruction Format:

25	24	23	22 14	13 10
MEMAC	0	0	Unused	MEMAC Rx

Figure 35: MAR LSNOP Instruction Format

Example: LSNOP

4.3.4 LD - Load

Assembly Language Syntax: LD ACM(Imm), Rx

Description: Load the MAR x with the value in the ACM at location Imm.

Instruction Format:

25	24	23	22 18	17 14	13 10
MEMAC	0	1	ACM(Imm)	LS Rx	MEMAC Rx

Figure 36: MAR LD Instruction Format

ACM(Imm) - the hexadecimal address of the source or destination in the ACM of a load or store.

LS Rx - the number of the MAR register being used by the load/store unit.

Example: LD ACM(0x7), R2

4.3.5 ST - MAR Store

Assembly Language Syntax: ST Rx, ACM(Imm)

Description: Store MAR x in the ACM at location Imm.

Instruction Format:

25	24	23	22 18	17 14	13 10
MEMAC	1	0	ACM(Imm)	LS Rx	MEMAC Rx

Figure 37: MAR ST Instruction Format

ACM(Imm) - the hexadecimal address of the source or destination in the ACM of a load or store.

LS Rx - the number of the MAR register being used by the load/store unit.

Example:
The ST instruction has an exposed 3-stage pipeline. This means it takes 3 Fast Mode instructions to actually accomplish a store. In the first of these three instructions the destination of the store in the ACM must be specified. The second instruction must have the entire ST instruction specified, and the third instruction must have the source MAR specified.

```
MNOP   Rx,   ACM(0x2)
ST     R2,   ACM(0x2)
MNOP   R2,   ACM(x)
```

This example will store the value of R2 in location 2 of the ACM. Note that since in the first instruction the value of the MAR is not specified, and in the last instruction the location in the ACM is not specified, consecutive stores can be overlapped so that two stores only take 5 instructions instead of 6.

4.4 SMC Group

The SMC (Stream Memory Controller) Group is not a part of the standard TGen architecture. It is a specialized execution unit designed to provide a method of stressing this particular addition to a memory system design. Since the concept of the SMC includes having separate instructions to address the SMC FIFOs, this part of TGen simulates the part of the processor that would handle SMC instructions.

The SMC is a dynamic access ordering memory system based on the high-level diagram shown in Figure 38. It is logically divided into two components: a Stream Buffer Unit (SBU), and a Memory Scheduling Unit (MSU). The MSU is a controller through which memory is interfaced to the CPU. It includes logic to issue memory requests and to determine the order of requests during streaming computations. For non-stream accesses, the MSU provides the same functionality and performance as a traditional memory controller.

The MSU has full knowledge of all streams currently needed by the CPU: using the base address, stride, and vector length, it can generate the addresses of all elements in a stream. It also knows the details of the memory architecture, including interleaving and device characteristics. The access-ordering circuitry uses this information to issue requests for individual stream elements in an order that attempts to maximize memory system performance.

The Stream Buffer Unit contains high-speed buffers for stream operands and provides memory-mapped control registers that the processor uses to specify stream parameters. From the processor's perspective, the stream buffers are logically implemented as a set of FIFOs within the SBU, with each stream assigned to one FIFO. The processor references the next element of a stream by referencing the corresponding FIFO head. This implementation of the SMC has six FIFOs - two write FIFOs and four read FIFOs.

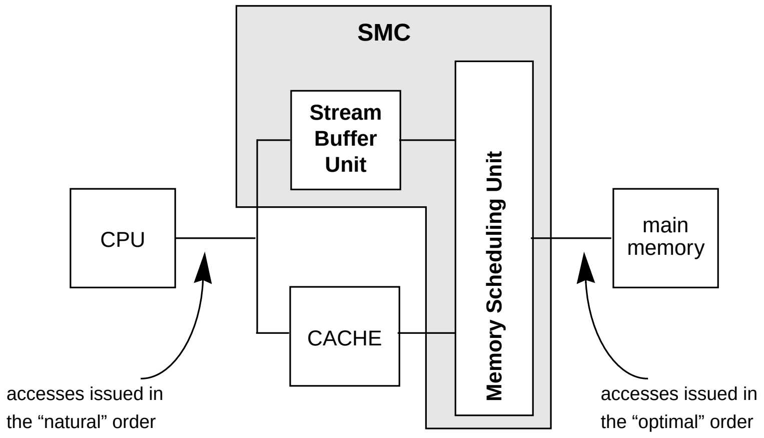
SMC
Stream Buffer Unit
Memory Scheduling Unit
CPU
main memory
CACHE
accesses issued in the "natural" order
accesses issued in the "optimal" order

Figure 38: SMC Architecture

The SMC execution unit is pictured in Figure 39. There are three basic functions supported by this unit. First, the control registers of the FIFOs must be initialized. This is supported by the WCNTRL instructions. Second, the ability to read or write the FIFO head as appropriate is needed. This is supported by the RDF and WRF instructions respectively. Finally, there is a need to be able to reset the entire SMC and also the communication state between the processor and the SMC. These are supported by the RSET instructions. For details on the operations of these instructions, please see the subsections below. All of these operations handled by one basic procedure. The words to be written out to the SMC, whether to control registers or FIFOs, is placed in the SCM in Slow Mode. Then the Fast Mode instructions outlined below are used to actually perform the write. The SCM(Imm) field of the instruction specifies which location in the SCM will be written out and embedded in the opcode field of the assembly language instruction is the address of the FIFO or control register that is being written. The SCM is 37-bits wide by 32 elements deep.

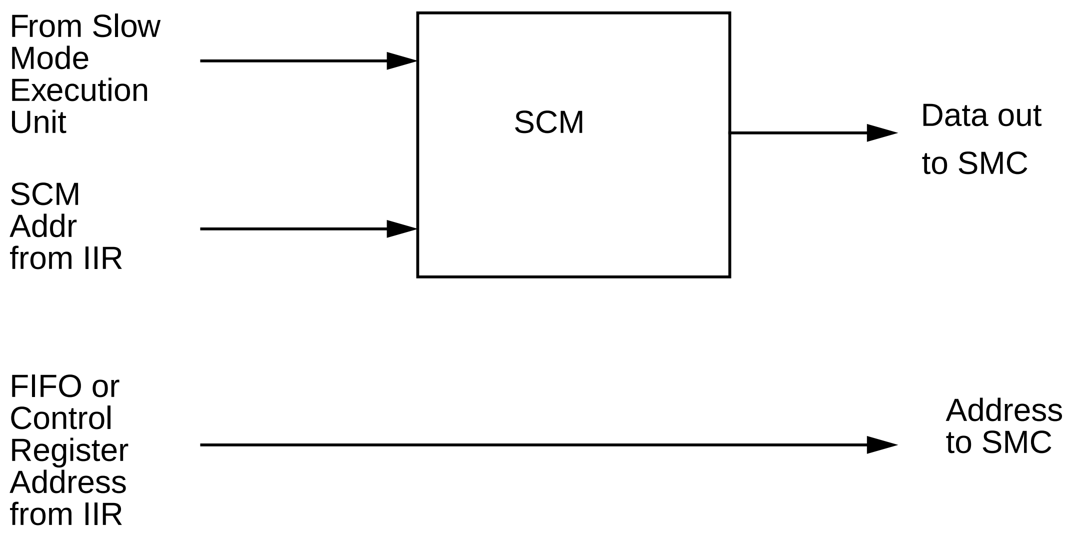
From Slow Mode Execution Unit
SCM Addr from IIR
SCM
Data out to SMC
FIFO or Control Register Address from IIR
Address to SMC

Figure 39: SMC Execution Unit

The general format of the SMC Group instructions is shown below. Note that the indicator of the FIFO, or the control register being dealt with is implicit in the opcode field.

Instruction Format:

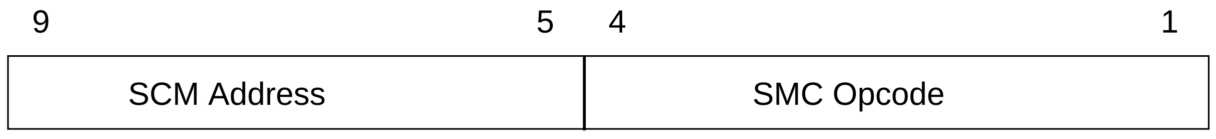

Figure 40: SMC Instruction Format

SCM Address - The hexadecimal address of the data in the SMC Control Memory (SCM) which will be written out to the SMC.

SMC Opcode - The bit-code specifying the operation to be performed by the SMC.

4.4.1 SNOP

Assembly Language Syntax: SNOP

Description: Do nothing in the SMC Group.

Instruction Format:

9 5	4	3	2	1
Unused	1	0	0	0

Figure 41: SMC SNOP Instruction Format

Example: SNOP

4.4.2 RDF - Read FIFO Head

Assembly Language Syntax: RDFx

Description: Read FIFO Head x. Note: No data is actually read into the processor, rather the instruction to do a read is written out to the SMC, it performs the read, holding the processor up and changing the state of the SMC as appropriate.

Instruction Format:

9 5	4	3 1
Unused	1	Fx

Figure 42: SMC RDF Instruction Format

Fx - The number of the FIFO head to be read. Values can range from 1 - 4.

Example: RDF1

4.4.3 WRF - Write FIFO Head

Assembly Language Syntax: WRFx SCM(Imm)

Description: Write FIFO Head 5 or 6 with the data in the SCM at hexadecimal address Imm.

Instruction Format:

9 5	4	3 1
SCM Address	0	Fx

Figure 43: SMC WRF Instruction Format

SCM Address - The hexadecimal address in the SCM of the data that will be written out to the SMC FIFO.

Fx - The FIFO Head which will be written. Values can be:

FIFO Head 5	000
FIFO Head 6	111

Example: WRF5 SCM(0x3)
Write FIFO Head 5 with the data from location 3 in the SCM.

4.4.4 WCNTRL - Write a control register

Assembly Language Syntax: WCNTRL Fx, SCM(Imm)

Description: Write the control register for FIFO x with the data in the SCM at hexadecimal address Imm. Note that the SMC control registers need to have two control words written to them in a row to be completely set up. The write of the second control word initiates action on the part of the SMC. The first write should be a word in SMC Data Format A, and the second write should be a word in SMC Data Format B. See Section 3.2.2 SCM - SMC Control Memory for details on these formats.

Instruction Format:

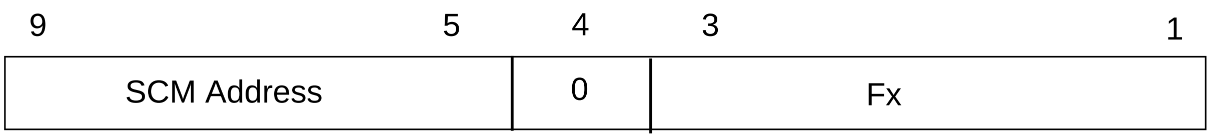

Figure 44: SMC WCNTRL Instruction Format

SCM Address - The hexadecimal address in the SCM of the data that will be written out to the SMC control register.

Fx - The number of the FIFO whose control register should be written. Values can range from 1-6. Recall FIFOs 1-4 are read FIFOs and FIFOs 5-6 are write FIFOs.

Example: WCNTRL F1, SCM(0x2)
WCNTRL F1, SCM(0x3)

4.4.5 RSET S - Reset the SMC

Assembly Language Syntax: RSET S

Description: Completely reset the SMC. ***** Might want to add more details here - talk with Adam

Instruction Format:

9	5	4	3	2	1
Unused		1	1	1	0

Figure 45: RSET S

Example: RSET S

4.4.6 RSET C - Reset SMC Communications

Assembly Language Syntax: RSET C

Description: When writing to the SMC control registers, the SMC expects to get a word in SMC Data Format A first, then one in SMC Data Format B. This order needs to be insured by the programmer. To facilitate this, the RSET C instruction resets the SMC so that it expects a word in SMC Data Format A.

Instruction Format:

9	5	4	3	2	1
Unused		1	1	1	1

Figure 46: RSET C

Example: RSET C

4.5 HALTF and FRZ

There are two instructions that can be executed in parallel with any of the other groups of instructions, but should not be. First is HALTF which halts fast mode execution and returns control to Slow Mode. The second is FRZ which freezes the memory system under test. This freeze and a subsequent Fast Mode halt is done to make “time stand still” for the memory system while TGen goes back to external memory for more data to continue the simulation.

4.5.1 HALTF

Assembly Language Syntax: HALTF

Description: Stop execution in Fast Mode and continue execution in Slow Mode one instruction past the last RUN executed in Slow Mode. Note that 4 consecutive HALTFs must be executed to guarantee the switch to Slow Mode. This is a result of the difference between the clock speeds of the Fast Mode execution units and the Slow Mode execution unit. Also, since it is unclear which of these HALTFs will be recognized, no other

Fast Mode operations should be combined with them.

Instruction Format:

63 54	53	52 37	36 26	25 10	9 1	0
Unused	F	ICIS	Counter	MAR	SMC	1

Figure 47: HALTF Instruction Format

Example: HALTF
HALTF
HALTF
HALTF

4.5.2 FRZ - Freeze the memory system

Assembly Language Syntax: FRZ

Description: Freeze the memory system under test. Note that the memory system is automatically frozen when TGen is in Slow Mode.

Instruction Format:

63 54	53	52 37	36 26	25 10	9 1	0
Unused	1	ICIS	Counter	MAR	SMC	H

Figure 48: FRZ Instruction Format

Example: FRZ

4.6 Instruction Summary

5.0 Examples

6.0 Appendices

6.1 Glossary of Acronyms

EIS - External Instruction Store, contains the program for the processor and is off-chip

MAR Group
MAR Group - Memory Address Register Group, these registers and associated memory are for generating addresses on the memory (SMC) side of the processor

R Registers - hold addresses to be put out on the memory address bus

I Registers - increment registers, hold the increment value for generating sequential address streams on the memory address bus

ACM - Address Control Memory, stores all the potential values of the trace language regular expression variables and their increments for use in a particular section of the program running at high speed.

Counter Group
Counter Group - This set of registers and associated memory allows for loop addresses to be generated. Each counter can keep track of the repetitions of one loop.

CCM - Counter Control Memory, stores all the potential values of the trace language repetition counts for use in a particular section of the program running at high speed.

C Registers - Counter Registers, hold the value of a counter in the Counter Group for use in processing the trace language repetition counts.

Z bits - Overflow/Underflow bits for the C Registers for use in indicating when a repetition count is complete.

ICIS Group
ICIS Group - Internal Control and Instruction Store Group, set of registers and associated memories which implements the subprogram component of control in the trace language.

IIR - Internal Instruction Register, contains the currently executing instruction

IIS - Internal Instruction Store, contains the particular section of the program currently running at high speed, including subprograms

IP Registers - Instruction Pointer Registers, contain pointers into the subprograms that are currently in use.

Next (Return) Register - contains the instruction pointer (IP) to switch to on a return from this subroutine or a pulse (ie the point of control within the “caller” of the subprogram)

Pulse Register/Bit - indicates a particular subprogram associated with this control register is in pulse mode

ICM - Instruction Control Memory, stores all the potential values of the trace language subprogram pointers in use for a particular section of the program running at high speed

ACC Group
ACC Group - Accumulator Group, set of registers and associated memories which implements the data going to and coming from memory (SMC) and/or the External Instruction

Store

EIE Group
EIE Group - External Instruction Execution, set of registers and associated memory which implements “slow mode”. Slow mode is primarily used for loading programs from the External Instruction Store and initializing the different registers in the other architectural groups for high speed execution.

6.2 Software Restrictions for SMC Implementation