digital

Memory Management on the StrongARM SA-110:

An Application Note

Order Number: EC–R4WCA–TE

June 1997

The SA-110 is the first StrongARM implementation of the ARM architecture. This application note provides an overview of memory management followed by details specific to the ARM architecture and the SA-110 implementation itself. The MMU (memory management unit) model for the ARM architecture is described along with its relationship to cache and write buffer control. Behavior of the SA-110 is then discussed and sample code provided for a simple one-to-one mapped virtual-to-physical translation.

Revision/Update Information: This is a new document.

Digital Equipment Corporation
Maynard, Massachusetts
http://www.digital.com/semiconductor

June 1997

Contents

Tables

Figures

1. Introduction

The SA-110 is the first StrongARM implementation of the ARM architecture. This document provides an overview of memory management followed by details specific to the ARM architecture and the SA-110 implementation itself. The MMU (memory management unit) model for the ARM architecture is described along with its relationship to cache and write buffer control. Behavior of the SA-110 is then discussed and sample code provided for a simple one-to-one mapped virtual-to-physical translation.

Remember that the *ARM Systems Architecture Manual* and the *SA-110 Technical Reference Manual* remain the definitive texts for how the device operates, and provide fuller explanations in many cases.

1.1 Memory Management Concepts

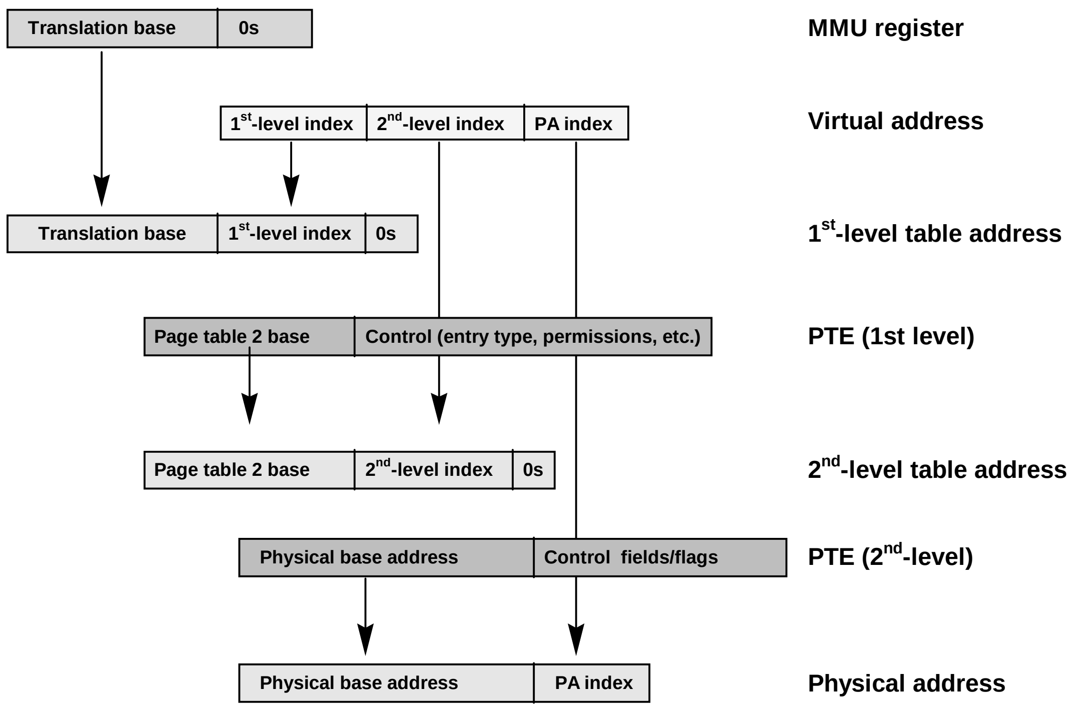
Memory management can be described as the ability to manage the system address space, typically using a blend of software and dedicated hardware. A memory-managed address space as seen by the program is often referred to as a **virtual address** (VA) space, which is then **translated** into a **physical address** (PA) prior to accessing memory or IO.

Memory management provides three functions:

- Access control
- Relocation (address translation)
- Consistent state allowing a faulting access to be corrected and replayed by an exception handler with the same result as a normally completing access. This is the key feature required for a demand-paged memory system.

Translation is performed using **page tables** and may involve multiple steps, each step providing finer granularity on the translated **page size**. Figure 1-1 illustrates a two-level lookup. A predefined translation base is merged with the first-level index to provide the address of a **page table entry** (PTE). The entry contains the base of the second-level table along with any associated control information required by the MMU. Combining this base with the second-level index from the original VA provides an address for the second-level PTE. This entry then provides the physical base, which is concatenated with the PA index to provide the required physical address. Other fields in the PTE are used for access control according to the MMU model.

Figure 1-1 Virtual-to-Physical Address Translation



The process of performing the above translation is commonly known as **table walking**, and may be performed in hardware or software. For performance reasons, microprocessors implement a cache of VA=>PA entries in **translation lookaside buffers** (TLBs) as part of the MMU.

Many schemes can be implemented using the principles outlined. Multitasking systems can implement virtual address spaces on a per-process basis, each with their own page tables, or share a single space across all processes. Protection can be used to prohibit access for both privileged and nonprivileged program execution. **Page faults** result in access aborts where it is up to the exception handler to determine the cause of the fault, and take the appropriate corrective action (for example, map in the requested page from disk to physical memory on an application page miss). The onus is normally on MMU software to keep all the TLBs and translation tables consistent and avoid translation conflicts.

2. The ARM Architecture – MMU

The ARM architecture is currently at Version 4. The evolution through the four versions is summarized in the preface to the *ARM Architecture Reference Manual*. Version 4 of the architecture introduced support for what is known as the Harvard Architecture – a computer architecture model with separate instruction and data paths to memory.

The main features of the ARM MMU architecture are as follows:

- All system architecture functions are controlled by reading or writing an ARM register (Rd) from/to a block of sixteen 32-bit registers (Rn) accessed at coprocessor number (cp_num) 15. Control bits are transferred within the opcode_1, opcode_2, and Rm instruction fields where necessary.

  The instruction format is:

  ```
  MRC/MCR   p<cp_num>, <opcode_1>, Rd, cRn, cRm, <opcode_2>
  ```

- A single set of tables is used for both instruction and data fetches irrespective of whether a Harvard or Von Neumann (unified address and data access stream) architecture is implemented.
- Caches (instruction, data, or unified) are controlled through a combination of the system control coprocessor register and control bits in the page tables.
- 1MB **sections** are supported through a single-level lookup.
- 64KB or 4KB **pages** are supported by a second-level lookup.
- Level 1 page tables consist of 4096, 32-bit entries (16KB table size).
- Level 2 page tables consist of 256, 32-bit entries (1KB table size) for each section. 64KB page table entries are replicated sixteen times each within the table. This is because the level 2 table and PA page indices overlap by four bits in the VA.
- Sections and pages can be protected using **access permissions** (AP bits) with no_access, read_only, or read_write for supervisor and user modes.
- **Domains** can be used to provide an additional level of protection across arbitrary PTEs. Domain settings determine whether access is enabled or not, and if enabled, whether the access permission checks are invoked or bypassed.
- Control bits are provided in the PTEs for enabling caching and write buffering on a section/page basis.

The first- and second-level page table entries are as illustrated in Figure 2-1 and Figure 2-2, respectively.

Figure 2-1 Level 1 Page Table Format

Fault	SBZ								0	0
Page table	Page table base address			SBZ	Domain	IMP			0	1
Section	Section base address	SBZ	AP	SBZ	Domain	IMP	C	B	1	0
Reserved	SBZ								1	1

Figure 2-2 Level 2 Page Table Format

Fault	SBZ								0	0
Large page	Large page base address	SBZ	AP3	AP2	AP1	AP0	C	B	0	1
Small page	Small page base address		AP3	AP2	AP1	AP0	C	B	1	0
Reserved	SBZ								1	1

Apart from the performance impact, translation is transparent to the program until an exception occurs, when the processor will enter **abort mode** and start executing from either the prefetch_abort or data_abort exception vectors. Table 2-1 provides a summary of all exceptions and their priorities.

Table 2-1 Exception Vectors and Their Priorities

Exception Type	Exception Mode	Vector Address	Priority
Reset	SVC	0x00000000	1 (highest)
Undefined instruction	UNDEF	0x00000004	6
SW interrupt (SWI)	SVC	0x00000008	6
Prefetch abort (instruction fetch memory abort)	ABORT	0x0000000C	5
Data abort (data access memory abort)	ABORT	0x00000010	2
IRQ (interrupt)	IRQ	0x00000018	4
FIQ (fast interrupt)	FIQ	0x0000001C	3

The ordering of data abort and FIQ exceptions are to ensure the correct capture of any faulting status that may happen coincident with the fast interrupt. Fast interrupt status is unchanged on entry to the data abort exception, meaning that a FIQ exception (assuming FIQs is enabled) will execute immediately on entry to the data abort handler, should this scenario occur.

2.1 The Coprocessor Interface

The first 9 of 16 coprocessor 15 (cp15) registers are architected as shown in Table 2-2.

Table 2-2 CP15 Register Summary

Register	Reads	Writes	MMU Function
0	ID register	UNPREDICTABLE	–
1	Control	Control	√
2	Translation Table Base	Translation Table Base	√
3	Domain Access Control	Domain Access Control	√
4	UNPREDICTABLE	UNPREDICTABLE	–
5	Fault Status	Fault Status	√
6	Fault Address	Fault Address	√
7	Cache Operations	Cache Operations	√
8	TLB Operations	TLB Operations	√

The control register includes bits for:

- Enabling the MMU
- Enabling address alignment fault checking
- Cache enables – unified/data and instruction (when separate I &D) bits
- Write buffer enable
- System (S) and ROM (R) protection bits.

Please see the SA-110 reference manual for a fuller description and complete listing of this register.

The S and R bits are used in conjunction with the AP bits in the page table entries to resolve the protection for a given mode within a domain according to Table 2-3.

Table 2-3 Access Permissions

AP	S	R	Supervisor_Mode	User_Mode
00	0	0	No Access	No Access
00	1	0	Read only	No Access
00	0	1	Read only	Read Only
00	1	1	UNPREDICTABLE	UNPREDICTABLE
01	x	x	Read/Write	No Access
10	x	x	Read/Write	Read Only
11	x	x	Read/Write	Read/Write

The Domain Access Control Register provides sixteen pairs of control bits, which are used to qualify the domain field in a PTE according to Table 2-4.

Table 2-4 Domain Access Values

Domain_crl	Access	Description
00	No Access	All accesses will cause a domain fault exception
01	Client	Accesses checked against the AP bits (see Table 2-3)
10	Reserved	UNPREDICTABLE
11	Manager	No access permission checks

Data aborts update two cp15 registers. Cp15_6 is updated with the faulting virtual address. Cp15_5 is updated with fault status as summarized in Table 2-5 and Table 2-6.

Table 2-5 Cp15 Fault Status Register Format

[31:9]	[8]	[7:4]	[3:0]
UNPREDICTABLE/SBZ	0	domain	status (FS)

Table 2-6 Fault Status Encoding

Priority	Sources		Domain	FS[3:0]
highest	Terminal exception		invalid	0b0010
	Vector exception		invalid	0b0000
	Alignment		invalid	0b00x1
	Ext. abort on translation	1st level	invalid	0b1100
		2nd level	valid	0b1110
	Translation	section	invalid	0b0101
		page	valid	0b0111
	Domain	section	valid	0b1001
		page	valid	0b1011
	Permission	section	valid	0b1101
		page	valid	0b1111
	Ext. abort on linefetch	section	valid	0b0100
		page	valid	0b0110
lowest	Ext. abort on non-linefetch	section	valid	0b1000
		page	valid	0b1010

Processors may only implement a subset of these encodings.

3. The SA-110 Implementation

The SA-110 has been implemented to V4 of the ARM architecture. It is the first ARM processor to adopt a Harvard Architecture and has the following features:

- A virtually addressed, 16KB instruction cache. The instruction cache is 32-way set associative, with a 32-byte cache line size. No coherence is maintained with main memory. Replacement uses a round-robin algorithm.
- A virtually addressed, 16KB data cache. The data cache is 32-way set associative writeback cache, with a 32-byte cache line size. Each entry has an associated valid bit and two dirty bits, allowing victim writes to be resolved to half lines for more efficient usage of system bandwidth to main memory. All victim writes occur through the write buffer. Data cache entries are only allocated on reads and use a round-robin replacement algorithm. The cache entries also include physical tag bits to allow writebacks without additional address translation.
- Fully associative instruction TLBs. These are updated using a round-robin algorithm.
- Fully associative data TLBs. These are updated using a round-robin algorithm.
- Eight 16-byte entry write buffers. These are used by the writeback cache as well as for handling buffered writes that miss the data cache. Each entry includes physical address, data, and byte mask information.

The SA-110 implements a five-stage pipe.

- Fetch – fetches an instruction from Icache or memory. An Icache miss will stall the fetch stage of the pipeline until the full cacheline (eight memory fetches) is completed. Instruction fetch permission checks occur here and will be flagged, however, the exception will only execute if the instruction enters the decode stage. Branching may mean that the faulting condition never occurs.
- Decode – decodes the instruction and reads input values from the register file.
- Execute – executes shifts and arithmetic operations. Multiplies start in this stage.
- Buffer – data cache or memory accesses. The integer multiplier completes execution in this stage; it retires 12 bits per clock. Results from the execute stage are buffered here, but available via bypasses in most cases. Translation and permission checks occur in this stage for data accesses, and will fault immediately if an exception is generated.
- Writeback – the register file is updated with memory or result data.

All translation occurs automatically once the tables have been set up, the coprocessor registers initialized, and the MMU enabled. Any of the control register functions (cp15_1 register writes) can be enabled in parallel; it is not necessary to perform a separate read-modify-write sequence to enable the MMU. Table walks will generate 32-bit reads from external memory. These reads do **not** check the write buffer, assuming that any updates to the tables have been flushed before translation uses the modified entries.

3.1 MMU and the Caches

The PTEs contain C and B bits for enabling caching and write buffering, respectively. While parts of the memory subsystem can be set up as bufferable but noncachable, the B bit should always be set for cachable D-space. This is because the write buffer is inherently used for writebacks. The SA-110 will automatically buffer cachable writes irrespective of the state of the B bit[1].

3.1.1 Instruction Caching

The Icache is always checked, even when disabled. Permission checks will occur if the MMU is enabled, otherwise, all hits will be taken. The Icache is enabled by setting bit 12 in cp15_1.

If the MMU is enabled, instruction caching is dependent on the state of the C bit in the PTE. If the MMU is disabled, instruction fetches are considered cachable by default; instruction fetches from memory are allocated to an entry if the Icache is enabled.

Instructions can be locked into the Icache by running a program with the Icache enabled, then disabling the Icache to stop any Icache updates. The other option is to bound cachable code in VA space such that no reallocation occurs, or it is minimized to a system-defined subset of the code. Additional code paths are then always fetched from noncachable VA space.

3.1.2 Data Caching

As with the Icache, the Dcache is always checked, even when disabled. It is enabled by setting bit 2 in cp15_1.

The MMU must be enabled and the C bit set for entries to be allocated in the Dcache. Dcache entries can be locked in a similar manner to the Icache, or by disabling the MMU with valid entries in the Dcache. It is not possible to lock portions of the cache.

3.1.3 Cache Management

When the MMU is disabled, the cache TAGs equate to physical addresses. Any mapping changes of virtual to physical addresses must ensure that the caches are flushed appropriately. Remapping can occur when the MMU is enabled, when it is disabled, or while it is enabled. Coprocessor writes to cp15_7 provide mechanisms to manage the caches as summarized in Table 3-1.

[1] The ARM8 uses C=1/B=0 to indicate writethrough caching, and C=B=1 for writeback caching. Please note that this is implementation (not architecturally) defined.

Table 3-1 Cache Control Operations

Function	Opcode_2	Rm	Data
Flush I+D	0b000	0b0111	Ignored
Flush I	0b000	0b0101	Ignored
Flush D	0b000	0b0110	Ignored
Flush Dcache entry	0b001	0b0110	Virtual address
Clean Dcache entry	0b001	0b1010	Virtual address
Drain write buffer	0b100	0b1010	Ignored

The whole Icache is flushed by a single instruction, whereas the Dcache can be flushed collectively or on a single-entry basis. To ensure that memory coherence is maintained, Dcache entries need to be cleaned prior to flushing. A loop is required to clean all Dcache entries. A fetch from a read_only of virtual addresses [2] can be used as an alternative to a coprocessor clean instruction for this purpose. A clean loop needs to be terminated with a drain write buffer command to ensure that all the victims are written to main memory.

Cache flushes may be necessary for several reasons:

- Copying code from ROM to RAM prior to execution.
- Self-modifying code. In this case the modifications will occur as data, but execution will occur from the separate instruction stream.
- Context switches involving changes to the memory map.

[2] Using an assigned virtual address range allows it to be set to the fastest resource on the system bus (with sufficient address range) without tying up that resource. Physical memory can be double mapped to the assigned addresses for this purpose as well as normal usage. The other safe method is to use a genuine read_only area of memory (for example, ROM), however, this may be an extremely slow path. For example, a ROM access may include byte-to-word packing in hardware with its associated timing overhead.

3.2 TLB Management

As with the caches, it is important to flush potentially conflicting entries when a context switch occurs. Four coprocessor write instructions to cp15_8 are provided to manage the TLBs as illustrated in Table 3-2. It is again up to the MMU software to ensure that translation correctness is maintained when PTEs are modified.

Table 3-2 TLB Control Operations

Function	Opcode_2	Rm	Data
Flush I+D	0b000	0b0111	Ignored
Flush I	0b000	0b0101	Ignored
Flush D	0b000	0b0110	Ignored
Flush Dcache entry	0b001	0b0110	Virtual address

3.3 MMU Fault Handling

ARM supports five types of exceptions as summarized in Table 2-1.

- Two levels of interrupt (IRQ and FIQ)
- Memory aborts
- Undefined instruction
- Software interrupts (SWIs used for OS syscalls)

Memory abort is the exception mechanism applicable here. The abort mode may be entered from one of two exception vectors, depending on whether it was an instruction or data fetch that caused the fault. The MMU can generate a memory abort for four reasons.

- An alignment fault on word or halfword loads or stores when two/one least significant address bits are nonzero, respectively.

  Alignment faults are enabled using bit 1 of cp15_1.
- A translation fault when the PTE accessed is marked invalid.
- A domain fault when access is disallowed by the domain protection in the current mode.
- A permission fault when access is disallowed by the access permission (AP) bits in the current mode.

An external abort pin can also be used to cause a data abort for instruction reads, data reads, PTE reads, unbuffered writes, or lock cycles on the system bus.

3.3.1 Prefetch Abort

Prefetch aborts occur as a result of an external abort, translation fault, or protection violation. They are flagged at the fetch stage, but will only cause an exception if it is about to execute. It is up to the prefetch exception handler to recover the faulting address from the link register (R14[value-4] for the SA-110) and use this in conjunction with the relevant cp15 registers to determine the cause of the fault and how to recover. The fault status (cp15_5) or the fault address (cp15_6) are **not** updated on prefetch aborts.

Once the fetch stage has seen an abort, no other instructions will be prefetched until the program counter (PC) has been changed by an exception, branch, or explicit write to R15.

When returning from the prefetch abort fault handler, the following instruction will reload the PC and CPSR, then replay the previously faulting instruction:

```
SUBS PC, R14_abt, #4
```

3.3.2 Data Abort

The cp15 fault status and address registers are used to determine the VA and cause of the abort. The registers can also be written by MCR instructions, which is useful for test and debug purposes. The saved PC for data aborts is the actual_PC+8.

The SA-110 supports only a subset of the architected fault status encodings listed in Table 2-6. The terminal exception is not supported.

When returning from the data abort fault handler, the following instruction will reload the PC and CPSR, then replay the previously faulting data access:

```
SUBS PC, R14_abt, #8
```

4. MMU Initialization and Reset

To enable the MMU, the following steps are necessary:

- Initialize the domain and translation base address registers.
- Initialize the level 1 and level 2 (where appropriate) page tables.
- Enable the MMU (and optionally, alignment faults, WB, and I and D caches).

To disable the MMU:

- Clean the Dcache as appropriate.
- Disable write buffering and the caches.
- Disable the MMU.
- Ensure that all cache entries are flushed where VA address conflicts may exist (valid entries will match as physical addresses when the MMU is disabled).

Note : When enabling or disabling the MMU, up to **three** instructions will be fetched and executed prior to the change taking effect. The actual number is dependent on whether the instructions hit in the Icache or not. Icache hits will propagate down the pipeline while the MCR instruction is executing, and will be drained through the execution path prior to the translation change taking effect. Icache misses will introduce pipeline bubbles that effectively introduce NOPs to the pipeline. This must be accounted for in any MMU management routines where the translated and untranslated addresses differ. It is normal practice to enable the MMU with a direct-mapped (VA = PA) translation, at least for the pages used within the MMU control code. If it is necessary to change these specific pages, program control should enable the MMU, switch to another area of memory, then modify these pages from there. Similar care is required when disabling the MMU, should that be necessary.

If the page with the MMU_enable is changed as part of this step, the MMU enabling/disabling routine must ensure the cp15_1 write instruction is immediately followed by the ITB and Icache flushes, and that they all reside on the same cacheline. This is bad programming practice, which should be avoided for code building/crafting reasons.

4.1 A Worked Example

A worked example is included as Appendix A. This code was written for the EBSA-110, a verification and example design available as an HDK (hardware design kit) through Digital Semiconductor's sales channels.

HDK order number: QR-21A81-11

The reference manual (and possibly more) is also available via the WWW (http://www.digital.com/info/semiconductor/…). The HDK includes the relevant technical documentation, the hardware database (diskettes), and a firmware tree (diskette). The *ARM Systems Architecture Manual* is available as a separately orderable item (order number: EC-QV45A-TE).

The code illustrates the following:

- A 1 ⇔ 1 mapping of VA and PA address spaces
- 1MB sections for 16MB of DRAM (2 SIMM slots)
- 64KB pages for synchronous SRAM, ROM, and FLASH
- Cachable, bufferable access to SRAM and DRAM
- Read_only user access to ROM and FLASH (supervisor's have write access, too)
- Invalid accesses configured for nonsupported memory space
- Simple noncachable, nonbufferable, aliased access to IO space

Note: The code assumes the MMU is disabled. It modifies the base address register before it generates the page table entries.

The code was designed as part of a test harness for running demonstrations and software benchmarks from *demon*, the remote debugger supplied with ARM's Software Development Toolkit (SDT) V2.0*x*. This required an extra level of link register preservation when entering supervisor mode from the demon environment, which should be self-explanatory from the comments in the code.

The reset code used to flush the caches and TBs runs a read loop from an area of VA space reserved specifically for this purpose. The area is mapped to SSRAM because this provides the fastest access path. All other valid VA addresses are direct mapped to the same physical address. The original code (as shipped in the early versions of the HDK) read ROM. This is a very slow path that includes 8-bit to 32-bit packing.

The flushes immediately following the disable command are only guaranteed because the VA and PA translations are the same (as described as described at the beginning of this section).

The code is written in ARM assembler.

Appendix B includes source code definitions plus macro calls for all coprocessor accesses.

A MMU Initialization and Reset

```
;; Memory Management and Cache Initialization Routines for EBSA-110
;
;
;
; History:
;
; V0.1  31-Jan-1996   DB   first draft
; V0.2 11-Apr-1996   DB   update include file references and add conditional assembly options
; V0.3 02-Aug-1996   DB   cleanups and “fast path” clean loop example for appnote release

; Routines which can be conditionally assembled to enable memory management
; with the following cache options:
;
;      no caches enabled
;      no caches with write buffering
;      Icache only
;      Icache only with write buffering
;      Dcache only with write buffering
;      I and Dcache with write buffering
;
;      PLEASE NOTE:
;
;      1)      Dcache with no write buffering is a nonsupported mode
;      2)      Base register updated before tables assumes MMU is DISABLED
;      3)      MMU enabled with S and R bits in CP15_1 both cleared
;              (AP bits govern the access making S & R “don’t cares”)
;      4)      Page tables require to be naturally aligned to their size
;              level1 tables occupy 16KB
;              level2 tabels occupy 1KB
;              this is stricter/more efficient than an alignment restriction to the page size
;
;
;      R0 used as the default scratch data register for these routines
;      R1 used as the default scratch address register for these routines
;
;
;      Arguments:     mmu_init(arg1) - IC/DC/WB enables passed as a value
;                     mmu_reset()    - nil
;

       INCLUDE address_map_h.s
       INCLUDE EBSA_110_defs_h.s


       EXPORT MMU_init
       EXPORT MMU_reset


       KEEP


       AREA MMU_code,CODE,READONLY


              ; write the translation base register
              ; set the domain register: domain0 CLIENT, domain1-15 NO_ACCESS

              ;***PLEASE NOTE*** - reassigning the base register before the tables***
              ;***                  are set up assumes the MMU is DISABLED  ***

              ; set up the page tables - all for domain0, flat address map
              ;      16 x 1MB sections for DRAM - Read/write all
              ;                                 - cachable, bufferable
              ;      2 x 64KB large pages for SSRAM - Read/Write all
              ;                                     - cachable, bufferable
              ;      1MB section for FLASH - Read/Write spvr, Read_only user
              ;                            - cachable, bufferable
              ;      8 x 64KB large pages for ROM - access as per FLASH
              ;                                   - cachable
```

```
                ;       *SPECIAL SECTION* reserved for clean loops
                ;               ***dedicated VA space (1MB @ CLEAN_BASE)
                ;               ***mapped to SSRAM for fastest access
                ;               ***exception to flat map translation rule
                ;       no access for aliased DRAM, SSRAM, FLASH and ROM
                ;       IO map enabled noncachable, nonbufferable incl. aliases

                ; Flush ITB's and DTB's
                ; Flush the Icache
                ; Enable MMU, alignment faulting, and conditionally the Icache,
                ; ...Dcache and Write Buffer

                ;;; next line required for standalone execution only
                ;;;comment out when used with .c files
        ;;ENTRY

                GBLA    CP15_1_MASK

MMU_init        MOV R2, LR      ; save Link Register prior to demon syscall
                                ; needed to ensure the correct return address
                                ; ...in a demon environment
                MRS R1, CPSR    ; need to save the status too!!!

                SWI SWI_EnterOS        ;Demon syscall to switch to spvr mode

                MOV LR,R2       ; reinstate Link Register for return
                MSR SPSR, R1     ; reinstate saved status to the *SPSR* for correct return

                STMDB sp!, {R4-R8}            ; save APCS register variables on the stack

                LDR R1, =(IC_ON + DC_ON + WB_ON)
                AND R0, R0, R1                ;sanitise argument passed to only valid bits
                STMDB sp!, {R0}               ;then save the argument on the stack

                LDR R0, =Level1tab     ;TTB address is 2**14 aligned
                WRCP15_TTBase   R0     ;Initialize Translation Table Base reg.
                LDR R0, =1
                WRCP15_DAControl R0    ;Initialize Domain Access Control
        ;;
        ;;First clear all TT entries - FAULT
        ;;
                LDR R0, =0             ; loop count
                LDR R1, =Level1tab
                LDR R2, =0
                LDR R3, =L1_TABLE_ENTRIES + 2*L2_TABLE_ENTRIES

TTCLRLoop       STR R2, [R1], #4
                ADD R0, R0, #1         ; increment loop count
                CMP R3, R0
                BNE TTCLRLoop
        ;;
        ;;Configure DRAM section accesses
        ;;
                LDR R1, =Level1tab
                LDR R2, =DRAM_SIZE
                LDR R3, =0             ;loop count

DRAMLoop        LDR R0, =DRAM_BASE + DRAM_ACCESS
                ADD R0, R0, R3, LSL #20       ; add section number field
                STR R0, [R1], #4                      ; store TT entry
                ADD R3, R3, #1                ;increment loop count
                CMP R2, R3
                BNE DRAMLoop
        ;;
        ;;Configure CLEAN_LOOP special section access
        ;;
                LDR R1, =Level1tab + CLEAN_BASE:SHR:(20-2)
                LDR R0, =SSRAM_BASE + FLASH_ACCESS    ;;map to SSRAM with
                                                      ;; read_only user space
                STR R0, [R1]
        ;;
        ;;Configure SSRAM section access
```

```
        ;;
                LDR R1, =Level1tab + SSRAM_BASE:SHR:(20-2)
                LDR R0, =Level2tab_SSRAM + L2_CONTROL
                STR R0, [R1]
        ;;
        ;;Configure FLASH section access
                LDR R1, =Level1tab + FLASH_BASE:SHR:(20-2)
                LDR R0, =FLASH_BASE + FLASH_ACCESS
                STR R0, [R1]
        ;;
        ;;Configure ROM section access
        ;;
                LDR R1, =Level1tab + EPROM_BASE:SHR:(20-2)
                LDR R0, =Level2tab_ROM + L2_CONTROL
                STR R0, [R1]
        ;;
        ;;Configure IO section accesses to the end of memory
        ;;
                LDR R1, =Level1tab + IO_BASE:SHR:(20-2)
                LDR R2, =L1_TABLE_ENTRIES/4   ; update top quartile of TT entries
                LDR R3, =0                    ; loop count
IO_Loop LDR R0, =IO_BASE + IO_ACCESS
                ADD R0, R0, R3, LSL #20       ; add section field
                STR R0, [R1], #4                      ; store TT entry
                ADD R3, R3, #1                ; increment loop count
                CMP R2, R3
                BNE IO_Loop
        ;;
        ;;Configure SSRAM large page accesses - 16 aliases per entry
        ;;
                LDR R1, =Level2tab_SSRAM
                LDR R2, =SSRAM_PAGE_COUNT
                LDR R3, =16
                LDR R4, =0                    ; loop count1 (pages)
SSRAMLoop2      LDR R5, =0                    ; loop count2 (aliases)
                LDR R0, =SSRAM_BASE + SSRAM_ACCESS
                ADD R0, R0, R4, LSL #16       ; add page field
SSRAMLoop1      STR R0, [R1], #4                      ; store TT entry
                ADD R5, R5, #1                ; increment alias count
                CMP R3, R5
                BNE SSRAMLoop1        ; large page entry alias loop

                ADD R4, R4, #1                ; increment page count
                CMP R2, R4
                BNE SSRAMLoop2        ; page count loop
        ;;
        ;;Configure ROM large page accesses - 16 aliases per entry
        ;;
                LDR R1, =Level2tab_ROM
                LDR R2, =EPROM_PAGE_COUNT
                LDR R3, =16
                LDR R4, =0                    ; loop count1 (pages)
EPROMLoop2      LDR R5, =0                    ; loop count2 (aliases)
                LDR R0, =EPROM_BASE + EPROM_ACCESS
                ADD R0, R0, R4, LSL #16       ; add page field

EPROMLoop1      STR R0, [R1], #4                      ;store TT entry
                ADD R5, R5, #1                ;increment alias count
                CMP R3, R5
                BNE EPROMLoop1        ;loop aliases

                ADD R4, R4, #1                ;increment page count
                CMP R2, R4
                BNE EPROMLoop2        ;loop for all pages
        ;;
        ;;
        ;;
                WRCP15_FlushITB_DTB R0 ;Flush ITB’s + DTB’s
                WRCP15_FlushIC     R0  ;Flush ICache
        ;;
        ;;Enable MMU, alignment faults and IC/DC/WB as required
```

```
        ;;

                LDMIA sp!, {R0}                ; recover argument from the stack
                LDR R1, =EnableMMU     ; NOTE:  no alignment checks enabled in this
                                       ;        example
                ORR R0, R0, R1
                WRCP15_Control              R0  ; Update control register

                LDMIA sp!, {R4-R8}     ;recover APCS register variables from the stack
                MOVS PC,LR             ; return to user mode
        ;;
        ;;end of MMU config code
        ;;

        ;;; next line used when running this source file as standalone
        ;;      SWI SWI_Exit           ; Halt execution - exit back to demon


        ;; Function used to reset the MMU after a benchmark
        ;; Flushes the Icache, cleans & flushes the Dcache, disabled IC, DC, WB and
        ;; MMU returning the memory system to its powerup state (flat map allows this)
        ;;
        ;; Required by demon to allow multiple loads/execution of tests without resetting
        ;; the debugger.
        ;:
        ;; As with MMU_init, this function requires priveleged mode to execute the
        ;; necessary coprocessor accesses

MMU_reset       MOV R2, LR     ; save Link Register prior to demon syscall
                               ; needed to ensure the correct return address
                               ; ...in a demon environment
                MRS R1, CPSR   ; need to save the status too!!!

                SWI SWI_EnterOS        ;Demon syscall to switch to spvr mode

                MRS R0, CPSR
                ORR R0, R0, #0xC0
                MSR CPSR, R0           ; disable interrupts

                MOV LR,R2      ; reinstate Link Register for return
                MSR SPSR, R1    ; reinstate saved status to the *SPSR* for correct return
                               ; (this will inherently reinstate interrupts on return too)

                ;; First clean the Dcache
                ;; Use reads from ROM to evict any dirty entries

                LDR R0, =CLEAN_BASE    ; address for dcache loads
                ADD R1, R0, #DCACHE_SIZE      ; compare address for dcache clean
                                              ; …completion

CLEANloop       LDR R2, [R0], #DCACHE_LINE    ; load a dcache line and increment address
pointer
                TEQ R1, R0                    ; IF clean still in progress
                BNE CLEANloop          ; THEN loop on dcache fills
                WRCP15_DrainWriteBuffer R0    ; Drain the write buffer
        ;;
        ;; Reset MMU control register
        ;;
                LDR R0, =0
                WRCP15_Control              R0  ; reset the control register in CP15
        ;;
                WRCP15_FlushITB_DTB R0 ;Flush ITB's + DTB's
                WRCP15_FlushIC_DC   R0 ;Flush ICache + Dcache
                NOP
                NOP
                NOP
                NOP                    ; make sure the pipeline clear of any cached entries
                MOVS PC,LR             ; return to user mode
        ;;
        ;;end of MMU config code
```

```
        ;;

        LTORG

        AREA TTentries,DATA,NOINIT,ALIGN=14

Level1tab           % 1:SHL:14          ; 4-byte entries,1MB sections

Level2tab_SSRAM     % 1:SHL:10          ; 4-byte entries, 64KB pages; x16 alias
Level2tab_ROM       % 1:SHL:10          ; 4-byte entries, 64KB pages; x16 alias
exit
        END
```

B Coprocessor Access Macros

```
;;Macros, constants and system variables associated with SA-110 and the EBSA-110 platform
;;=================================================================
;;
;;History:
;;
;; V0.1   02_Feb-1996   DB
;; V0.2   11-Apr-1996   DB    Add demon SWI call number definitions
;; V0.3   12-Apr-1996   DB           Merge definition and coprocessor macro files
;; V0.4 05-Aug-1996   DB      Add CLEAN_BASE for new clean loop code

;;; EBSA-110 platform data
;;; DRAM_SIZE assumes 2 x 8MB SIMM's fitted -

DRAM_SIZE              EQU     16      ; 16MB in 2 x SIMM's
SSRAM_PAGE_COUNT       EQU      2      ; 2 x 64k = 128KB
EPROM_PAGE_COUNT       EQU      8      ; 8 x 64k = 512KB
DCACHE_SIZE            EQU     0x4000  ; 16KB Dcache
DCACHE_LINE            EQU     0x20    ; 32B cache line entry

L2_CONTROL             EQU     0x1     ; domain0, page table pointer

L1_TABLE_ENTRIES       EQU     0x1000  ; 16KB table (word entries)
L2_TABLE_ENTRIES       EQU     0x100   ;  1KB table (word entries)

IO_BASE        EQU     0xC0000000  ; top quartile of address space
CLEAN_BASE             EQU     0x3FF00000  ; reserve VA of last section in bottom quartile

DRAM_ACCESS    EQU     0xC0E   ; AP=11, domain0, C=1, B=1
SSRAM_ACCESS   EQU     0x0FFD  ; AP=11, domain0, C=1, B=1
FLASH_ACCESS   EQU     0x80A   ; AP=10, domain0, C=1, B=0
EPROM_ACCESS   EQU     0x0AA9  ; AP=10, domain0, C=1, B=0
IO_ACCESS              EQU     0xC02   ; AP=11, domain0, C=0, B=0

;;
;; Definitions used in conditional assembly of Icache, Dcache and Write Buffer
;; options
;;

IC_ON                  EQU     0x1000
IC_OFF                 EQU     0x0

DC_ON                  EQU     0x4
DC_OFF         EQU     0x0

WB_ON                  EQU     0x8
WB_OFF         EQU     0x0
```

```
;*********************************************************
;*      Duplicate of information in demon subdirectory's level1_h.s *
;*      (too many redundant dependencies for reuse directly)        *
;*                                                            *
;*                        SWI numbers as used in demon                               *
;*********************************************************

SWI_WriteC                    EQU     &0
SWI_Write0                    EQU     &2
SWI_ReadC                     EQU     &4
SWI_CLI                       EQU     &5
SWI_GetEnv                    EQU     &10
SWI_Exit                      EQU     &11
SWI_EnterOS                   EQU     &16

SWI_GetErrno                  EQU     &60
SWI_Clock                     EQU     &61
SWI_Time                      EQU     &63
SWI_Remove                    EQU     &64
SWI_Rename                    EQU     &65
SWI_Open                      EQU     &66
SWI_Close                     EQU     &68
SWI_Write                     EQU     &69
SWI_Read                      EQU     &6a
SWI_Seek                      EQU     &6b
SWI_Flen                      EQU     &6c

SWI_IsTTY                     EQU     &6e
SWI_TmpNam                    EQU     &6f

SWI_InstallHandler            EQU     &70
SWI_GenerateError             EQU     &71

;;;
;;;
;
;Definitions and Macros for SA-110 Coprocessor Access
;========================================
;

; SA-110 *only* supports Coprocessor number 15
; Only MCR and MRC coprocessor instructions are supported - the others
; ...generate an UNDEFINED exception
;
; CP15 registers are architected as per the ARM V4 architecture spec
;
;       Register0     ID register                   READ_ONLY
;       Register1     Control                       READ_WRITE
;       Register2     Translation Table Base        READ_WRITE
;       Register3     Domain Access Control         READ_WRITE
;       Register4     Reserved
;       Register5     Fault Status                  READ_WRITE
;       Register6     Fault Address                 READ_WRITE
;       Register7     Cache Operations              WRITE_ONLY
;       Register8     TLB Operations                WRITE_ONLY
;       Register9-14  Reserved
;       Register15    SA-110 specific tst/clk/idle  WRITE_ONLY

;; Bit definitions for the control register:
;;
;; enables are logically OR'd with the control register
;; use bit clears (BIC's) to disable functions
;;      *** all bits cleared on RESET ***
;;
EnableMMU             EQU     0x1
EnableAlignFault      EQU     0x2
EnableDcache          EQU     0x4
EnableWB              EQU     0x8
EnableBigEndian       EQU     0x80
EnableMMU_S           EQU     0x100          ; selects MMU access checks
EnableMMU_R           EQU     0x200          ; selects MMU access checks
```

```
EnableIcache           EQU    0x1000

;; Defined Macros:
;;
;RDCP15_ID                    Rx     read of ID register
;RDCP15_Control               Rx     read of Control register
;WRCP15_Control               Rx     write of Control register
;RDCP15_TTBase        Rx     read of Translation Table Base reg.
;WRCP15_TTBase        Rx     write of Translation Table Base reg.
;RDCP15_DAControl             Rx     read of Domain Access Control reg.
;WRCP15_DAControl             Rx     write of Domain Access Control reg.
;RDCP15_FaultStatus           Rx     read of Fault Status register
;WRCP15_FaultStatus           Rx     write of Fault Status register
;RDCP15_FaultAddress          Rx     read Fault Address register
;WRCP15_FaultAddress          Rx     write of Fault Address register
;WRCP15_FlushIC_DC            Rx     cache control   - Flush ICache + DCache
;;                                   Rx redundant but rqud for MACRO
;WRCP15_FlushIC               Rx     cache control   - Flush ICache
;;                                   Rx redundant but rqud for MACRO
;WRCP15_FlushDC               Rx     cache control   - Flush DCache
;;                                   Rx redundant but rqud for MACRO
;WRCP15_CacheFlushDentry      Rx     cache control   - Flush DCache entry,
                                     Rx source for VA
;WRCP15_CleanDCentry          Rx     cache control   - Clean DCache entry,
                                     Rx source for VA
;WRCP15_Clean_FlushDCentry    Rx     cache control   - Clean + Flush DCache entry,
                                     Rx source for VA
;WRCP15_DrainWriteBuffer      Rx     Drain Write Buffer
;;                                   Rx redundant but rqud for MACRO
;WRCP15_FlushITB_DTB  Rx     TLB control     - Flush ITB's + DTB's
;;                                   Rx redundant but rqud for MACRO
;WRCP15_FlushITB              Rx     TLB control     - Flush ITB's
;;                                   Rx redundant but rqud for MACRO
;WRCP15_FlushDTB              Rx     TLB control     - Flush DTB's
;;                                   Rx redundant but rqud for MACRO
;WRCP15_FlushDTBentry Rx     TLB control     - Flush DTB entry, Rx source for VA
;WRCP15_EnableClockSW Rx     test/clock/idle control - Enable Clock Switching
;;                                   Rx redundant but rqud for MACRO
;WRCP15_DisableClockSW Rx    test/clock/idle control - Disable Clock Switching
;;                                   Rx redundant but rqud for MACRO
;WRCP15_DisablenMCLK  Rx     test/clock/idle control - Disable nMCLK output
;;                                   Rx redundant but rqud for MACRO
;WRCP15_WaitInt               Rx     test/clock/idle control - Wait for Interrupt
;;                                   Rx redundant but rqud for MACRO

;Coprocessor read of ID register
;
    MACRO
    RDCP15_ID $reg_number
    MRC p15, 0, $reg_number, c0, c0 ,0
    MEND

;Coprocessor read of Control register
;
    MACRO
    RDCP15_Control $reg_number
    MRC p15, 0, $reg_number, c1, c0 ,0

    MEND

;Coprocessor write of Control register
;
    MACRO
    WRCP15_Control $reg_number
    MCR p15, 0, $reg_number, c1, c0 ,0

    MEND

;Coprocessor read of Translation Table Base reg.
;
    MACRO
    RDCP15_TTBase $reg_number
    MRC p15, 0, $reg_number, c2, c0 ,0

    MEND
```

```
;Coprocessor write of Translation Table Base reg.
;
    MACRO
    WRCP15_TTBase $reg_number
    MCR p15, 0, $reg_number , c2, c0 ,0

    MEND

;Coprocessor read of Domain Access Control reg.
;
    MACRO
    RDCP15_DAControl $reg_number
    MRC p15, 0, $reg_number, c3, c0 ,0

    MEND

;Coprocessor write of Domain Access Control reg.
;
    MACRO
    WRCP15_DAControl $reg_number
    MCR p15, 0, $reg_number, c3, c0 ,0

    MEND

;Coprocessor read of Fault Status register
;
    MACRO
    RDCP15_FaultStatus $reg_number
    MRC p15, 0, $reg_number, c5, c0 ,0

    MEND

;Coprocessor write of Fault Status register
;
    MACRO
    WRCP15_FaultStatus $reg_number
    MCR p15, 0, $reg_number, c5, c0 ,0

    MEND

;Coprocessor read of Fault Address register
;
    MACRO
    RDCP15_FaultAddress $reg_number
    MRC p15, 0, $reg_number, c6, c0 ,0

    MEND

;Coprocessor write of Fault Address register
;
    MACRO
    WRCP15_FaultAddress $reg_number
    MCR p15, 0, $reg_number, c6, c0 ,0

    MEND

;Coprocessor cache control
;Flush ICache + DCache
;
    MACRO
    WRCP15_FlushIC_DC $reg_number
    MCR p15, 0, $reg_number, c7, c7 ,0

    MEND

;Coprocessor cache control
;Flush ICache
;
    MACRO
    WRCP15_FlushIC $reg_number
    MCR p15, 0, $reg_number, c7, c5 ,0

    MEND
```

```
;Coprocessor cache control
;Flush DCache
;
    MACRO
    WRCP15_FlushDC $reg_number
    MCR p15, 0, $reg_number, c7, c6 ,0

    MEND

;Coprocessor cache control
;Flush DCache entry
;
    MACRO
    WRCP15_CacheFlushDentry $reg_number
    MCR p15, 0, $reg_number, c7, c6 ,1

    MEND

;Coprocessor cache control
;Clean DCache entry
;
    MACRO
    WRCP15_CleanDCentry $reg_number
    MCR p15, 0, $reg_number, c7, c10 ,1

    MEND

;Coprocessor cache control
;Clean + Flush DCache entry
;
    MACRO
    WRCP15_Clean_FlushDCentry $reg_number
    MCR p15, 0, $reg_number, c7, c14 ,1

    MEND

;Coprocessor Drain Write Buffer
;
    MACRO
    WRCP15_DrainWriteBuffer $reg_number
    MCR p15, 0, $reg_number, c7, c10 ,4

    MEND


;Coprocessor TLB control
;Flush ITB + DTB
;
    MACRO
    WRCP15_FlushITB_DTB $reg_number
    MCR p15, 0, $reg_number, c8, c7 ,0

    MEND

;Coprocessor TLB control
;Flush ITB
;
    MACRO
    WRCP15_FlushITB $reg_number
    MCR p15, 0, $reg_number, c8, c5 ,0

    MEND

;Coprocessor TLB control
;Flush DTB
;
    MACRO
    WRCP15_FlushDTB $reg_number
    MCR p15, 0, $reg_number, c8, c6 ,0

    MEND
```

```
;Coprocessor TLB control
;Flush DTB entry

    MACRO
    WRCP15_FlushDTBentry $reg_number
    MCR p15, 0, $reg_number, c8, c6 ,1

    MEND

;Coprocessor test/clock/idle control
;Enable Clock Switching
;
    MACRO
    WRCP15_EnableClockSW $reg_number
    MCR p15, 0, $reg_number, c15, c1 ,2

    MEND

;Coprocessor test/clock/idle control
;Disable Clock Switching
;
    MACRO
    WRCP15_DisableClockSW $reg_number
    MCR p15, 0, $reg_number, c15, c2 ,2

    MEND

;Coprocessor test/clock/idle control
;Disable nMCLK output
;
    MACRO
    WRCP15_DisablenMCLK $reg_number
    MCR p15, 0, $reg_number, c15, c4 ,2

    MEND

;Coprocessor test/clock/idle control
;Wait for Interrupt
;
    MACRO
    WRCP15_WaitInt $reg_number
    MCR p15, 0, $reg_number, c15, c8 ,2

    MEND
    END
```

Support, Products, and Documentation

If you need technical support, a *Digital Semiconductor Product Catalog*, or help deciding which documentation best meets your needs, visit the Digital Semiconductor World Wide Web Internet site:

http://www.digital.com/semiconductor

You can also call the Digital Semiconductor Information Line or the Digital Semiconductor Customer Technology Center. Please use the following information lines for support.

For documentation and general information:	
Digital Semiconductor Information Line	
United States and Canada:	1-800-332-2717
Europe:	+44 118 920 3972
Other Countries:	1-510-490-4753
Electronic mail address:	semiconductor@digital.com

For technical support:	
Digital Semiconductor Customer Technology Center	
Phone (U.S. and international):	1-508-568-7474
Fax:	1-508-568-6698
Electronic mail address:	ctc@hlo.mts.dec.com

Digital Semiconductor Products

To order StrongARM-related and other Digital Semiconductor products, contact your local distributor. The following tables list some of the products available from Digital Semiconductor:

Chips	Order Number
Digital Semiconductor 21285 SA-110 core logic chip	21285 –AA

Evaluation board kits include an evaluation board, and can include a complete design kit, an installation kit, or an accessories kit.

Evaluation Board Kits	Order Number
EBSA-285 (SA-110/21285) evaluation board kit	21A85 –01
EBSA-BPL-5V PCI development backplane – 5V signalling	21A85 –02

EBSA-BPL-3V3 PCI development backplane signalling	– 3.3V	21A85 –03

Design kits include full documentation and schematics. They do not include evaluation boards or related hardware.

Design Kits	Order Number
SA-110/21285 Hardware Developer's Kit	QR–21A85 11

Digital Semiconductor Documentation

The following table lists some of the available Digital Semiconductor documentation.

Title	Order Number
Digital Semiconductor 21285 Core Logic for SA-110 Microprocessor Data Sheet	EC–R4CHA –TE
Digital Semiconductor SA-110 Microprocessor Technical Reference Manual	EC–QPWLC –TE
The StrongARM/ARM Technical Document Kit (includes the SA-110 Technical Reference Manual and the ARM Architecture Reference Manual [1])	QR–ARMKT –TE

[1] Also available from independent booksellers: publisher, Prentice-Hall; ISBN 0 13 736299 4.

Third-Party Documentation

You can order the following third-party documentation directly from the vendor:

Title	Vendor
PCI Local Bus Specification, Revision 2.1 PCI Multimedia Design Guide, Revision 1.0 PCI System Design Guide PCI –to–PCI Bridge Architecture Specification, Revision 1.0 PCI BIOS Specification, Revision 2.1	PCI Special Interest Group U.S. 1 –800 –433 –5177 International 1 –503 –797 –4207 FAX 1 –503 –234 –6762
IEEE title	The Institute of Electrical and Electronics Engineers, Inc. U.S. 1 –800 –701 –4333 International 1 –908 –981 –0060 FAX 1 –908 –981 –9667