AmigaMail

COLLABORATORS

	TITLE : AmigaMail	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 23, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AmigaMail

1.1 III-11: 68040 Compatibility Warning

by Michael Sinz

> editor's note: the new exec.library routines referred to in this article are part of the 2.04 OS release.

Now that Motorola's 68040ä CPU is available, it will not be long before it appears in Amiga-based products. In fact, many Amiga magazines already have ads for 3rd party 68040 CPU cards.

The 68040 is a much more powerful CPU than its predecessors. It has 4K of cache memory for instructions and another 4K cache for data. The reason for these two separate caches is so that the CPU core can access data and CPU instructions at the same time.

Although the 68040 provides greater performance it also brings with it greater compatibility problems. Just the fact that the caches are so much larger than Motorola's 68030 CPU can cause problems. However, this is not its biggest obstacle.

The 68040 data cache has a mode that can make the system run much faster in most cases. It is called CopyBack mode. When a program writes data to memory in this mode, the data goes into the cache but not into the physical RAM. That means that if a program or a piece of hardware were to read that RAM without going through the data cache on the 68040, it will read old data. CopyBack mode effects two areas of the Amiga: DMA devices and the CPU's instruction reading.

CopyBack mode effects DMA devices because they read and write data directly to memory. Using DMA with CopyBack mode requires a cache flush. If a DMA device needs to read RAM via DMA, it must first make sure that data in the caches has been written to memory. It can do this by calling the Exec function CachePreDMA(). If a DMA device is about to write to memory, it should call CachePreDMA() before the write, do the DMA write, and then call CachePostDMA(), which makes sure that the CPU uses the data just written to memory.

An added advantage of using the CachePreDMA() and CachePostDMA() functions

is that they give the OS the chance to tell the DMA device that the physical addresses and memory sizes are not the same. This will make it possible in the future to add features such as virtual memory. See the Autodocs for more information on these calls (the Autodocs have been included in this article). The other major compatibility problem with the 68040's CopyBack mode is with fetching CPU instructions. CPU instructions have to be loaded into memory so the CPU can copy them into its instruction cache. Normally, instructions that will beexecuted are written to memory by the CPU (i.e. loading a program from disk). In CopyBack mode, anything the CPU writes to memory, including CPU instructions, doesn't actually go into memory, it goes into the data cache. If instructions are not flushed out of the data cache to RAM, the 68040 will not be able to find them when it tries to copy them into the instruction cache for execution. It will instead find and attempt to execute whatever garbage data happened to be left at that location in RAM.

To remedy this, any program that writes instructions to memory must flush the data cache after writing. The V37 Exec function CacheClearU() takes care of this. Release 2.0 of the Amiga OS correctly flushes the caches as needed after it does the LoadSeg() of a program (LoadSeg() loads Amiga executable programs into memory from disk). Applications need to do the same if they write code to memory. One such example was the article ''Creating Multiple Processes with Re-entrant Code'' from the November/December 1989 issue of Amiga Mail (Volume I). For that example to work with the 68040's CopyBack mode, it needs to flush the cache before it asks the system to execute the code it wrote to memory (which may still be in the cache). It can do that by calling CacheClearU() before the call to CreateProc(). In C that would be:

```
extern struct ExecBase *SysBase;


    . . .

    /* If we are in 2.0, call CacheClearU() before CreateProc() */
    if (SysBase->LibNode.lib_Version >= 37) CacheClearU();

    /* Now do the CreateProc() call... */
    proc=CreateProc(... /* whatever your call is like */ ...);

    . . .
```

For those of you programming in assembly:

```
*****************************************************************************
* Check to see if we are running in V37 ROM or better.  If so,
* we want to call CacheClearU() to make sure we are safe on future
* hardware such as the 68040.  This section of code assumes that
* a6 points at ExecBase.  a0/a1/d0/d1 are trashed in CacheClearU()
*
            cmpi.w  #37,LIB_VERSION(a6)     ; Check if exec is >= V37
            bcs.s   TooOld                  ; If less than V37, too old...
            jsr     _LVOCacheClearU(a6)     ; Clear the cache...
TooOld:
*
```

****

Note that CreateProc() is not the only routine where CopyBack mode could be a problem. Any program that copies code into memory for execution that is not done via LoadSeg() (for example, ``Using SetFunction() in a Debugger'' from the March/April 1991 issue of Amiga Mail), you will need to call CacheClearU(). Many input.device handlers have been known to allocate and copy the handler code into memory and then exit back to the system. These programs also need to have this call in them. The above code will work under pre-2.0 versions of the OS, and will do the correct operations in 2.04 (and beyond).

Heeding these compatibility guidelines will allow the system to enable CopyBack mode, enhancing the performance of the Amiga considerably. Ignoring these guidelines will prevent the Amiga from taking advantage of CopyBack mode and possibly other advanced features of the 68040 (and beyond).

CacheControl CacheClearU CachePreDMA
CacheClearE CachePostDMA

1.2 CacheControl

exec.library/CacheControl exec.library/CacheControl

NAME
CacheControl - Instruction & data cache control

SYNOPSIS
oldBits = CacheControl(cacheBits,cacheMask)
D0 D0 D1

ULONG CacheControl(ULONG,ULONG);

FUNCTION
This function provides global control of any instruction or data caches that may be connected to the system. All settings are global -- per task control is not provided.

The action taken by this function will depend on the type of CPU installed. This function may be patched to support external caches, or different cache architectures. In all cases the function will attempt to best emulate the provided settings. Use of this function may save state specific to the caches involved.

The list of supported settings is provided in the exec/execbase.i include file. The bits currently defined map directly to the Motorola 68030 CPU CACR register. Alternate cache solutions may patch into the Exec cache functions. Where possible, bits will be interpreted to have the same meaning on the installed cache.

INPUTS
cacheBits - new values for the bits specified in cacheMask.

```
    cacheMask - a mask with ones for all bits to be changed.

RESULT
    oldBits   - the complete prior values for all settings.

NOTE
    As a side effect, this function clears all caches.

    Selected bit definitions for Cache manipulation calls from
    <exec/execbase.i>

    BITDEF  CACR,EnableI,0          ;Enable instruction cache
    BITDEF  CACR,FreezeI,1          ;Freeze instruction cache
    BITDEF  CACR,ClearI,3           ;Clear instruction cache
    BITDEF  CACR,IBE,4              ;Instruction burst enable
    BITDEF  CACR,EnableD,8          ;68030 Enable data cache
    BITDEF  CACR,FreezeD,9          ;68030 Freeze data cache
    BITDEF  CACR,ClearD,11          ;68030 Clear data cache
    BITDEF  CACR,DBE,12             ;68030 Data burst enable
    BITDEF  CACR,WriteAllocate,13   ;68030 Write-Allocate mode (must
                                    ;always be set)
    BITDEF  CACR,EnableE,30
*
*   With this bit set, the external cache states then match the internal
*   cache states.  That is, if the internal data cache is turned on,
*   the external data cache is turned on.  Same for instruction caches.
*   If the external cache only exists as a single unified cache,
*   this will turn on if the data cache is on.  (This is done for
*   compatibility reasons)  With this bit clear, the external caches
*   are turned off.

    BITDEF  CACR,CopyBack,31    ;Master enable for copyback caches


    BITDEF  DMA,Continue,1      ;Continuation flag for CachePreDMA
    BITDEF  DMA,NoModify,2      ;Set if DMA does not update memory


SEE ALSO
    exec/execbase.i, CacheClearU, CacheClearE
```

1.3 CacheClearE

```
exec.library/CacheClearE                          exec.library/CacheClearE

NAME
    CacheClearE - Cache clearing with extended control (V37)

SYNOPSIS
    CacheClearE(address,length,caches)
                a0      d0     d1

    void CacheClearE(APTR,ULONG,ULONG);

FUNCTION
```

```
    Flush out the contents of the CPU instruction and/or data caches.
    If dirty data cache lines are present, push them to memory first.

    Motorola CPUs have separate instruction and data caches.  A data
    write does not update the instruction cache.  If an instruction is
    written to memory or modified, the old instruction may still exist
    in the cache.  Before attempting to execute the code, a flush of
    the instruction cache is required.

    For most systems, the data cache is not updated by Direct Memory
    Access (DMA), or if some external factor changes shared memory.

    Caches must be cleared after *any* operation that could cause
    invalid or stale data.  The most common cases are DMA and modifying
    instructions using the processor.

    Some examples:
                    Self modifying code
                    Building Jump tables
                    Run-time code patches
                    Relocating code for use at different addresses.
                    Loading code from disk

INPUTS
    address - Address to start the operation.  This may be rounded
              due to hardware granularity.
    length  - Length of area to be cleared, or $FFFFFFFF to indicate all
              addresses should be cleared.
    caches  - Bit flags to indicate what caches to affect.  The current
              supported flags are:
                    CACRF_ClearI    ;Clear instruction cache
                    CACRF_ClearD    ;Clear data cache
              All other bits are reserved for future definition.

NOTES
    On systems with a copyback mode cache, any dirty data is pushed
    to memory as a part of this operation.

    Regardless of the length given, the function will determine the most
    efficient way to implement the operation.  For some cache systems,
    including the 68030, the overhead partially clearing a cache is often
    too great.  The entire cache may be cleared.

    For all current Amiga models, Chip memory is set with Instruction
    caching enabled, data caching disabled.  This prevents coherency
    conflicts with the blitter or other custom chip DMA.  Custom chip
    registers are marked as non-cacheable by the hardware.

    The system takes care of appropriately flushing the caches for normal
    operations.  The instruction cache is cleared by all calls that
    modify instructions, including LoadSeg(), MakeLibrary() and
    SetFunction().

SEE ALSO
    exec/execbase.i, CacheControl, CacheClearU
```

1.4 CacheClearU

```
exec.library/CacheClearU                          exec.library/CacheClearU

NAME
    CacheClearU - User callable simple cache clearing (V37)

SYNOPSIS
    CacheClearU()

    void CacheClearU();

FUNCTION
    Flush out the contents of any CPU instruction and data caches.
    If dirty data cache lines are present, push them to memory first.

    Caches must be cleared after *any* operation that could cause
    invalid or stale data.  The most common cases are DMA and modifying
    instructions using the processor.  See the CacheClearE() autodoc
    for a more complete description.

    Some examples of when the cache needs clearing:
                    Self modifying code
                    Building Jump tables
                    Run-time code patches
                    Relocating code for use at different addresses.
                    Loading code from disk

SEE ALSO
    exec/execbase.i, CacheControl, CacheClearE
```

1.5 CachePostDMA

```
exec.library/CachePostDMA                        exec.library/CachePostDMA

NAME
    CachePostDMA - Take actions after to hardware DMA  (V37)

SYNOPSIS
    CachePostDMA(vaddress,&length,flags)
                 a0       a1      d0

    CachePostDMA(APTR,LONG *,ULONG);

FUNCTION
    Take all appropriate steps after Direct Memory Access (DMA).  This
    function is primarily intended for writers of DMA device drivers.  The
    action will depend on the CPU type installed, caching modes, and the
    state of any Memory Management Unit (MMU) activity.

    As implemented
            68000 - Do nothing
            68010 - Do nothing
            68020 - Do nothing
```

```
            68030 - Flush the data cache
            68040 - Flush matching areas of the data cache
            ????? - External cache boards, Virtual Memory Systems, or
                    future hardware may patch this vector to best emulate
                    the intended behavior.
                    With a Bus-Snooping CPU, this function my end up
                    doing nothing.

INPUTS
    address - Same as initially passed to CachePreDMA
    length  - Same as initially passed to CachePreDMA
    flags   - Values:
                    DMA_NoModify - If the area was not modified (and
                    thus there is no reason to flush the cache) set
                    this bit.

SEE ALSO
    exec/execbase.i, CachePreDMA, CacheClearU, CacheClearE
```

1.6 CachePreDMA

```
exec.library/CachePreDMA                          exec.library/CachePreDMA

NAME
    CachePreDMA - Take actions prior to hardware DMA  (V37)

SYNOPSIS
    paddress = CachePreDMA(vaddress,&length,flags)
    d0                     a0       a1      d0

    APTR CachePreDMA(APTR,LONG *,ULONG);

FUNCTION
    Take all appropriate steps before Direct Memory Access (DMA).  This
    function is primarily intended for writers of DMA device drivers.  The
    action will depend on the CPU type installed, caching modes, and the
    state of any Memory Management Unit (MMU) activity.

    This function supports advanced cache architectures that have
    "copyback" modes.  With copyback, write data may be cached, but not
    actually flushed out to memory.  If the CPU has unflushed data at the
    time of DMA, data may be lost.

    As implemented
            68000 - Do nothing
            68010 - Do nothing
            68020 - Do nothing
            68030 - Do nothing
            68040 - Write any matching dirty cache lines back to memory.
                    As a side effect of the 68040's design, matching data
                    cache lines are also invalidated -- future CPUs may
                    be different.
            ????? - External cache boards, Virtual Memory Systems, or
                    future hardware may patch this vector to best emulate
                    the intended behavior.
```

```
                    With a Bus-Snooping CPU, this function my end up
                    doing nothing.

INPUTS
    address - Base address to start the action.
    length  - Pointer to a longword with a length.
    flags   - Values:
                    DMA_Continue - Indicates this call is to complete
                    a prior request that was broken up.

RESULTS
    paddress- Physical address that coresponds to the input virtual
              address.
    &length - This length value will be updated to reflect the contiguous
              length of physical memory present at paddress.  This may
              be smaller than the requested length.  To get the mapping
              for the next chunk of memory, call the function again with
              a new address, length, and the DMA_Continue flag.

NOTE
    Due to processor granularity, areas outside of the address range
    may be affected by the cache flushing actions.  Care has been taken
    to ensure that no harm is done outside the range, and that activities
    on overlapping cache lines won't harm data.

SEE ALSO
    exec/execbase.i, CachePostDMA, CacheClearU, CacheClearE
```