

## Identifying the Amiga's CPU

by Dave Haynie

Amiga's come in a variety of configurations including models that use the Motorola 68000, the 68020 and now the 68030. While it's not that difficult to figure out which CPU you're using by trying certain instructions and trapping the exception if they break, this isn't necessary. In fact, the Amiga OS sets an ExecBase flag on system startup that can identify which processor (and coprocessor) is installed. The CPU type flags are stored in ExecBase->AttnFlags.

Under V1.3 and earlier versions of the OS, the system can't tell the difference between a 68020 and a 68030, or a 68881 and a 68882. That's no big surprise, since both the 68030 and 68882 were introduced well after the last version of the OS was released. Under the new V2.0 version of the operating system, support for identifying these processors has been added. If you are running under V2.0 of the operating system, you can just look at ExecBase->AttnFlags to find out what processor is installed. The AttnFlags field will be set as follows:

```
/****** V2.0 Bit defines for AttnFlags *****/

/* Processors and Co-processors: */
#define AFB_68010 0 /* also set for 68020 */
#define AFB_68020 1 /* also set for 68030 */
#define AFB_68030 2 /* New flag under V2.0 */
#define AFB_68040 3 /* New flag under V2.0 */
#define AFB_68881 4 /* also set for 68882 */
#define AFB_68882 5 /* New flag under V2.0 */

#define AFF_68010 (1L<<0)
#define AFF_68020 (1L<<1)
#define AFF_68030 (1L<<2) /* New flag under V2.0 */
#define AFF_68040 (1L<<3) /* New flag under V2.0 */
#define AFF_68881 (1L<<4)
#define AFF_68882 (1L<<5) /* New flag under V2.0 */
```

However, if you are running under V1.3 or earlier versions of the operating system, this method will not work with the 68030 and 68882 processors. To overcome this problem you can use the code shown below to identify which CPU the system is using under V1.3. There are three functions to link with your code to identify the processor and coprocessor: GetCPUType(), GetFPUType(), and GetMMUType().

# Amiga Mail

## **ULONG GetCPUType(void)**

Returns a number, representing the type of CPU in the system: 68000L, 68010L, 68020L, or 68030L.

## **ULONG GetFPUType(void)**

Returns a number, representing the type of FPU in the system: 0L (no FPU), 68881L, or 68882L.

## **ULONG GetMMUType(void)**

Returns a number, representing the type of MMU in the system: 0L (no MMU), 68851L, 68030L, or 0xFFFFFFFFL (this means an FPU responding to a MMU address).

In order to find out which processor is present, GetCPUType() first checks ExecBase->AttnFlags. Under V1.3 if this is set to AFF\_68020, then the processor may be either a 68020 or a 68030. GetCPUType() then checks to see if the processor is really a 68030 by trying to invert the instruction burst enable bit, which doesn't exist on the 68020. If that bit can be changed, then it is a 68030 system.

Similar methods are used in GetMMUType() and GetFPUType(). These functions first look at ExecBase->AttnFlags and then do extra tests based on unique features of a given coprocessor to find out what is present in the system. The functions are listed below.

```
=====
; From SetCPU V1.5
; by Dave Haynie
;
; 68030 Assembly Function Module
;
; This module contains functions that access features of the 68020,
; 68030, and 68851 chips, and ID all of these, plus the 68881/68882
; FPU chips, reset stuff, and exception handler.
;=====
; Macros & constants used herein...
;=====

CALLSYS macro *
    jsr     LVO\1(A6)
endm

CIB_ENABLE EQU 0
CIB_FREEZE EQU 1
CIB_ENTRY  EQU 2
CIB_CLEAR  EQU 3
CIB_BURST  EQU 4

CDB_ENABLE EQU 8
CDB_FREEZE EQU 9
CDB_ENTRY  EQU 10
CDB_CLEAR  EQU 11
CDB_BURST  EQU 12
CDB_WALLOCEQU 13

AFB_68030 EQU 2

ATNFLAG EQU $129

LVOSupervisor EQU -30
LVOSuperState EQU -150
LVOFindTask EQU -294
LVOAllocTrap EQU -342
LVOfreeTrap EQU -348
```

```
;=====
;   Need just a little more stuff
;=====

        NOLIST
        include "exec/execbase.i"
        include "exec/tasks.i"
        LIST

*       machine mc68020
*       mc68881

;*****
;   This section contains functions that identify and operate on CPU
;   things.
;*****

XDEF _GetCPUType    ; ID the CPU
XDEF _GetFPUType    ; ID the FPU
XDEF _GetMMUType    ; ID the MMU

;=====
;   This function returns the type of the CPU in the system as a
;   longword: 68000, 68010, 68020, or 68030. The testing must be done
;   in reverse order, in that any higher CPU also has the bits set for
;   a lower CPU. Also, since 1.3 doesn't recognize the 68030, if I
;   find the 68020 bit set, I always check for the presence of a
;   68030.
;
;   This routine should be the first test routine called under 1.2
;   and 1.3.
;
;   ULONG GetCPUType(void);
;=====

_GetCPUType:
        movem.l     a4/a5,-(sp)          ; Save this register
        move.l      4,a6                ; Get ExecBase
        btst.b      #AFB_68030,ATNFLGS(a6) ; Does the OS think an '030 is here?
        beq         0$
        move.l      #68030,d0           ; Sure does...
        movem.l     (sp)+,a4/a5
        rts
0$:
        btst.b      #AFB_68020,ATNFLGS(a6) ; Maybe a 68020?
        bne         2$
        btst.b      #AFB_68010,ATNFLGS(a6) ; Maybe a 68010?
        bne         1$
        move.l      #68000,d0           ; Just a humble '000
        movem.l     (sp)+,a4/a5
        rts
1$:
        move.l      #68010,d0           ; Yup, we're an '010
        movem.l     (sp)+,a4/a5
        rts
2$:
        move.l      #68020,d0           ; Assume we're an '020
        lea         3$,a5              ; Get the start of the supervisor code
        CALLSYS     Supervisor
        movem.l     (sp)+,a4/a5
        rts
3$:
        moveccacr,d1                    ; Get the cache register
        move.l      d1,a4                ; Save it for a minute
        bset.l      #CIB_BURST,d1        ; Set the inst burst bit
        bclr.l      #CIB_ENABLE,d1       ; Clear the inst cache bit
        movecdl,cacr                     ; Try to set the CACR
        moveccacr,d1
        btst.l      #CIB_BURST,d1        ; Do we have a set burst bit?
        beq         4$
        move.l      #68030,d0           ; It's a 68030
        bset.b      #AFB_68030,ATNFLGS(a6)
4$:
        move.l      a4,d1                ; Restore the original CACR
        movecdl,cacr
        rte

;=====
;   This function returns 0L if the system contains no MMU,
;   68851L if the system does contain an 68851, or 68030L if the
;   system contains a 68030 (built-in MMU).
;
;   This code runs just fine on boards from Ronin and
;   Commodore, as well as all 68030 boards it's been tested on.
;
;   ULONG GetMMUType(void)
;=====
```

# Amiga Mail

```
_GetMMUType:
    move.l    d0,d1          ; Get ExecBase
    movem.l   d1,d0          ; Save this stuff
    move.l    d0,d1          ; Call FindTask(0L)
    CALLS     FindTask(0L)
    move.l    d0,d1          ; Change the exception vector
    move.l    d0,d1          ; Try to detect undecode FPU
    ; Let's try an MMU instruction
    ; like PMOVE tc,(sp)
    ; Any MMU here?
    ; Hardware "features"?
    ; Does the OS think an '030 is here?

    ; Return that local
    ; Reset exception stuff
    ; and return the registers

    ; matter what machine we're on,
    ; s in place, we should get a
    ; -Line emulation exception.

    ; Get Amiga supplied exception #
    ; Is it an F-Line?
    ; If so, go to the fail routine
    ; We have MMU
    ; Skip the MMU instruction

    ; t dinna woik,
    ; Skip the MMU instruction

    ; FPU in the system as a

    ; this register
    ; cBase
    ; e OS think an FPU is here?

1$
    rts
    lea       supervisor code
    CALLS
    move.l    d0,d1
    rts

2$
    move.l    fsave-(sp)
    moveq.l   #0,d0
    move.b    1(d0)
    cmpi      #$18,d0
    beq       3$
    move.l    #68881,d0

3$
    frestore (sp)+
    rte
end
```