HowToCode7

COLLABORATORS			
	TITLE : HowToCode7		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 6, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

HowToCode7

1.1 HowToCode: 680x0 Issues

```
        Using 680x0 Processors
        ----------------------

Now the Amiga 600 is the only Amiga that uses the 68000
processor (except CDTV), many of you want to use the new
68020+ instructions to make your code, faster, better
and more fun at parties!

 1.  How can I tell what processor I am running on?
 2.  How to optimise for A1200 / 68020
 3.  Using a 68010 processor
```

1.2 How can I tell what processor I am running on?

```
How can I tell what processor I am running on?
----------------------------------------------

Look inside your case. Find the large rectangular (or Square) chip,
read the label :-)

Or...

        move.l  4.w,a6
        move.w  AttnFlags(a6),d0        ; get processor flags

d0.w is then a bit array which contains the following bits

Bit     Meaning if set

0  68010 processor fitted (or 68020/30/40)
1  68020 processor fitted (or 68030/40)
2  68030 processor fitted (or 68040)   [V37+]
3  68040 processor fitted              [V37+]
4  68881 FPU fitted       (or 68882)
5  68882 FPU fitted                    [V37+]
```

```
6  68040 FPU fitted                    [V37+]


The 68040 FPU bit is set when a working 68040 FPU
is in the system.  If this bit is set and both the
68881 and 68882 bits are not set, then the 68040
math emulation code has not been loaded and only 68040
FPU instructions are available.  This bit is valid *ONLY*
if the 68040 bit is set.

Don't forget to check which ROM version you're running.

DO NOT assume that the system has a >68000 if the word is non-zero!
68881 chips are available on add-on boards without any faster processor.

And don't assume that a 68000 processor means a 7Mhz 68000. It may well
be a 14Mhz processor.

So, you can use this to determine whether specific processor functions
are available (more on 68020 commands in a later issue), but *NOT*
to determine values for timing loops. Who knows, Motorola may
release a 100Mhz 68020 next year  :-)

There is *NO* easy way to check for a Memory Management Unit. The
MMU is present in a broken form in many 680EC30 chips.
```

1.3 68020 Optimization

```
A1200 speed issues:
-------------------

The A1200 has a fairly large number of wait-states when accessing
chip-ram. ROM is zero wait-states. Due to the slow RAM speed, it may be
better to use calculations for some things that you might have used tables
for on the A500.

Add-on RAM will probably be faster than chip-ram, so it is worth
segmenting your game so that parts of it can go into fast-ram if available.

For good performance, it is critical that you code your important loops
to execute entirely from the on-chip 256-byte cache. A straight line loop
258 bytes long will execute slower than a 254 byte one.

A loop of 258 bytes will only cause a cache miss (a word at either
the beginning or the end of the loop). Only a loop of 512 bytes will
cause the entire cache to miss. Of course a loop of 254 bytes will
be faster than one of 258 bytes, but only marginally.

The '020 is a 32 bit chip. Longword accesses will be twice as fast when
they are aligned on a long-word boundary. Aligning the entry points of
routines on 32 bit boundaries can help, also. You should also make sure
that the stack is always long-word aligned.

Write-accesses to chip-ram incur wait-states. However, other processor
instructions can execute while results are being written to memory:
```

```
        move.l  d0,(a0)+        ; store x coordinate
        move.l  d1,(a0)+        ; store y coordinate
        add.l   d2,d0           ; x+=deltax
        add.l   d3,d1           ; y+=deltay

;       will be slower than:

        move.l  d0,(a0)+        ; store x coordinate
        add.l   d2,d0           ; x+=deltax
        move.l  d1,(a0)+        ; store y coordinate
        add.l   d3,d1           ; y+=deltay
```

The 68020 adds a number of enhancements to the 68000 architecture, including new addressing modes and instructions. Some of these are unconditional speedups, while others only sometimes help:

Adressing modes:

- Scaled Indexing. The 68000 addressing mode (disp,An,Dn) can have a scale factor of 2,4,or 8 applied to the data register on the 68020. This is totally free in terms of instruction length and execution time. An example is:

```
68000                   68020
-----                   -----
add.w   d0,d0           move.w  (0,a1,d0.w*2),d1
move.w  (0,a1,d0.w),d1
```

- 16 bit offsets on An+Rn modes. The 68000 only supported 8 bit displacements when using the sum of an address register and another register as a memory address. The 68020 supports 16 bit displacements. This costs one extra cycle when the instruction is not in cache, but is free if the instruction is in cache. 32 bit displacements can also be used, but they cost 4 additional clock cycles.

- Data registers can be used as addresses. (d0) is 3 cycles slower than (a0), and it only takes 2 cycles to move a data register to an address register, but this can help in situations where there is not a free address register.

- Memory indirect addressing. These instructions can help in some circumstances when there are not any free register to load a pointer into. Otherwise, they lose.

New instructions:

- Extended precision divide an multiply instructions. The 68020 can perform 32x32->32, 32x32->64 multiplication and 32/32 and 64/32 division. These are significantly faster than the multi-precision operations which are required on the 68000.

- EXTB. Sign extend byte to longword. Faster than the equivalent EXT.W EXT.L sequence on the 68000.

- Compare immediate and TST work in program-counter relative mode on the 68020.

o Bit field instructions. BFINS inserts a bitfield, and is faster than 2 MOVEs plus and AND and an OR. This instruction can be used nicely in fill routines or text plotting. BFEXTU/BFEXTS can extract and optionally sign-extend a bitfield on an arbitrary boundary. BFFFO can find the highest order bit set in a field. BFSET, BFCHG, and BFCLR can set, complement, or clear up to 32 bits at arbitrary boundaries.

o On the 020, all shift instructions execute in the same amount of time, regardless of how many bits are shifted. Note that ASL and ASR are slower than LSL and LSR. The break-even point on ADD Dn,Dn versus LSL is at two shifts.

o Many tradeoffs on the 020 are different than the 68000.

o The 020 has PACK an UNPACK which can be useful.

1.4 Using a 68010 processor

```
Using a 68010 processor
-----------------------

The 68010 is a direct replacement for the 68000 chip, it can
be fitted to the Amiga 500,500+,1500, CDTV  and 2000 without any
other alterations (I have been told it will not fit an A600).

The main benefit of the 68010 over the 68000 is the loop cache mode.
Common 3 word loops like:

           moveq  #50,d0
.lp        move.b (a0)+,(a1)+  ; one word
           dbra   d0,.lp       ; two words

are recognised as loops and speed up dramatically on 68010.
```