Yacc (Yet Another Color Computer) Functional Description

The Yacc is a 68010 based single board computer. It's main components are the 68010, a megabyte of RAM, up to 1/2 megabyte of ROM, a Zilog SCC for serial communication, an Apple IWM for controlling Sony microfloppies, 3 6522 VIA's, a byte wide port to a Priam Datatower (85 MB), and a byte wide port to an Apple ProFile or a 3COM Ethernet box. The translation buffer portion of a paged MMU is implemented. The following is a simple description of each of the main components.

Memory Management Unit

The MMU consists of 2 - 2Kx8 static RAMs, comparators, multiplexors, bus drivers, and a controlling PAL. There are two translation buffers that are 1024 entries in length, with each entry having 16 bits. Which of the two buffers are selected are determined by the Supervisor/User output from the 68010, and both buffers are accessible to the CPU when in Supervisor state. The pages that are mapped by the MMU are 1024 bytes (1K) in size; this allows the entire megabyte of the machine to be mapped at one time. The 16 bits of each entry are used as follows:

Bit #15 - Referenced Bit. Set to '1' if the page is ever referenced.
Bit #14 - Modified Bit. Set to '1' is the page is ever written to.
Bit #13 - Valid Bit. User controlled, set to '1' is page is mapped.
Bit #12:03 - Physical Address. These are physical addresses 19:10.
Bit #02:00 - Tag Field. These map 8 logical megabytes to 1 physical meg.

The 16 megabyte address space of the 68010 is divided into two spaces, Logical RAM and I/O space. The I/O space is selected by via address line #23 being equal to 1; there is no protection feature that disables a user from addressing the I/O space.

Because the hardware only supports the translation of addresses and the referenced and modified bits, the entire overhead of providing a virtual system must be done in software. The hardware will cause a bus error if the page is not mapped; the software must maintain it's own page tables and other such data structures.

Video Frame Buffer

The Yacc has an eight plane deep frame buffer with a 256 x 16 bit color lookup table. The video access to memory is interleaved with CPU access; this interleave is transparent to the software. The physical memory layout to support the 8 planes needs support by MMU mapping to allow the 8 planes to appear contiguous. The memory can be viewed as divided into 8 128Kb partitions. Each video plane resides in one of these 8 partitions. The physical starting address is the same for each of the planes, and can be viewed as an offset from the start of partition. This starting address loaded from a 16 bit wide register.

Interrupt Structure

There are five lines into a 8 to 3 line priority encoder. They consist of three interrupt request lines from the 3 VIA's, one interrupt request line from the SCC, and a non-maskable interrupt. Each VIA has 2 timers that can interrupt. The VIA #1 can interrupt because of video blanking, mouse movement, every second from a real time clock, and SCC data available(don't ask). The VIA #2 can interrupt because of the Priam Port, and the Parallel Port. The VIA #3 is unconfigured

and has no dedicated usage. The SCC can interrupt because of available data and other such state transistions. The NMI interrupt is caused by pressing a button. The following is the assignment of the interrupt lines:

Interrupt Level	Chip	Possible source
00		
01	VIA #3	currently undedicated
02	VIA #2	Priam and Parallel Ports
03	VIA #1	Mouse, Video, Real Time Clock, SCC
04		
05	SCC	SCC state transistions
06		
07	NMI	Button Push

Interrupts

The CPU is capable of being interrupted at five different levels (1-3,5,7); they are allocated as follows:

Level	Function
1	Slow-speed general purpose parallel port (SSPvia)
2	Miscellaneous control / status (MSCvia)
3	MAC-compatible control / status (MACvia)
5	Serial communication controller (SCC)
7	'NMI' console button (NOT DEBOUNCED)

Memory Management Unit

Believe me, you really do not want to know how this works.

Serial I/O Subsystem

The YACCintosh uses the Zilog/AMD SCC Serial Communications Controller as the heart of its asynchronous/synchronous communications subsystem.

I now suggest you read (skim) the AMD SCC/Z8530 Technical Manual for a full description of the part; the following paragraphs will be significantly easier to understand.

The SCC interfaces via a control/data register protocol. The control register is first loaded with the desired internal register ID (a number from 0 to 15), then the SAME control register is read or written to access the actual register contents. Thus an internal register access is (MUST BE) an INDIVISIBLE pair of control register accesses. A read of the control register will always get the SCC 'in sync' so that a write-address/read-write-data sequence can be successfully performed (this synchronization need only be done if it is not already known that the SCC has been left in a synchronized state). After a valid internal register data access, the internal SCC indirect pointer value is set to 0 (the status register).

A data register access refers to the data buffer registers within the SCC, it is NOT used when refering to internal SCC control registers (ie, the external data register is actually just a fast way to access internal register 8).

Separate control and data register sets exist for each of the two channels of the SCC (A,B).

Note also that the SCC registers DO NOT access at the same external address for reading and writing. Only byte moves to and from the SCC are valid; all other operations are UNDEFINED.

The two DCD ports of the SCC are used not for serial I/O related functions, but rather as input sources for movements of the mouse. See the below section on the mouse interface for further details.

The SCC interrupts the processor at level 0x5.

(This may all seem pretty wierd at first; but this is the way the MACintosh did it, and the YACCintosh is MACintosh compatible).

Notes
---

1) The SCC hardware requires a guaranteed minimum of 1800 nanoseconds between any two references to its control register (address and/or data). This corresponds to four NOPs (or equivalent number of memory accesses to instructions or data) on the YACCintosh.

2) A sample setup for ONE channel of the SCC in async mode is (note that the ORDER of writing thew registers IS important):

```
wr9  <- 0x00   interrupts off
wr1  <- 0x00    ditto
wr15 <- 0x00    ditto

wr4  <- 0x4C   BRG in x16 mode, 2 stop bits, no parity
wr11 <- 0xD0   RcvClk=XmtClk=BRG, BRG from XTAL
wr10 <- 0x00   NRZ mode (necessary ?)
wr12 <- 0xFF&(BRGcnt>>0)   BRG low byte
wr13 <- 0xFF&(BRGcnt>>8)   BRG high byte
wr14 <- 0x01   BRG from XTAL & enable
wr3  <- 0xC1   Rcv=8 bits & enable
wr5  <- 0x6A   Xmt=8 bits & enable, RTS enable

wr8  <- '?'    Transmit some data ...
```

3) The formula for computing the BRG counter value is:

```
BRGcnt = 3686400/(2*16*baud) - 2;
```

Then for baud=9600, BRGcnt=10 (all numbers in decimal). This assumes the BRG source = XTAL at 3.6864 MHz, and the divider is in divide-by-16 mode (REQUIRED FOR ASYNC MODE).

EtherNet/3Com Subsystem

Priam DataTower Subsystem

Floppy Disk Interface

Believe me, you REALLY do NOT want to know how this works.

Mouse Interface

Believe me, you really do not want to know how this works.

Keyboard Interface

Believe me, you really do not want to know how this works.

Time-of-day Clock Interface

Believe me, you really do not want to know how this works.

VideoMap Lookup Table

Believe me, you really do not want to know how this works.

YACCintosh MMU Description DNN, 15-NOV-1984

OVERVIEW

The YACCintosh (YACC) memory management unit (MMU) implements a simple subset of a demand-paged virtual memory architecture. Specifically, hardware supports a translation buffer cache (TBUF) mechanism and access control (valid/referenced/modified) bits on a per-page basis, for one active user-level and a system-level process context.

The 24bit virtual address (VA) of the 68010 processor provides access to a 16 Megabyte (MB) process address space. This is subdivided into two 8 MB spaces; I/O space and R/W memory space.

I/O space is unmapped, and is thus the same for all processes (system and user). There is no access protection mechanism - any process may access any I/O address. This is NOT a bug; as the YACC is a development machine, unrestricted access to I/O was explicitly desired so that interfaces to new I/O devices could be easily developed without having to write a device driver and relink the kernal (on UNIX, for example). Certain I/O devices will just be 'off limits' to application access; violate this protocol and the response of the operating system will be 'undefined'.

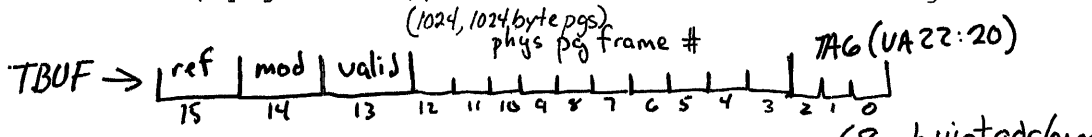
Memory space is mapped via the MMU to provide isolation between processes, and allow process sizes (program plus data) significantly larger than the limits imposed by the actual 1MB physical memory size:

1MB total - 0.3MB bitmap - 0.1?MB os = 0.6MB available

Process memory space is divided into 8192 - 1024 byte pages (8MB total). The TBUF is a one-way set-associative direct-mapped cache of 1024 (pure coincidence that this is also the page size) of the possible 8192 entries in a process page table.

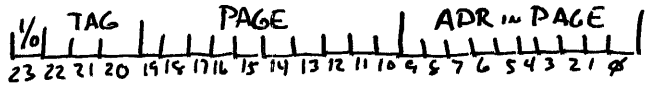
There are two separate, non-interacting TBUF contexts; one is used while the processor is accessing memory in SYSTEM mode, the other while in USER mode.

During memory accesses (not I/O), access validation (VALID bit and TAG field compare), maintenance of REFERENCED/MODIFIED status bits, and physical memory address generation is done automatically for all pages mapped validly by the contents of the TBUF. Any access fault interrupts the processor, which must fix the problem (TBUF maintenance, page faults), and then continue the faulting process.



DETAILED OPERATION

Here is a pseudo-code description of the MMU operation; first some definitions:



VA<23:00> - processor virtual address
- 23 0=memory / 1=I/O space
- 22:20 tag compare field, if memory
- 19:10 TBUF index field, if memory
- 09:00 displacement in page, if memory

MA<19:00> - memory address

DB<15:00> - processor data bus

TB[0..2047]<15:00> - the TBUF itself
- REF = 15 referenced bit, set by hdwe on any access
- MOD = 14 modified bit, set by hdwe on WRITE access
- VAL = 13 valid bit, set by sfwe if TBUF entry is valid

```
        PFN = 12:03     physical page frame number; set by sfwe
        TAG = 02:00     TBUF tag entry; VA<22:20> of the PTE to
                        which this entry corresponds; set by sfwe

    WRITE = 1 for cpu write cycle, 0 for read cycle

    CNTXT = 1 for cpu in SYSTEM mode, 0 in USER mode
```

On a per-memory cycle basis, the operation of the MMU proceeds as follows:

```
    /* '||' is bitfield concatenation */

    if (VA<23> eq 0) {
        /* VA<23>=0: r/w memory access */
        I = CNTXT || VA<19:10>;                     /* just for clarity */
        if (TB[I]<VAL> eq 1 and VA<22:20> eq TB[I]<TAG>) {
            /* TBUF entry is correct ... proceed */
            TB[I]<REF> = 1;                         /* set REF bit */
            TB[I]<MOD> = TB[I]<MOD> or WRITE;       /* set MOD bit on write */
            MA<19:00> = TB[I]<PFN> || VA<09:00>;    /* compute memory address */
            ALLOW_MEMORY_WRITE();                   /* but only if WRITE set */
            return(DTACK);                          /* and return OK */
        } else {
            /* TBUF entry is NOT valid for this virtual address */
            INHIBIT_MEMORY_WRITE();                 /* since MA is junk ... */
            return(BUS_ERROR);                      /* return ERROR & abort */
        }
    } else {
        /* VA<23>=1: i/o access */
        if (VA<23:20> eq 0x900000>>20) {
            /* i/o access to mmu register set */
            I = VA<12> || VA<10:01>;                /* just for clarity */
            if (WRITE eq 1) {
                /* register write - SYSTEM only */
                if (CNTXT eq SYSTEM) {
                    TB[I] = DB<15:00>;
                }
            } else {
                /* register read */
                DB<15:00> = TB[I];
            }
            return(DTACK);                          /* always return OK */
        } else {
            /* some other i/o ... */
        }
    }
```

PROGRAMMING INFORMATION

Power-Up Initialization

When the system is first powered on or RESET, the contents of the MMU will be somewhat random. The POWER_UP bit in the MAC VIA will be ON, relocating the ROM to start at address 0x0, thus fetching the restart PC/SP from ROM. The ROM initialization code will then set the system-level MMU context to be a (linear) map of all available memory (1MB), with VALID bits set and the MMU tag field set to 000b. The ROM code then begins executing at its image starting at 0x80xxxx, and clears the POWER_UP bit. This effectively turns the MMU ON, and the 1MB RAM memory now appears at 0x000000-0x0FFFFF, each page mapped uniquely and marked as valid.

Setting a context

The TBUF accesses as two 1024-entry banks of word-wide registers in I/O space. The single user context is accessible at 0x900000-0x9007FE; the single system context is at 0x901000-0x9017FE. See the above tables (or the file 'yacc-hdwe.h') for the exact format of an entry. Note that only word-wide write accesses are allowed to the TBUF (no byte writes). The TBUF is readable from any process; only a process in system mode can write to it, however. This provides a reasonable level of protection between mutually interfering processes.

Translation buffer faults, page faults

Any error in translating the current VA thru the TBUF (due to either a 'real' page fault; or a TBUF fault when the indexed TBUF entry holds a tranlation entry for another page) will result in a BUS_ERROR to be generated on the 68010 CPU. It will then stack however many words it feels is necessary to save its state, and enter the BUS_ERROR interrupt routine, which then services the TBUF fault or page fault (determined by looking at the faulting VA on the stack and the contents of the TBUF location it would index). Note that this implementation is NOT a high performance virtual memory system; but it will get the job done. There is enough hardware support to allow emulation of a reasonable subset of the of the Motorola MMB/MMC/PMMU subsystem.

For a TBUF fault, the current contents of the desired TBUF location must be saved in the memory image of the process's page table, and then the TBUF location loaded with the appropriate data from the new page table entry [this is all done by software].

Referenced, modified bits

These bits are set automatically by the hardware whenever an indexed TBUF entry is valid and subsequently accessed or written, respectively. All other bits in a TBUF entry are not modified by the hardware in any way.

```
YACCintosh AddressMap                                   DNN, 06-Feb-85

           22221111111111000000000
BaseAddr   32109876543210987654321
           -----------=========----
$00,0000   00xx..00**************** A r/- d ROM (64KB), power-up
$40,0000   01xx****************** A r/w c RWM (1024KB), power-up
$00,0000   0********************** A r/w c RWM (8192KB), normal
           -----------=========----
$8C,0000   1000..00**************** A r/- d ROM (64KB, bank0)
$8D,0000   1000..01**************** A r/- d ROM (64KB, bank1)
$8E,0000   1000..10**************** A r/- d ROM (64KB, bank2)
$8F,0000   1000..11**************** A r/- d ROM (64KB, bank3)
           -----------=========----
$9F,E800   1001.......*.**********0 W r/w d MemoryManagement
           -----------=========----
$AC,FFFF   1010..00.............1 L r/w d FastPPort (fast access)
$AD,FE00   1010..01.......********0 W r/w d VideoMap
$AE,....   1010..10.............. - -/- d unused
$AF,....   1010..11.............. - -/- d unused
           -----------=========----
$BC,FF81   1011..00.........******1 L r/w d PriamPort
$BD,....   1011..01.............. - -/- d unused
$BE,....   1011..10.............. - -/- d unused
$BF,....   1011..11.............. - -/- d unused
           -----------=========----
$C.,....   1100.................. - -/- d unused
           -----------=========----
$DC,FFFE   1101..00.............* A -/w p MicroFloppySpeed (PWM)
$DD,FFE1   1101..01...........****1 L r/w p MicroFloppyData/Cntl (IWM)
$DE,FFF8   1101..10............**0 U r/- p SerialIO (SCC)
$DE,FFF9   1101..10............**1 L -/w p SerialIO (SCC)
$DF,....   1101..11.............. - -/- p unused
           -----------=========----
$EC,FFFF   1110..00.............1 L r/w p FastPPort (slow access)
$ED,FF80   1110..01........100****0 U r/w p VIA, GP SlowPort
$ED,FF40   1110..01........010****0 U r/w p VIA, MACbits
$ED,FF20   1110..01........001****0 U r/w p VIA, MISCbits
$EE,FFFE   1110..10.............0 W r/w p SoundBuffer DMAddress
$EF,FFFE   1110..11.............0 W r/w p VideoBuffer DMAddress
           -----------=========----
$FC,....   1111..00.............. - -/- p unused
$FD,....   1111..01.............. - -/- p unused
$FE,....   1111..10.............. - -/- p unused
$FF,FFFE   11111111.............* A r/w p InterruptAcknowledge
           -----------=========----\|  |  |
           22221111111111000000000 |  |  d == fast dtack cycle
           32109876543210987654321 |  |  c == conditional dtack cycle
                                   |  |  p == peripheral (vpa) cycle
                                   |  |
                                   |  r == read access allowed
                                   |  w == write access allowed
                                  /|
                                . - == unused space, all accesses NOPed
                                * A == word/byte access, any combination
                                0 W == word access only
                                * B == upper or lower byte access only
                                0 U == upper byte access only
                                1 L == lower byte access only
```