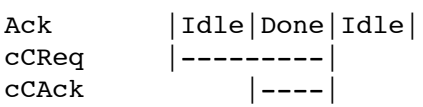
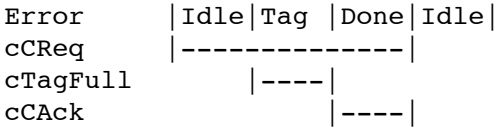
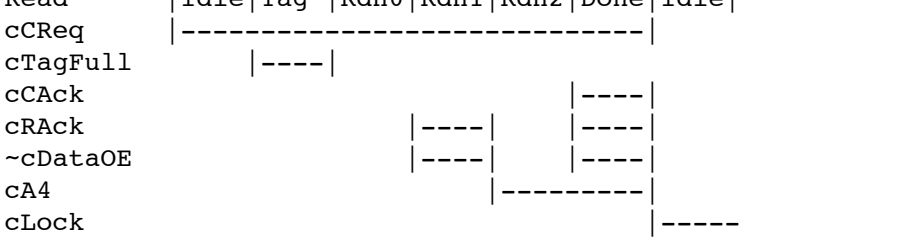
ALPHA PC flows
Narrow (64 bit) main memory
dgc
20-Aug-91

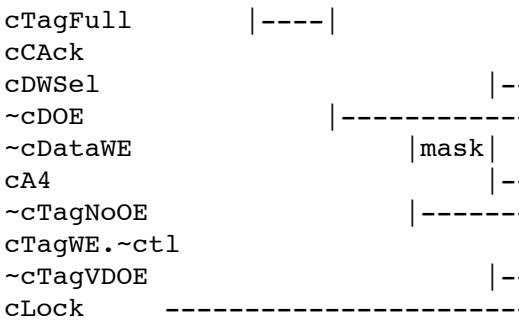
Cycles that just get a fast ACK, like
FETCH, BARRIER, and STxC that are already known to
need to fail.



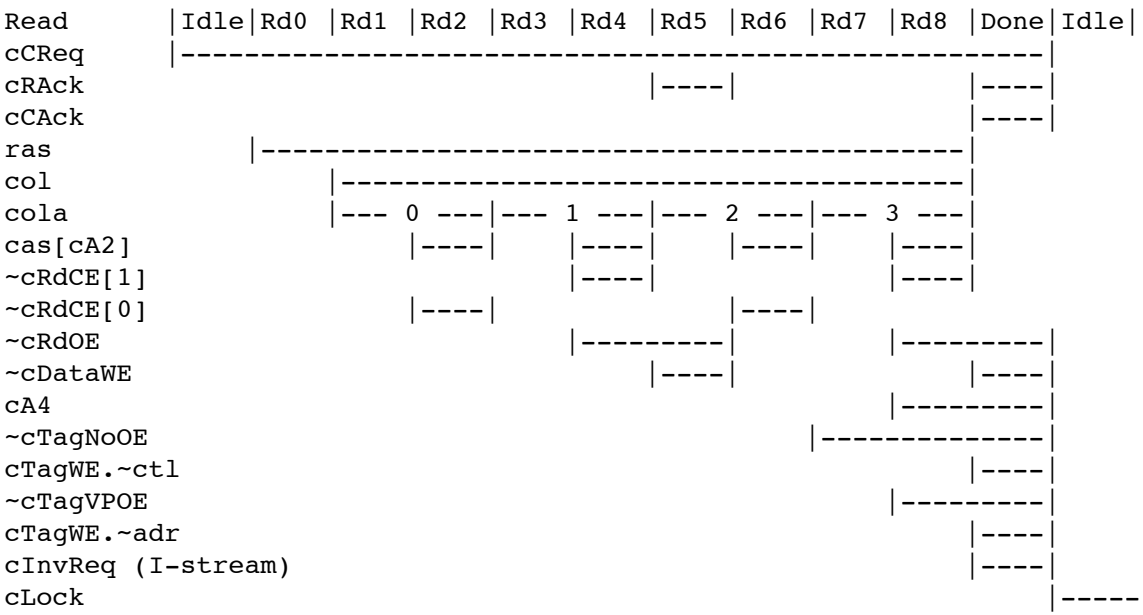
Cache hit cycles. These can only happen on
LDxL and STxC cycles. We add an extra cycle (the Tag
cycle) to let the tag check settle out. The cLock
signal sets on LDxL, and clears on STxC.
The cTagFull signal asserts in the Tag state to
tell the tag logic (page 6) that it must do the full
tag check, as opposed to the quick one.
On writes, the ~cTagNoOE signal is used to
disable the output of the tag rams.
The ~cDOE and the cDWSel signals do not
need to be extended for an extra cycle since the write
pulses on the cache rams are generated directly by
the cpu chip (there is no clock buffer delay between the
advance of the write data and the write pulse).





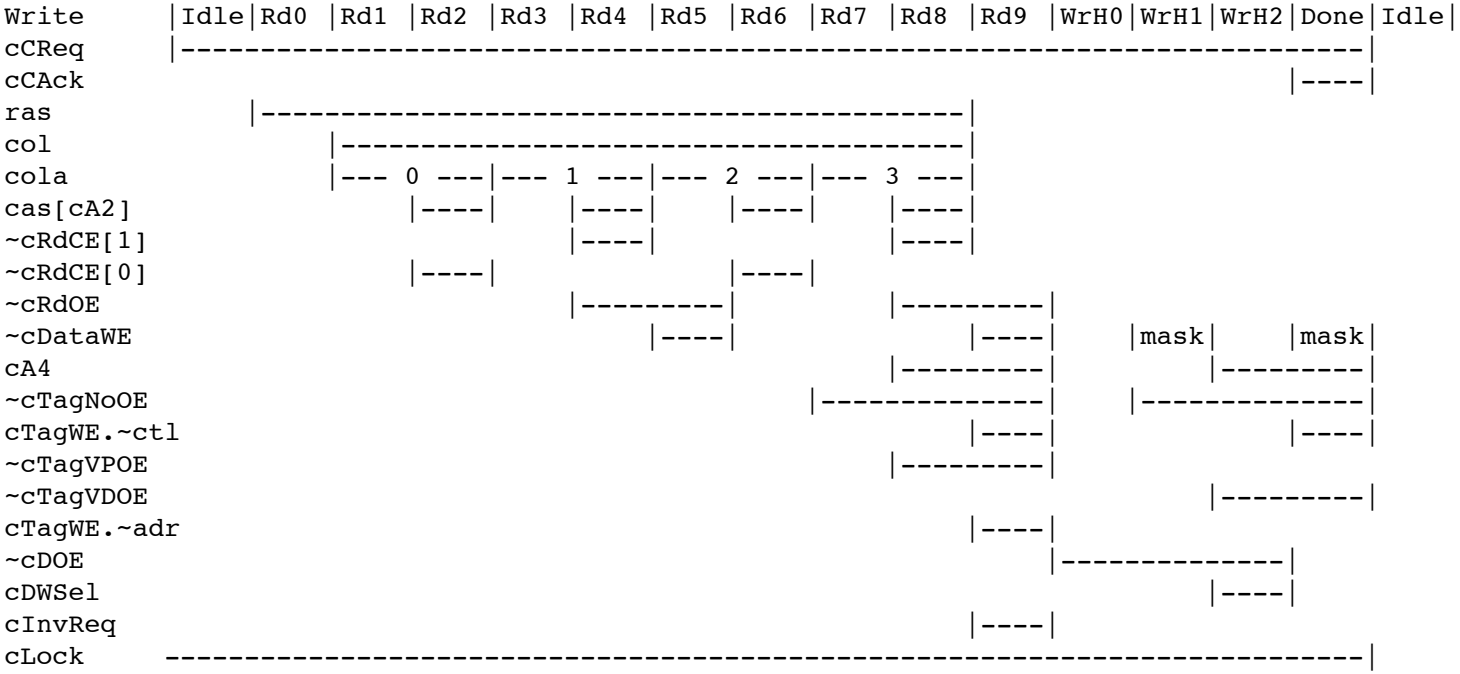


Read miss. This is the READ_BLOCK case. The LDxL miss case looks
the same, but there is a Tag state between the Idle state and the Rd0
state. The cLock signal sets on LDxL.
We always generate the cIncReq if the read is for I-stream. If the
block in the backup cache is invalid then the block cannot possibly be
in the data cache, and in the invalidate is a (harmless) no-op.
If the block in the backup cache is valid then the block *might* be
in the data cache, and if we replace it with an I-stream block then an
invalidate of the data cache is needed.

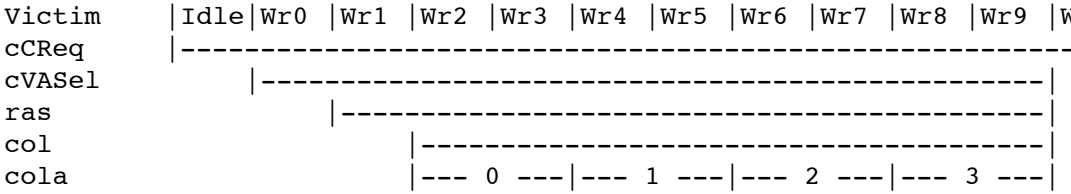


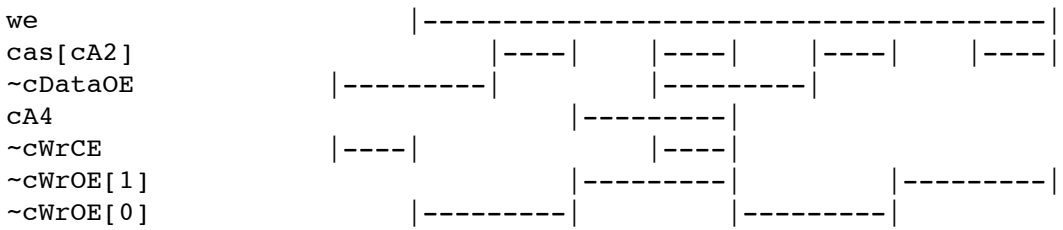
Write miss. This is the WRITE_BLOCK case. The STxC miss case looks
the same, but there is a Tag state between the Idle state and the Rd0 state.
The cLock signal clears on STxC. The tag write in Rd9 isn't really needed,

but it makes the front of a read miss the same as the front of a write miss,
which seems like a good idea. Since the block in the B-cache is being
replaced with a new block, and invalidate of the D-cache is needed to ensure
that the D-cache is a subset of the B-cache.

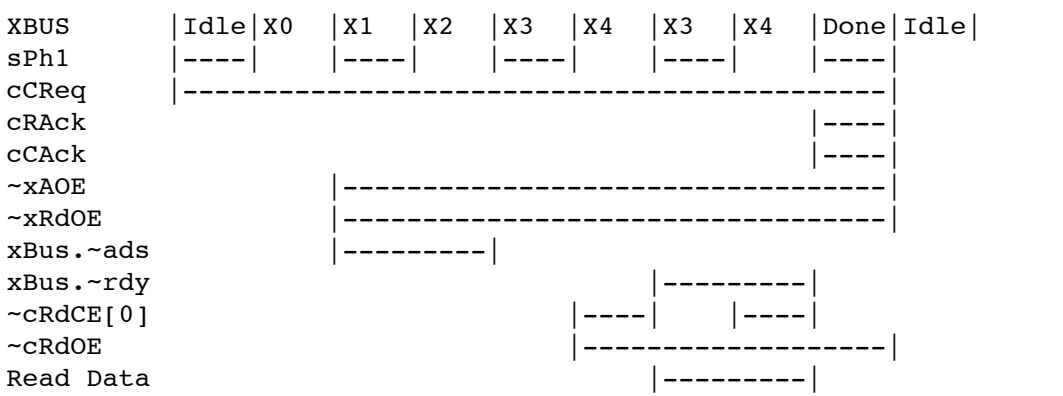


Victim write. We are deliberately allowing the memory bus tranceivers to
tri-state overlap. There are several ways to correctly design this fragment of the
logic, all of which avoid the overlap, and one of which should be done in a
production machine. One way would be to build a 3 port bidirectional data switch in
a big pld. Another way uses a pre-existing data path switch, like the 49FCT804
from IDT. I chose to be a little sleazy, rather than spending the time to do it right.
The alternative would be to add dead cycles, which would eliminate the overlaps,
but would make the machine run more slowly, and confuse any performace studies. For
early FCC scans we can try running contrived programs with the same data in
all quadwords, and see what difference the overlap makes in the spectrum.

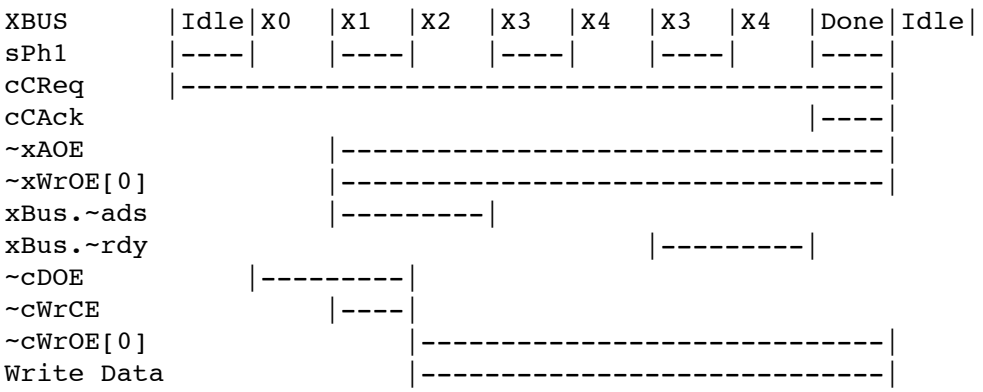




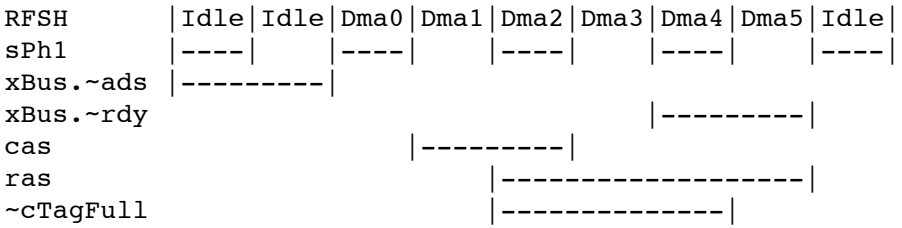
XBUS read. We only generate a single RACK, since there
is garbage in all but longword 0. The RACK always says that the
reference should not be cached, and also says that it should
not be error checked (longword 0 is ok, but the garbage does in
the other longwords does not have good parity).
The cycle must run in the correct phase on the XBUS. The CPU
state machine looks at sPh1 when it is in the Idle state, and only
makes the Idle->X0 transition when it is phase1, so that X0
will be phase2, making X1-X2 a phase1-phase2 pair.
Note that if the source of the data is the 82380, which is clocking
on mosClk rather than sysClk, it is possible that the data has
only 3nS of hold at the fct2952's (6nS 82380 min - 5nS worst case
mosClk/sysClk skew + 2nS f623 min), but this is enough for the
fct2952's (who only need 2nS).



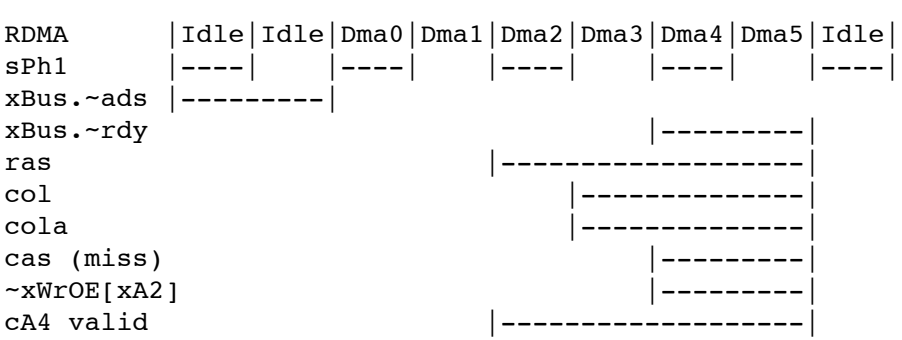
XBUS write. On a 386 the write data comes valid in the middle
of the ADS cycle, and holds into the middle of the cycle after the
RDY. The ~cDOE signal is held for 2 cycles, rather than the
more intuative 1 cycle, because the cpu is, in fact, clocking a
little ahead of the rest of the machine, and if you asserted the
~cDOE for a single cycle it would switch the data away just
as the registers were clocking it in.

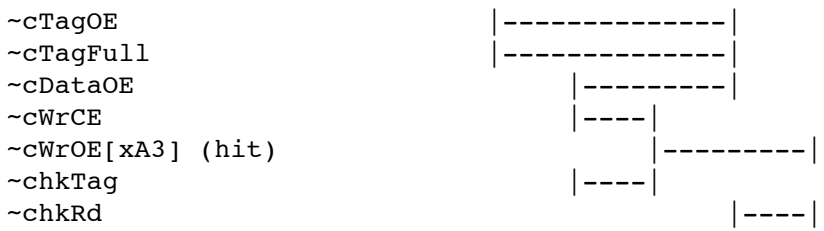


Refresh cycles use cas-before-ras mode, so that we can
program the address generator in the Intel chip in the strange
way needed to generate refresh addresses on the ISA bus.
The cTagFull assertion just simplifies the logic.

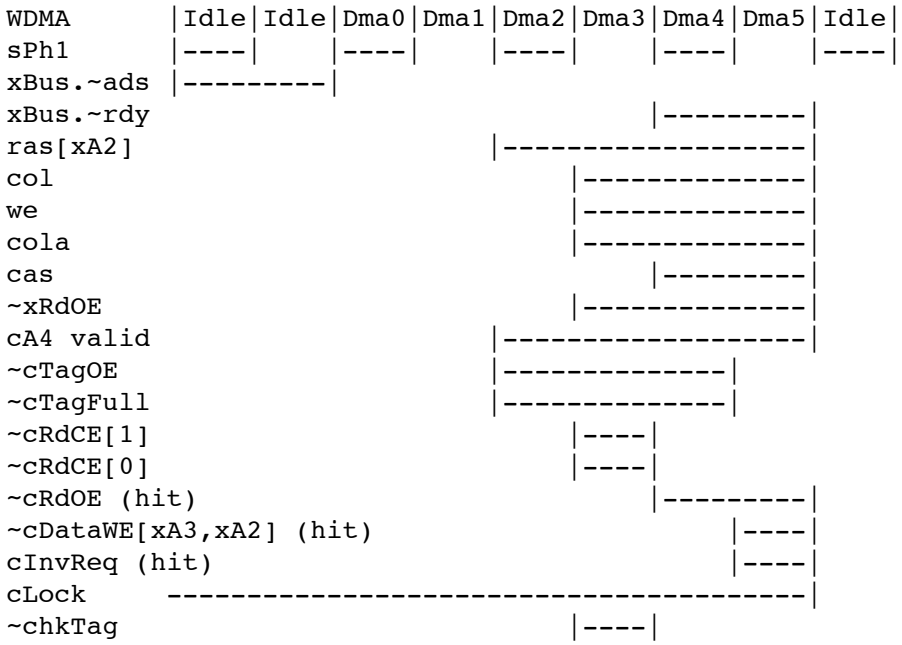


Read DMA. We start the DRAM, and at Dma2 we send the
right xA[4] to the cache, and enable the output of the tag
and data rams. At the end of Dma3 we grab the outputs of the
cache (just in case) and we make the hit-miss decision.
On misses we CAS the DRAM. On hits we output enable the data
from the cache. In all cases we enable the XBUS buffers,
and feed the right longword down the XBUS.

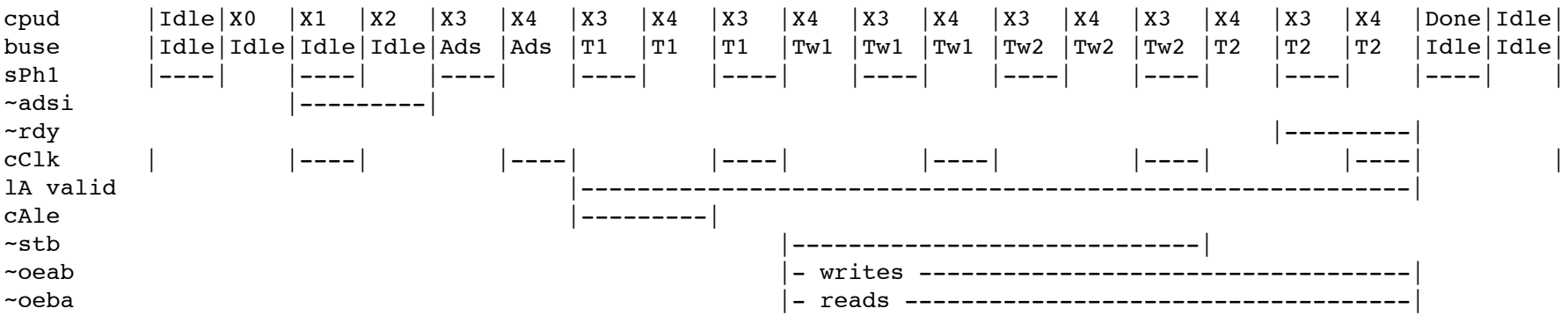




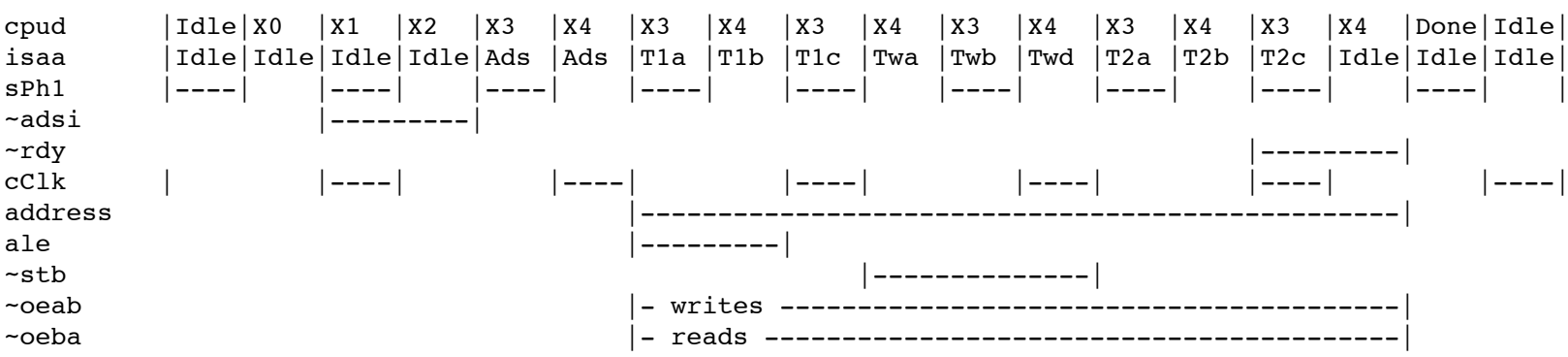
Write DMA. We always run the DRAM. At Dma2 we send
the right xA[4] to the cache, and enable the output of the
tag so we can make a hit-miss decision 2 cycles later.
The ~xRdOE comes on early, since we have to meet write data
setup to cas at the DRAM. On hits we also send write
data to the cD bus, hit the write pulse on the cache
SRAM, and send an invalidate to the CPU. The lock flag
clears at this time. Note that the clearing of the lock
flag is *not* a function of hit.



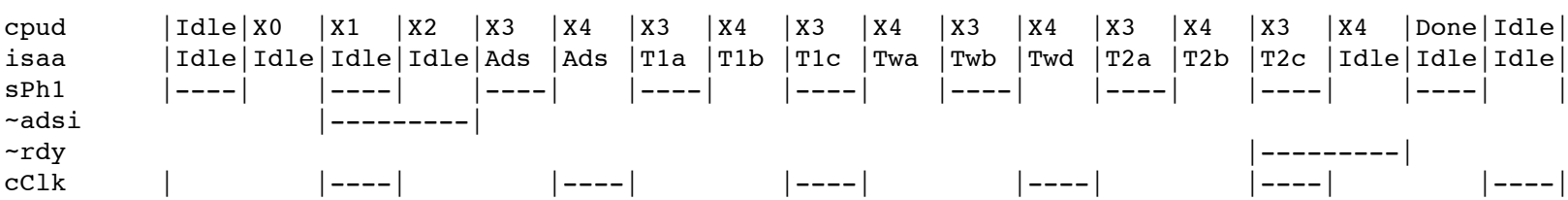
Local cycles. The local bus looks like an 8MHz XT bus. The
length of the cycle is constant, since there isn't anybody on the
bus that wants to stretch it (cRdy on the combo is only an
output from the programable ready generator). This is just one
of the many possible allignments.

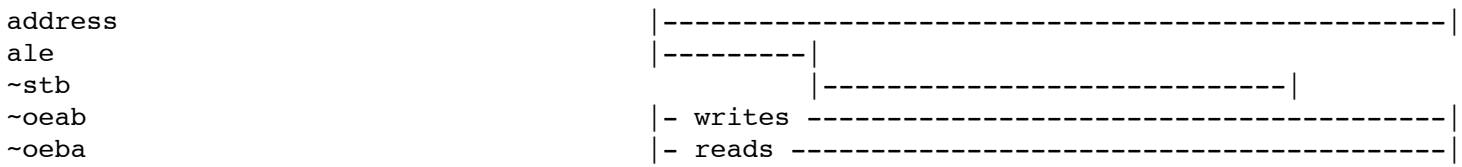


XT style cycles. This is the shortest possible cycle, which cannot, in
fact, every happen on a *real* IBM PC. The *real* IBM PC always asserts the
strobes for 2 T-states. The programmable delay bits can be set to get
this cycle length. The ready signal is sampled at the end of each twb state. If
ready is false the state after twb is twc, then twa.



AT style cycles. This is the shortest possible cycle non 0-wait, which cannot, in
fact, every happen on a *real* IBM PC. The *real* IBM PC always asserts the
strobes for 2 T-states. The programmable delay bits can be set to get
this cycle length. The ready signal is sampled at the end of each TWB state.
ready is false the state after TWB is TWC, then TWA. The 0-wait state is sampled
false at the end of T1C.





AT style 0-wait state cycle. The 0-wait state sample happens at the end of T1C.