CTSS Hardware

Bob Supnik (fifth revision, 22-Sep-2005)

All of the "specifications" given below are derived from the CTSS sources on Paul Pierce's site and from surviving CTSS documentation. They are likely to be incomplete; only the functionality used by CTSS can be deduced.

1. Interval timer

Location 5 is incremented at 60Hz. When it overflows, that is, bits 1-35 increment to 0, a trap occurs. The PC is saved in location 6, and the next instruction is taken from location 7. Traps are inhibited, as with data channel traps.

It also appears that ENB <17> controls the interval timer trap, but this is not certain; data channel traps are always gang-enabled in CTSS.

This is similar, but simpler, than the interval timer option for the 7044. In particular, there is no interval timer reset trap (which indicates that a second interval timer overflow occurred without the first one being serviced). The differences are documented in the 7044 Principles Of Operation.

2. Extended memory

Memory is doubled with the addition of "B core", a second bank of 32KW. There are separate controls for use of B core for instruction (and indirect) references, and for data references. Four new instructions control B core:

```
SEA (-076100t00041)      set data references to A core
SEB (-076100t00042)      set data references to B core
TIA (+0101f0tyyyyy)      transfer to A core at eff addr
                         (do not set user mode)
TIB (-0101f0tyyyyy)      transfer to B core at eff addr
                         and set user mode
```

When a trap occurs that writes the decrement (protection trap, clock trap, channel trap, or floating point trap), the use of A and B core is recorded in the decrement of the saved trap word, as follows:

```
bit<3>          0 = instructions executing in A core
                1 = instructions executing in B core
bit<4>          0 = data references in A core
                1 = data references in B core
```

All traps store the trap word in A core, set the data reference mode to A core, start executing in A core, and reset user mode.

[The behavior of traps that only modify the address of the trap word cannot be deduced from the sources. For examples, on an STR trap, CTSS simulates the behavior by copying A-core location 0 to B-core location 0 and resuming execution at B-core location 2. For a floating point trap, CTSS clears bit <3:4> before copying the address and decrement (only) of A-core location 0 to B-core location 0.]

Channels specify A core vs B core by bit<20> of the channel command word. Bit<20> = 0 is A core, = 1 is B core. This effectively extends the channel initial starting address to 16b. The channel address register remains 15b and does not change the A/B select bit on overflow.

[The selection of A vs B core is illustrated in routine CMEXIT, which sets up memory for a return to an interrupted user process or a system task. The selection of A vs B core for a channel is illustrated in the CTSS bootstrap.]

3. Protection

Protection is implemented through two mechanisms: implementation of user mode, and implementation of memory relocation and protection.

3.1 User Mode

User mode is a subset of standard 7094 mode. In user mode,

- Memory accesses are subject to memory relocation and protection
- Certain instructions are forbidden and cause a protection trap

The TIB instruction sets user mode. User mode is cleared by any trap, including a protection trap. Note that TIA is privileged but does not set user mode. Thus, TIA allows utilities to test whether timesharing is running. If TIA traps, then timesharing is in operation; if it does not, timesharing is off, and user mode is not enabled.

The list of privileged instructions:

- all I/O instructions (RDS, WRS, BSR, BSF, SDN, RUN, REW, etc)
- all channel instructions (RCHx, LCHx, SCHx, etc)
- all I/O transfer instructions (TEFx, TRCx, TCOx, TCNx)
- plus and minus sense (+0760... and -0760...)
- HPR and HTR
- ENB
- SEA, SEB, TIA, TIB, LRI, and LPI

Nested XEC's were <not> trapped, and XEC * would hang the system.

Protection traps save the PC and A/B flags in the address and decrement, respectively, of location 032, clear user mode, set instruction and data memory to A core, and execute the instruction in location 033.

The list of privileged instructions is derived from the CTSS background task emulation routine, which executes a subset of the privileged instructions on behalf of a background process. The lack of trapping on nested XEC's comes from Jerry Saltzer and Stan Dunten.

3.2 Memory Relocation and Protection

According to the CTSS book, memory is managed in 256W blocks. Memory relocation and protection is done by three 7b registers:

- base: defines the 256W block that is the base memory addr
- end: defines the 256W block that is the end memory addr
- relo: defines the value to be added to the block number prior to memory access

The relocation and protection registers are loaded as follows:

```
LRI (+0562f0tyyyyy)      load relocation from M[ea]<addr>
LPI (-0564f0tyyyyy)      load base from M[ea]<addr>
                         load end from M[ea]<decrement>
```

The CTSS code operates as though the relocation and protection registers were 15b rather than 7b, and as though the base and relocation are always zero. The algorithm is:

- load image base into both base and relo
- add number of words
- load sum into end

Various code sequences that check the legality of virtual addresses skip relocation and check only the end address. This is consistent with the so-called "onion-skin" swap algorithm used in CTSS. However, one sequence adds in the relocation register and then checks it against the end register. One could conjecture that relocation is performed before protection checks, but there's no way to confirm that from the code.

The actual protection check appears to be:

base<0:6> <= addr<0:6> <= end<0:6>

This is confirmed by a statement in the CTSS programmers guide, that a user may be able to access more than the allocated number of words, because protection checks are done on 256W boundaries; but only the allocated number of words is swapped in and out. If the user was allocated 1024 words, then base = 0 and end = 4; any address up to 1024 + 255 would be considered valid.

4. I/O

The CTSS system corresponding to the sources that we have is the “red machine” from Project MAC. It has the following I/O configuration:

```
channel A:     7607, card reader and punch, line printer,
               8 tape drives, Chronolog clock
channel B:     7607, 7 tape drives
channel C:     7909, 2 2302 disks, 1 7320 drum
channel D:     'direct channel' connection to display
channel E:     7909, 7750 communications controller
channel F:     unused
channel G:     7289 (7389?), 2 7320A high-speed drums
```

4.1 Channel A (7607)

The card reader, card punch, and line printer are described in the 7094 Principles Of Operation. CTSS allows the background processor to use all three, although the line printer can only be used in BCD mode.

The tape drives are described in the 7094 Principles Of Operation. CTSS allows the background processor to use drives 1-7 and 0. [Note that this allows the background processor to read the Chronolog clock.]

Tape drive "7" is the Chronolog clock. It returns 2 words of BCD digits giving the current time of year (without the year):

```
word1:         month|day in month|hour in day
word2:         minute in hour|second in minute|60th second
```

The months and day in month are 1-based; all others are 0-based. The Chronolog apparently accounts for leap years correctly.

4.2 Channel B (7607)

The tape drives are described in the 7094 Principles Of Operation. CTSS allows the background processor to use drives 1-6 and 0.

4.3 Channel C (7909)

4.3.1 7631 File Control

Channel C is a 7909 supporting a single 7631 file controller. The 7631 in turn supports ten logical units (called modules), of which CTSS uses 5. The best online document for the 7631and its devices is "IBM 1301, Models 1 and 2, Disk Storage, and IBM 1302, Models 1 and 2, Disk Storage, with IBM 7040 and 7044 Data Processing Systems", A22-6768. The material is directly applicable to the 709x series as well.

4.3.2 2302 Disk

The 2302 disk is apparently a later model of the 1302 disk, which in turn was the successor the original 1301 disk. The basic building block of the series is the module, a stack of 25 fixed disk platters, 20 of which hold data on both sides (40 data surfaces). Each surface was divided into tracks (250 on the 1301, 500 on the 1302 and 2302). Each track is a serial bit stream. The 2302-2 that CTSS used has two modules per physical enclosure, and each system has two enclosures, for a total of four modules – a total of less than 80MW (340MB) of storage.

Tracks are numbered sequentially, from 0-9999, based on the cylinder number and the track within cylinder. On the 1302/2302, the second set of 250 cylinders is addressed via a second access arm.

The 1301/2302 series allows variable formatting per cylinder. Each track has a fixed home address (the track number), a user-specified variable home address, and then user-specified record numbers of variable length. The formula for how much data fit in a track is a complex function of the length of home address 2, the length of the record numbers, and the records themselves. Fortunately, CTSS uses the 2302 disk is a simplified way:

```
home address 2 =         67676767676 (=HXXXXXX)
record address =         TTTTRM, where
    TTTT =                track number
    R    =                record within track (0 or 1)
    M      =              physical module number
records per track =      6: 2 data, 4 filler
                          435 word data record
                          31 word filler record
                          14 word filler record
                          435 word data record
                          16 word filler record
                          1 word filler record
```

The mapping from logical modules to physical disks is as follows:

```
            logical              physical

              0              access 0, module 0
              1              access 1, module 0
              2              access 0, module 1
              3              access 1, module 1
              4              access 0, module 4
              5              access 1, module 4
              6              access 0, module 5
              7              access 0, module 5
[drum         8              access 0, module 2]
```

Presumably, the 1301 had just a single record per track, in the same format as the 7320 drum, but this cannot be proven from the sources.

4.3.3 7320 Drum

The 7320 drum is much smaller. It has 400 tracks. As with the 1301 series disks, the tracks support variable format. CTSS uses the drum in a simplified way:

```
home address 2 =         67676767676 (=HXXXXXX)
record address =         TTTTRM, where
    TTTT =                track number
    R    =                record within track (always 0)
    M      =              physical module number (always 2)
records per track =      3: 1 data, 2 filler
                          435 word data record
                          16 word filler record
                          1 word filler record
```

Thus, the drum holds 174K 36b words - less than 6 core loads. The drum is on access 0, module 2, of the 7631 file controller and responds as logical module 8.

4.4 Channel D (7607)

Channel D is the experimental channel. For the sources that we have, it is connected to an experimental display driven by a PDP-7 and uses the 7094's "direct interrupt" capability. The experimental display is not documented, except by the source code, and is unlikely to be reproducible in simulation.

4.5 Channel E (7909)

Channel E is a 7909 supporting a 7750 communications computer. The 7750 is not well documented, other than an overview in the IBM Journal of Research, March 1963. For simulation purposes, its detailed behavior is irrelevant; instead, it has to be reverse-engineered from the CTSS source.

According to the Journal article, the 7750 supports up to 112 lines. The CTSS sources allow for 62. Lines 0-3 are "high-speed" lines and are used for computer-to-computer communications. High-speed lines 0-1 are 12b only, lines 2-3 support 6b and 12b. For simulation purposes, the high-speed lines can be ignored.

The 7750 has two basic operations: read and write. Read is used to drain accumulated input from terminals and is an asynchronous operation. Write is used to output to terminals and is a synchronous operation. Reads occur in response to user input and is signaled by the ATTENTION1 signal. Writes occur in response to program operations.

The 7750 relied on local echo ("half duplex") to provide timely feedback for typed characters. This includes inserting LF after CR on ASCII terminals.

The 7750 is initialized by sending it a 36b word of all 1's.

4.5.1 Read Messages

Read messages consist of a string of 12b characters, aligned to a 36b (word) boundary. The message terminates with at least one EOM (end of message) of 3777; if the message must be padded to a 36b word boundary, the padding also consists of EOM's.

The first character is the transmission number, which counts modulo 2048. There is no apparent protocol for initializing this count. CTSS ignores message number mismatches and resynchronizes after each message. From the comments, one error is to be expected, i.e., an error on the first message. According to Stan Dunten, this feature was added to debug a problem of dropped messages between the 7750 and the 7094; once the problem was found, the feature was no longer used.

After the message number, the read message consists of pairs of line number and input characters. The line number character is expected to be between 1024 and 2046, that is, to have the 2**10 bit set. If that bit is clear, the character pair is discarded.

Input characters can be either control or data characters. Control characters have the 2**10 bit set. There are six control characters:

```
2001          dialup
2002          end ID sequence
2003          interrupt
2004          quit
2005          hangup
30nn          completed typeout of nn (<= 31) characters
```

Dialup is the first character received from a newly dialed in line. It is followed by the ID sequence in subsequent data characters. The first data character following dialup contains the device ID:

```
<0:6>         discarded
<7:11>        device ID
              1 = KSR-35
              2 = 1050
              3 = Telex
              4 = TWX
              5 = inhouse TWX
              6 = KSR-35 standard
              7 = KSR-37
              8 = 2741
              9 = ESL scope
```

The low order 6b of subsequent data characters are shifted into bits <6:35> of the UNITID field until an end ID character is received.

Interrupt and quit are generated by specific character sequences on each input device (for example, BREAK generates quit on a KSR-28). The keyboard mappings for these keys are not documented in the CTSS listings. Hangup is generated when a line disconnects.

The 7750 sends data characters in 1's complement form. This is evident from two tables, one for identifying the spacing count, the other for character conversion. The former has entries for 001-137, 162, 166, 167, that is, 176-040, 015 (CR), 011 (tab), and 010 (backspace). The latter clearly shows the BCD equivalent of A for entry 76 (~101), B for entry 75 (~102), etc. Stan Dunten speculates that IBM's engineers didn't understand or didn't like the serial line protocol that uses 1's as the idle line state and made 0 the idle state, effectively inverting the protocol.

Data characters consist of 7b plus parity. IBM terminals and the KSR-37 generate even parity. Parity is checked on the incoming (1's complement) character. All other terminals do not generate parity.

4.5.2 Write Messages

Unlike read messages, write messages are always for a single line. There are two formats for write messages: control messages and data messages. Control messages are always one 36b word long and contain:

```
<0:11>          3000 + line number
<12:23>         control function
<24:35>         7777 (end of medium)
```

The only defined control function is all 0's, which resets a line and releases all buffered characters (see flow control).

Data messages consist of a 12b line number, a 12b character count, and then either 6b or 12b characters, followed by end of medium. If the message is 6b characters, then bit 2**10 in the line number is clear, and end of medium is 077; if 12b, then bit 2**10 in the line number is set, and end of medium is 07777. The largest write that CTSS can do in one operation is 94 words (559 6b characters). In practice, 6b mode is used only for high-speed lines, that is, for computer-to-computer communication without character translation.

In 12b mode, data characters consist of 7b plus parity plus start bit. As with input, the actual character is 1's complemented; this is evident from the code conversion table. The start bit is inserted by code for data characters but comes from a table for control characters. Because of the 1's complement coding, the inserted start bit is always a 1. Only the KSR-37 and ESL scope require even parity; other devices do not.

Character 3777 is treated specially. It causes the 7750 to repeat the last bit sent for the number of bit times specified in the next character. If the last bit was a space (idle), as is always the case for ASCII devices, then 3777 holds the line idle for the specified number of bit times. This is used to add delay for positioning characters like CR, LF, and TAB. If the last bit was a mark, then 3777 produces a break sequence, which was needed for the IBM terminals. The special sequences for the KSR-37 are:

```
CR              753,3777,0
TAB             755,3777,1
FORM FEED       747,3777,240
CR-NO-LF        345,3777,24
LF              333,333,0
POFF            711,633         (printer off)
PON             711,211         (printer on)
VT              351,3777,30
BLACK           711,277
RED             711,631
```

The BCD equivalents are (where UC implies values >= 0100):

```
UC 07 =>        PON
UC 16 =>        LF
UC 17 =>        VT
UC 36 =>        POFF
LC 52 =>        FF
LC 55 =>        CR
UC 61 =>        CR-NO-LF
LC 72 =>        TAB
UC 72 =>        RED
UC 75 =>        BLACK
```

For the KSR-35, some equivalent sequences:

```
CR              345,3777,0
TAB             355,3777,22
FORM            347,3777,54
```

The KSR-37 treats LF as CR-LF on output, and recognizes an auxiliary code (022) as bare LF. The KSR-35 treats CR as CR-LF on output.

4.5.3 Flow Control

On input, the 7750 relies on the "natural" balance between slow typists and a "fast" CPU to keep buffers from overflowing. But on output, a program can generate data much faster than the terminals can output. The 7750 cooperates with the operating system to implement global flow control on output.

CTSS tracks the number of characters of 7750 buffer consumed by a given user. It increments this total each time a buffer is output. The 7750 sends asynchronous “output complete” messages to let CTSS know when to decrement the user's character count. Output complete messages are sent at least every 31 characters, as the maximum character count in the message is 31. I suspect that they are also sent if the user's output line goes idle (all characters output).

4.5.4 Possible Simulation Strategy

- Ignore the high-speed lines; start the effective line count above the high-speed/low-speed boundary. This eliminates 6b mode.
- Simulate only a single type of terminal. The KSR-37 would seem to be the easiest, as it requires trivial code conversion and no tracking of shift codes. Red/black could be handled by ANSI compatible bold/normal sequences, for ANSI-compliant Telnet clients.

4.5.5 Unanswered questions:

- What keystrokes generated interrupt and quit for various terminals? Stan Dunten recollects that BREAK was used for quit on all ASCII terminals.
- Are there other output control messages in addition to line reset?
- Was 6b mode used for anything other than the high-speed lines?

4.6 Channel F

Unused.

4.7 Channel G (7289)

Channel G is a unique 7289 (possibly 7389) channel supporting two "high-speed" 7320A drums. The programming model for this drum is a hybrid between the select-driven model of the 7607 and the channel-driven model of the 7909. Read or write select is done with RDS and WDS, respectively, and data transfers with IOCP and IOCD, just like the 7607; but the channel takes the first word of the channel program as a drum address, and responds to the SCDx command, like the 7909. In addition, the drum channel has unique characteristics for storing information on SCD (store channel diagnostic).

Each physical drum has 192K 36b words, organized as six "logical" drums of 32K 36b words each. Each "logical" drum, in turn, is divided into 16 "chunks" (sectors) of 2048 words. The drum channel supports some number of physical drums, but CTSS only has 2. Drum addresses to the channel have the following format:

```
bits<3:5>      physical drum
bits<15:17>    logical drum
bits<21:35>    sector (word) address
```

The high-speed drum controller in CTSS responds as unit 330 on channel G. RDS and WDS prepare the controller for a read or write operation. When the channel is started (with LCHx or RCHx), the first word of the channel program is the drum address. The second and subsequent words are normal 7607 commands, of which CTSS uses only IOCP and IOCD.

When a drum operation completes, and the channel disconnects, if the enable bit is set for channel G, a data channel trap occurs. The trap location contains the standard 7607 error indicators in <15:17>: end of file, redundancy check, other error. SCDG returns 36b of channel error status. These are not defined, but bits <0:2> and <13> are considered to be errors; bit <0> is apparently I/O check.

Acknowledgements

First and foremost, I want to acknowledge the help of the CTSS team, whose continued interest in, and lively recollection of, the project made possible the salvaging of the sources as well as resolution of many obscure points. Tom Van Vleck, Stan Dunten, Jerry Saltzer, Donald Widrig, Dan Edwards, Bob Fenichel, Bob Daley, Roger Roach and Karolyn Martin have read all the drafts of this note and supplied corrections to errors and anecdotes that illuminated operation of the system. Needless to say, any remaining errors are my responsibility.

Paul Pierce did the mammoth task of transcribing the CTSS source tapes to online form. Paul also wrote the first 709 simulator, which contributed substantially to my knowledge of the architecture, and transcribed key hardware documents for online use. Dave Pitts extended Paul's simulator to be a full 7094 and got it to run IBSYS successfully. Dave's cross-assembler and cross-linker for the 7094 have made it possible to create executable modules from the CTSS source set and will be key to any reconstruction of an executable CTSS image. Rob Storey's 7094 simulator has also been very helpful.

Last, but hardly least, Al Kossow's multi-year project to transcribe manuals for online use has been critical in this effort, as in so many others. Al transcribed not only extant 7094 manuals, but also 704X manuals, which provided documentation about the interval clock and the 7631 file control.

Web resources

Paul Pierce's CTSS source set: http://www.piercefuller.com/library/ctss.html

Al Kossow's document archive: http://bitsavers.org/pdf/

Tom Van Vleck's CTSS page: http://www.multicians.org/thvv/7094.html