SESSION REPORT

SHARE

67	0317	Setting a Computer's Clock Correctly	350-400
HARE NO.	SESSION NO.	SESSION TITLE	ATTENDANCE
MVS SCP		Bill Choate	CAG
PROJECT		SESSION CHAIRMAN	INST. CODE

Contel Cprporation, 400 Embassy Row, Atlanta, GA 30328 (404) 393-0866

SESSION CHAIRMAN'S COMPANY, ADDRESS, and PHONE NUMBER

Setting a Computer's Clock Correctly: Does anybody know what time it is?

Dave Crow
TUCC, Box 12076
Research Triangle Park, NC 27709
Installation Code: TUC
MVS System Control Program (MVS SCP)
Session 0317

ABSTRACT:
Keeping a clock set to the correct time is a problem that has been with us for thousands of years, and the technology exists to completly solve the problems that we face in the computer industry. A brief summary of this technology and its history will be presented followed by some examples of how it has been applied to present day operating systems. These examples range from simple validity checks to systems that must keep their clocks within a micro-second of each other. The material is equally applicable to both VM and MVS systems, and should be useful to any installation that has ever worried about what can happen if their clock is set wrong.

INTRODUCTION

About three years ago, my boss walked into my office and asked if we could use a box that received the National Bureau of Standards time signal to eliminate the possibility of setting the computer's clock wrong. We now have a system in place that assures the clock on our system is within a few seconds of the correct time, and our operators haven't set the time for over a year. In this paper I will present some background material and then describe what we as users of MVS and VM systems can do to avoid errors in the TOD clock. At the end of the paper I will describe how my company uses an external clock to achieve accuracy of a few seconds, and will also describe a system in use at NASA in Houston that keeps their clocks within a few milliseconds of the correct time.

Setting the time wrong costs money.

Twenty years ago, an error in the date didn't cost too much because very little code expected the date to be right in the first place. But lately, more and more applications assume the time is correct. Disk and tape management systems, automated production control systems, billing algorithms and so on all depend on a correct date and time. As this happens, we are hearing more and more stories about things gone wrong. These stories tell of major outages while people are called in and the damage is repaired or at least minimized. Of course, even if you get the correct date, there is still the problem with the correct time. Many billing systems give reduced rates for off-peak usage, and a mistake of setting 0100 instead of 1300 can be very expensive even if it causes no other trouble. Today's computer systems need the correct date and time in order to function correctly, and we can be sure that the cost of a mistake will only increase as time goes on.

The operator is not the problem.

So what can we do about it? Are we totally at the mercy of the operator to set the date and time correctly? No. And in fact, it really isn't fair to put that kind of load on the operator in the first place. The computer clock is always set when things are very confusing. Usually there has been a major failure, or at the very least, the machine has been down for maintenance and there is some doubt about how well the system will work. The operator is always under a lot of pressure to set the clock quickly, and there is no verification of what the operator enters. The system does not make any reasonability checks of any kind, it just accepts the value of the date and time without question. Simply put, that is bad design. Adding code to verify that a reasonably correct date and time has been set is not expensive, can be done without modification to the system, and doesn't even require PLMs or microfiche.

Before we look specifically at computer systems, we will first cover some basic concepts:

* The Calendar
* The Clock
* Time of Day
* GMT and Time Zones
* How Long is a Second?

CALENDAR

The Egyptian calendar.
The Gregorian calendar of today evolved from the 365-day calendar of the ancient Egyptians that was in use in 4000 BC. Since the Egyptian calendar didn't have a leap year, the first day of the year drifted slowly through the seasons, completing a cycle every 1460 years, which the Egyptians called a Sothic cycle.

The Julian calendar.
The ancient Roman calendar was not a very good one, it only had ten months, and needed constant adjustment to keep it in step with the solar year. Over the years, some improvements were made, but the calendar was never satisfactory. So, in 46 BC, Julius Caesar decreed a replacement. The new calendar was based on the Egyptian calendar and included a leap year every four years to correct for the Sothic cycle. The new calendar had month names based on the old Roman calendar, which is why the last four months are named for the numbers seven through ten: September, October, November, and December. A few years later, because of mistakes made in calculating leap year, Caesar Augustus ordered another calendar correction, and in honor of himself changed the name of the month Sextilus to August. The Romans did not have days of the week as we know them, and years were always known in relation to the current emperor. Between 300-400 AD, the Christians brought the idea of days of week over from the Jewish tradition, and renumbered the years in honor of Jesus. This calendar was used for the next 1000 years, and is called the Julian calendar.

The Gregorian reform.
The Julian calendar has an average year of 365.25 days, but the solar year is a little shorter than that. Therefore, every four years is too often to have a leap year. By the 14th century, the Catholic Church was getting worried: Easter was slipping toward summer, while Passover was staying put in spring. This was bad because the Last Supper was a Passover celebration. (The Jewish religious calendar is very complicated, but it keeps the correct relationship to the solar year). In 1582, Pope Gregory XIII issued a papal bull that, among other things,

included a 10-day correction in the calendar, and changed the rules so that century years that are not a multiple of 400 are not leap years. Thus in the Gregorian calendar 1700, 1800, and 1900 were not leap years, but 2000 will be.

As you might expect, changing the calendar is a rather emotional issue, and was not adopted everywhere immediately. Britain (and her colonies) did not change until 1750, Russia adopted it in 1918, and Greece did not change until 1923!

So for historical purposes, dates between 1582 and 1923 must be checked to be sure which calendar is being used. But for today, it means that we now have a single calendar in use throughout the world. It might not be as good as it could be, but hopefully nobody will try to change it any time soon. Getting the date right is hard enough without having to worry about which calendar to use.

Julian dates.

You might think that Julian dates have something to do with the Julian calendar, but they don't. Julian dates were invented in the 14th century by a French scholar named Joseph Justus Scaliger. He wanted to correlate events that had been recorded by different cultures, but because so many different calendars had been used, it was nearly impossible.

Scaliger's system as it is used today is the number of days since noon, January 1, 4713 BC. For purposes of eliminating the confusion caused by multiple calendars nothing could be simpler; you just translate all calendars to this one. Unfortunately, Scaliger named his calendar in honor of his father Julius, so a date in his scheme is called a Julian date! For example, the Julian date for August 10 is 2,446,653.

In the computer industry, the date is sometimes written in the form yy.ddd, and some people call that the Julian date. That usage does not seem to exist outside the computer world.

WHAT'S A CLOCK?

A clock is a gadget used to tell time. It has a frequency source to measure a time interval, a counting mechanism, and a display. In ancient times, clocks were either sundials or water clocks. Today they are electrical or mechanical, but the principles are the same.

The earliest clocks.

The movement of the stars and the sun has always been used for determining time of day or day of year, and our calendars and clocks simply keep track of those motions. Up until the 12th century, time of day meant time as indicated by a sundial, and on a sundial the length of an hour varies from season to season. When the clock was first invented, some clock makers tried to make their clock's hour match the sundial, but as clocks became accepted the hour began to be regarded as a fixed length of time, so sundials started showing the correction instead. Today the idea of an hour as a fixed length of time is so widely accepted that most people find the idea of a variable hour a little quaint.

The evolution of accurate clocks.

In the 12th century, Christian monks invented the mechanical clock (with an alarm) to keep them from sleeping through matins (midnight prayer). In the 15th century, astronomers used pendulum clocks accurate to a second a day to measure the motion of celestial bodies. In the 17th century, spring-driven clocks accurate to better than a second a day were invented for use in navigation. Today, atomic clocks (clocks that use atomic oscillations as their frequency source) gain or lose only a few nanoseconds a day, and we have many applications in geophysics, astronomy, navigation, and other fields that require that kind of precision. The motion of the earth is much too irregular for these applications, in fact, one of the applications for accurate clocks is to measure the irregularities in the earth's motion. However, not every application needs an atomic clock. Figure 1 shows some examples of what kind of precision is needed for various application.

Errors in clocks.

One of the most accurate clocks that we have in daily life is the normal electric wall clock. It will stay very close to the correct time indefinitely (as long as the power stays on). But that doesn't happen by accident. It happens because somewhere back in the plant where the power is generated there is an electric clock. Whenever that clock gets too far ahead they slow down the generators. Whenever it gets too far behind they speed them up. So the apparent accuracy of a wall clock comes from a very determined effort to keep it adjusted to the correct time.

Clocks always have some error built into them. The clock can simply be set wrong to begin with. The frequency source will have some irregularity, and it can be fast or slow. The counter may not always count properly, and lastly, there is always some uncertainty in the display.

Clocks always have some error, so when choosing a clock, you must answer the question "How much error can be tolerated?" The clock that Captain Bligh took with him on the H.M.S. Bounty in 1789 was better than the one on a 3081. But then nobody tries to navigate using the clock on a 3081. That doesn't mean that there aren't requirements for more accurate clocks on a mainframe CPU. If a data base is being updated from two CPUs in a shared DASD environment, it would be very simple to maintain a transaction log if the clocks were known to be within a millisecond of each other. Studies of communications networks would be much easier if each site could simply log the time a packet was handled. Instead, in both cases, very elaborate schemes are needed to reconstruct event sequences.

Gross errors in a clock cause problems that everyone recognizes. Small errors have a more subtle effect: applications simply are not done. If an application requires that two independent computers have clocks set within a second (or less) of each other, or if it needs to measure a very precise interval of elapsed time, then the application cannot work and is never developed in the first place.

STANDARD TIME, TIME ZONES, and GMT

GMT was developed for celestial navigation.
Historically, the first real requirement for a standard time was for celestial navigation. Prior to the availability of good clocks on board ship, navigation was very much hit or miss. Ships had no good idea of their east-west location so they set sail to the proper north-south latitude and they turned east or west until they saw land. This took a lot longer than sailing a true course, but it was a lot safer. If you have a good clock, you can set it to local time at a known location (say Greenwich, England). Then at an unknown location you can measure the local time based on observations of the sun or stars, and calculate your east-west location from the difference between the clock and local time. Once chronometers were invented, navigators could choose any course they wished without fear of getting lost.

Because of England's dependence on ocean travel, they were the first to produce chronometers capable of accurate navigation, and as a result, developed navigation tables based on the location of the observatory at Greenwich, England. Navigators from other countries simply adopted the British work as a standard.

Time zones were needed by railroads.
Time zones developed a little later, for different reasons. As long as everyone stays around a small town, a single clock in the town square will serve quite nicely as a standard. All you have to do is keep the clock more or less on time with a sundial, and everybody can go about their business with no confusion. That is, there will be no confusion as long as travel between two such towns is slow.

With the coming of the railroad, also came confusion about the time. People couldn't keep track of schedules because they couldn't keep up with which clock was being used. With the invention of the telegraph it was a simple matter for all stations on a line to use the same time. No longer should the town clock be set by a sundial, instead it should be set to railroad time. This will work for a whole country, unless the country is very wide like the United States, in which case you need to divide the country into a few time zones just to keep some semblance of local time.

Standard time and time zone agreements.
By the end of 1884, international agreements had been reached that established Greenwich, England as the location of the zero degree longitude line, established 24 time zones at 15 degree intervals in international waters, and established GMT as the standard time zone for navigational purposes. Within national boundaries, however, countries are still free to use any time they wish. Most keep the one hour intervals and move the boundaries east or west to accommodate local interests, but some use half hour offsets, and others still use true local time so their offset from GMT is hours, minutes and seconds. Many countries also use daylight savings at times during the year. Figure 2 shows the time zones on a world map so you can see how imaginative local politicians have been.

HOW LONG IS A SECOND?

How long is a day?
One of the famous programming mistakes of all time was by the programmer who thought the earth revolved on its axis once every 24 hours and used that fact to calculate landing sites for the Mercury space flights. Our first astronauts spent a lot more time in the water than they should have because of that. Our day is a solar day. That is the time the sun takes to return to the same position in the sky, and is around 4 minutes longer than the time the earth takes for one revolution on its axis. The extra time is because the earth has moved in its orbit around the sun. However, since the earth's orbit is an ellipse, the solar day has different lengths at different times of the year. That's easy for a sundial to account for, but very difficult for a mechanical clock to handle. So the idea of mean solar day was developed. In a mean solar day, the length of a second is picked so that a 24 hour period is equal to the average solar day for the year.

The definition of a second.
Prior to 1972, a second was defined so there were exactly 24 hours of 3600 seconds each in a mean solar day. Unfortunately, the mean solar day changes slightly from year to year, and that meant the second and everything based on the second had to change slightly. For instance, the transmitters that transmit standard frequency signals had to be adjusted slightly so their frequency would stay the same. This presented all kinds of problems so the definition of a second was changed to

9,192,631,770 oscillations of a cesium atom. This solved the problem of a variable length second, but raised the problem that a mean solar day is not an integral number of seconds. To keep standard time about the same as mean solar time, the idea of a leap second was invented.

Leap seconds.
Leap seconds work like this: Several governments maintain departments that keep track of standard time using atomic clocks. Periodically they compare the time as measured by the atomic clocks to mean solar time as determined by an observatory. Whenever the difference exceeds 0.5 seconds, the International Time Bureau (BIH) in France directs that a leap second be inserted or dropped. Leap seconds pose a problem if you know the date and time of two events and need to know the elapsed time. To solve this problem, you must either keep tables that show when leap seconds occurred, or you must include leap second information in the time stamps.

Coordinated Universal Time.
The introduction of leap seconds means that standard time can now differ from Greenwich Mean Time by up to a second, so standard time is now called UTC (Coordinated Universal Time). UTC can be obtained to an accuracy of a few milliseconds world-wide using inexpensive radio equipment, and microsecond accuracy can be achieved at modest cost. For most purposes, the difference between UTC and GMT is not important, so the GMT name survives.

370 TOD CLOCKS AND RELATED CALCULATIONS

What is the TOD clock?
The Time of Day (TOD) clock on the 370 computer is at first an imposing design. It is a 64-bit unsigned number that is incremented so that bit position 51 has a value of 1 microsecond. The clock cycles every 143 years. According to the Principles of Operation manual, it should be set so that the current time is measured as an interval from 00:00 January 1, 1900 GMT, ignoring leap seconds.

Figuring out what that definition means requires knowing most of the seemingly unrelated material that has been covered so far. You will also see that the TOD clock value is a fiction. The length of a second wasn't even fixed until 1972, and the Principles of Operation manual says not to use leap seconds.

Calculating a TOD value.
Nevertheless, if you assume uniform days, hours, minutes, and seconds, calculating a TOD clock value is fairly easy. Figure 5 shows code that converts the year, day of year and time of day to a TOD clock value. The key to this code is the DAY-OF-

CENTURY calculation. That turns out to be a handy result, so it will be described in depth. The rest of the example is just some tricky code to do arithmetic with results that go over 31 bits.

Calculating the Day of Century.
Loosely speaking, you get the number of days in this century by multiplying the number of years times the days per year and adding that to the day of the year. However, the days per year is a fraction (365.25) so that complicates matters. It is a pretty safe bet to guess that the floor or ceiling functions will get involved, and a few minutes of playing with a calculator gives the formula:

```
day of century = CEIL(yy X 365.25) - 1 + ddd
(The -1 comes in because 1900 wasn't a leap year)
```

Since the TOD clock started at zero on January 1, to get a TOD clock value, you subtract 1 from the day of century, multiply by the microseconds in a day, add in the microseconds since midnight, and shift left 12 places.

Calculating the Day of Week.
If instead you want day of week, you take the remainder after dividing by 7. January 1, 1900, was a Monday, so the remainder corresponds to day of week as follows:

0 - Sunday	(Sun's day)
1 - Monday	(Moon's day)
2 - Tuesday	(Tiw's day)
3 - Wednesday	(Woden's day)
4 - Thursday	(Thor's day)
5 - Friday	(Frigg's day)
6 - Saturday	(Saturn's day)

For instance, August 10, 1986:

```
CEIL(86 X 365.25) - 1 + 222 = 31633
31633 divided by 7 = 4519 r 0
So August 10 must be a Sunday.
```

Figure 5b shows code to implement this. Other date calculations are simple extensions of this; Figure 5a shows how you can go about calculating the last Sunday in October.

Using the TOD value for a time stamp.
The TOD clock format turns out to be extremely easy to work with, and is a very useful way to time stamp an event. The low order bit in the first word is in units of about a second, so for human type actions, only the first word is important and double precision arithmetic is not needed. The TOD clock simply times a very long interval, so you can readily calculate the length of time between two events by just subtracting without having to worry about year end, leap days and all the other

stuff that makes working with dates tricky. Likewise, if you want to know the date 100 days from now you simply add the TOD equivalent of 100 days to the current TOD value and convert that to a date. In working with the TOD value, it is usually best to convert it to units of microseconds since 1900 by shifting right 12 bits.

HOW MVS AND VM USE THE TOD CLOCK

Both MVS and VM set the TOD clock using GMT, but they both recognize the need to have local time available, so they maintain local time as an offset from GMT. Both systems have problems in the way they handle the two times, so installations that have no need for using GMT frequently run local time in the TOD clock, and use a local offset of zero.

VM support for local time.

In VM the nucleus assembly specifies the difference between local time and GMT in hours, minutes, and seconds, and whether the zone is east or west of GMT. This is assembled as a signed number of seconds that must be added to GMT to get local time. When the operator sets the clock under VM, he specifies the time and date in local time. VM subtracts out the offset to get GMT and uses that to set the TOD clock. The operator does not need to know about GMT.

Problems with VM.

When you need local time VM does not simply add the local offset back in, instead it keeps track of the TOD value at midnight and subtracts that from the TOD value to get the interval into the day. This approach makes changing the local offset difficult, and VM requires a nucleus assembly and an IPL to change from daylight time to standard time. That's two IPL's a year just to change a constant that should have been a variable.

In displaying the time, VM always uses local time, except for the EXEC 2 system functions that return the GMT date and time and make no provision for determining the local time. Many sites have had EXECs quit working when they changed to using GMT in the TOD clock instead of local time.

RSCS can also cause trouble with time zones. When you define a site to RSCS you specify the offset in hours from GMT. RSCS deduces whether you are in daylight savings time (based on U.S. law) and adjusts the time and time zone label accordingly. But not everybody in this country uses daylight savings time, and not everyone uses hour offsets from GMT. The RSCS support is very nice if it fits your situation, but it needs to allow exceptions.

MVS support for local time.

MVS uses a slightly different strategy to provide local time. The TOD clock is still set to the GMT time, but the offset from GMT is maintained in units that correspond to the first word of the TOD clock (1.04... seconds). To convert a TOD clock value to local time, you need only do a single ADD LOGICAL.

The local offset may be specified in PARMLIB member PARMTZ, and the local offset (but not the TOD clock) can be updated while the system is running by using the SET CLOCK command.

In general, the MVS strategy is very good, but the MVS implementation has a whole host of problems.

Problems with MVS support.

- **Setting GMT.**
  When the MVS operator sets the TOD clock during IPL, MVS displays both GMT and local time in nearly identical formats on successive lines. If the TOD clock (GMT) is wrong, the operator must enter the current date and time using GMT, not local time, and specify ",GMT" in his reply. Since the GMT date changes several hours before the local date does, the operator really has to be sharp to avoid a mistake. In addition to having to know GMT, the date must be entered in the yy.ddd format and nobody knows what that is without looking at a calendar.

- **Setting local time.**
  Once the GMT date is correct, a separate set of replies are then used to set the local time. Since local time is maintained as an offset from GMT, there are usually only two possible valid values for the local offset. Yet MVS not only makes it easy to set a bad value for the local offset, it is nearly impossible to set the local offset exactly because the value is calculated based on when you enter the reply.

  The possibilities for mistakes are endless, and MVS does all this when the operator is in a very confusing situation and under a lot of pressure to hurry instead of being careful.

  Probably the most common mistake is to set GMT wrong, but local time correctly so that the bad GMT and the bad local offset add to the correct local time. The whole mess is so confusing that people who have run MVS for years get confused.

- **The daylight savings change.**
  Once you get the GMT time and local offset set correctly the other problem is the daylight savings change. Unlike VM, MVS allows you to change the local time without an IPL so you should be able to make the daylight savings change on the fly. We even did it once in production. SMF data uses local time for time stamps, so it really got messed up. JES2 automatic

commands also use local time and they were halted. There are probably other problems, but those were enough for us. We IPL for the daylight savings change.

- **Local offset error.**
  Since local offsets are in seconds, but the system records it in units of 1.04 seconds, the local offset can never be exact (unless it's zero). For Eastern time, the local offset always has an error of at least 0.14 seconds, so an MVS DISPLAY TIME command usually shows a difference in the seconds position between GMT and local time. This should not cause any trouble until the accuracy of the TOD clock is improved substantially.

Since most folks are able to run their systems, it is clear that the problems with the date and time are not insurmountable, but it is also clear that both MVS and VM need to start making improvements in their time-of-day support. Already the IPLs needed for the daylight savings change are a significant fraction of the IPLs for a year in many shops. But for today, the risk of setting the date or time wrong is probably the worst problem that most shops face.

VERIFYING THE DATE AND TIME

Both MVS and VM accept a bad date with no questions. That makes absolutely no sense. Code to verify that the TOD clock and local offset are correct can be run early in the IPL sequence. For VM this could be in the AUTOLOG1 PROFILE EXEC, and for MVS it can run as a started task either prior to or just after bringing up JES. Any mistake that you discover this early can usually be corrected without any damage.

Verifying the date.
A program that runs at each IPL, shutdown, and at least once a day in between can record the current date and time. This program can then verify that the date at IPL is correct to within a day or two by requiring that an IPL be after the last recorded time, and not more than a couple of days later. That will eliminate most of the really pyrotechnic mistakes.

Use the day of the week.
Of course, if the operator has simply turned the calendar two pages instead of one you are probably still going to get the date wrong. However, operators usually know the day of week without using a calendar, so if you ask for the day of week you can usually detect errors that come from looking at the wrong day in the calendar.

Ask dumb questions, get dumb answers.
The design of verification routines deserve some care. If all they do is ask "is this right?", operators will get into the

habit of replying "yes". If you care about accurate information, all replies have to require thought. (ask "What day is it?", instead of "Is it Tuesday?"), and if the verification routine thinks the date is wrong, a reply like "USE BAD DATE" should be required in order to force the system to continue. I would not want to be the operator who made that reply with the date set wrong.

Verifying the time.

This technique for verifying the date works because most mainframe systems run at least a little every day, but the time of day can't be done that way. Gross checks on the time can be made by asking what shift is running. This will detect an error in which the operator forgets to use 24 hour time in setting the clock (0100 instead of 1300). Most other errors such as misreading the computer room clock or transposing digits can be detected by asking the operator to re-enter the time a couple of minutes after it was first set. For instance, if 12:15 was entered instead of 12:51, two minutes later the correct response would be 12:53, while the validation routine would be expecting 12:17. If the times match up, the operator is probably setting the clock correctly if the clock being used as a source is correct.

Don't trust an electric clock.

The computer room clock has been the source of a bad time more than once. If you use a regular electric wall clock, and have a power failure you have a pretty good chance of getting the computer clock set before somebody corrects the wall clock. Throw it out. Battery powered wall clocks that are accurate to a few seconds a year are not expensive, or you can get a wrist watch, take off the band and glue the watch to the console. In any event, make sure the clock has a 24-hour display.

Let the computer read the clock.

The easiest way to ensure an accurate time is with a clock the computer can read. For purposes of getting the time set reliably, probably the best clock is one that is physically separate from a computer. This can either be set once and forgotten, or it can include a radio that periodically sets itself to the standard time via satelite or radio broadcasts. However, any external source of the time will work. Some installations use other CPUs that they access via shared DASD. If you connect to another site via a TP line, you might be able to get the time from that site.

Check the local offset.

Once the TOD clock is set correctly, it is usually easy to verify that the local offset is correct. Most sites use only two local offset values: one for standard time, the other for daylight time. The change between them is made on predetermined dates, so given a correct GMT, a program can determine which of the two values should be in effect. Because of the ease with which MVS allows the local offset to be set to total garbage, this check is much more important for MVS than it is for VM.

TUCC EXPERIENCE WITH AN EXTERNAL CLOCK

As I described at the beginning of this paper, a while back, my boss walked into my office with a HEATHKIT catalog, pointed to a funny gadget and said, "Can you hook it up?" The funny gadget was a radio that receives the National Bureau of Standards time signal and puts it out on a standard ASCII interface. As we talked about it, several things became clear. If we could get this thing working, we could at least verify that the TOD clock was set properly. We might even be able to set the TOD clock automatically and take it out of the hands of the operators completely. Furthermore, we had wanted to change to running GMT in the TOD clock but had never had the nerve to try it because we were afraid of getting things fouled up, especially at night when the local and GMT date are different. With an independent source of the time, that would be no problem.

We got the clock, and eventually did all of this. Figure 7 shows a diagram of how the clock is hooked up today.

Antenna

The antenna is outside the computer room in order to get away from all the electrical noise. We have frequent thunderstorms in our area so the antenna is grounded to the building frame. Nevertheless, a nearby strike did knock the clock out one time, so we now disconnect the antenna during thunderstorm season.

Heath Clock

The HEATH clock receives radio station WWV from Boulder, Colorado. WWV transmits the day of year and the time of day once a minute, and can be used to get within a few milliseconds of the correct time. When the receiver decodes four consecutive times that match each other it updates its internal clock with the correct date and time. Since the WWV signal does not include the year it is maintained manually with switches that must be changed each year. Once the clock is set, the time may be read on demand through the ASCII interface. The clock is accurate to a tenth of a second when it has been set recently, but because we disconnect the antenna for several weeks at a time, it is sometimes off by several seconds.

One note of caution: The HEATH company has an outstanding reputation as a builder of home electronic gear. However, that is their market, and a customer used to the care that a mainframe vendor gives may find their support lacking. There are other clocks on the market that are more suitable for a mainframe environment. Some like ours, receive radio broadcasts to set themselves, others are set manually.

Battery Backup

Without the battery backup the clock would be nearly useless. We do not have a UPS, so we need the clock most after a power failure. The HEATH clock can take up to a day to initially set itself given our antenna and location, so without the backup it will not be available after a power failure.

RS-232 Interface

The clock provides a terminal interface (DTE) and the Memorex 1270 requires a modem interface. The RS-232 interface cable performs that conversion and sets up the interface so a write from the 1270 triggers the clock to send its time out.

Memorex 1270

The 1270 is a TP box that provides a 2703 type interface. The programs that read the clock allocate the Memorex port directly without any TP access method such as TCAM. A 37X5 running EP can do the same thing, but because the 37X5 is loaded by the operating system you cannot always get to the clock at the appropriate time.

Experiences With the External Clock.

The clock arrived in kit form (you can also get it assembled), and like all HEATHKITs, the assembly was straightforward. The clock worked the first time I turned it on, but with the indoor whip antenna it took about a week to set itself. By then we had a program working that could read the clock. We first used that to measure how fast the TOD clock drifts from the correct time, and determined it was almost a second a day. None of the CEs believed that it could be that bad and were going to fix it until they found specifications that said up to a 2-second error is acceptable.

I went to work on a verification program that uses the ideas I outlined earlier, while another programmer, Frank Schubert, started working on a program to set the clock. We weren't sure what the system would think if its TOD clock changed while it was up and running, so Frank took the approach of writing a stand-alone utility to set the clock. That makes the job extremely complicated, but it definitely involves no mods to the system.

By August, 1984, I had the validation programs written and in production, and we had installed the external antenna and battery backup. Once we were confident that we could depend on the clock, we got brave and made a change to use GMT in the TOD clock. Much to our surprise, that change went off without a hitch. A couple of months later, Frank got his TODSET program working and we have been using it for almost two years. For much of that time, we also ran a 370/168. To keep the CPUs set to the same time, we used a signal splitter to allow both machines to access the external clock.

At the August, 1984 SHARE, Jack Schudel of the Northeast Regional Data Center in Florida became interested in the project and within a few months they had gotten their own HEATH clock and were using it with our software. Jack made significant enhancements to the programs we sent him, and those improvements are now part of the programs that we run. One of the big surprises came when Jack noticed that their clock didn't always show the correct time. He developed a monitoring program which we picked up and used to determine that we had the same problem. In fact, we had a whole host of problems, including reporting a date of November 31, and spurious garbage in the output. Of course the worst problem was setting itself incorrectly. We observed an error of 3 days, and Jack saw one of 100 days. The HEATH people eventually resolved the problems and sent us new chips. We have not seen any significant problems in almost a year of continuous monitoring.

On a couple of occasions, the clock has broken and once had to be sent back to HEATH for repairs. For such times we have a small wrist watch glued to the 3081 service console that is set to display the GMT date, time and day of week.

Programs to use the clock.

We have developed three programs for use with the clock:

GMTREAD	- A diagnostic aid that simply reads the clock
TIMECHK	- Verifies the TOD clock and local offset are correct
TODSET	- Sets the clock (stand-alone)

GMTREAD

GMTREAD was written solely to test the read logic. We still use it as a diagnostic aid to see if the clock is working.

TIMECHK

TIMECHK runs under MVS/TSO or MVS/XA. It first reads the external clock and verifies that the TOD clock is set correctly. It also handles the year change and verifies that the HEATH clock switches are set correctly. Once the TOD clock has been verified, it determines if we are in daylight time or not, calculates the correct local offset, and checks that against the system value.

Once the TOD clock and local offset have been verified, TIMECHK updates a dataset with the date and time of this run. This data is then used in the next run to verify the date. The date verification logic requires that it be run at least once a day. We run it at 2 AM so we will be able to warn the operators when the daylight savings change takes place.

TIMECHK can be run as a normal started task or as a subsystem prior to starting JES. TIMECHK also has the monitor function that was used to chase down the bugs in the HEATH clock, and can be used to set the TOD clock if the computer only has one TOD clock.

TODSET

TODSET runs stand-alone and sets the TOD clock on a 370 mode CPU. It uses a 3270 or 3066 console for operator communication. When it is loaded it reads the external clock, calculates the new TOD clock value and requests that the operator press the TOD ENABLE switch. The program uses the CPU timer to measure the delay in enabling the TOD SET, so that the clock will be set correctly. Once set, it records the date so that it can properly handle a year change. TODSET supports multiple TOD clocks, and does not lose any accuracy due to the time it takes to synchronize the clocks.

Once the time has been set, TODSET selects an IPL unit based on the CPU serial number, displays the volume serial and waits for the operator to press enter. When that occurs, it loads the IPL text from the IPL unit and proceeds with the IPL of the true system. If the operator wishes, a different load unit can be requested for loading a different system. The new load address can be a temporary change, such as for loading a test system, or a permanent update can be made.

TODSET recognizes a VM system in need of a warm start, and bypasses the set logic in order to avoid damaging any warm start data. Once VM has saved warm start data and gone into a wait state, TODSET can be reloaded to set the TOD clock.

SOLVING THE TOD DRIFT PROBLEM -- THE NASA APPROACH

If you ask most people how close the TOD clock needs to be they will usually say a couple of minutes, but as we have learned, the accuracy really depends on the application. When I asked a programmer who works on the NASA ground control system what their TOD clock requirements were, he said, "Oh, they're pretty loose, only about 10 milliseconds or so." Our 3081 loses 10 milliseconds in 15 minutes. Setting the clock is only half their problem. They also have to maintain an accurate value.

The NASA system for maintaining the TOD clock uses an external clock and hardware RPQs (available only on 303x and 308x machines) to improve the TOD clock hardware. The RPQs allow the clock to be set very precisely and keep the TOD clock on time by using the external clock's frequency source. Once set, the TOD clock will stay on time indefinitely. The external clock that NASA uses is part of their overall system for keeping their various tracking stations synchronized, and should allow them to keep their TOD clocks within a few microseconds of the correct time.

The programs that set the clock run in a 370 or XA environment and can handle CPUs with more than one TOD clock. They run as MVS started tasks and set the clock after the system is up and running. They should of course be run early in an IPL.

In essence, the NASA system solves all problems with maintaining the correct time except for leap seconds. When a leap second occurs, something has to give. Either you don't change your TOD clock and accept a 1 second error (when 10 milliseconds is the goal), or you update the clock and give up the ability to do exact interval calculations. To properly solve the leap second problem, you need to be able to run the TOD clock without leap seconds, and keep track of how many leap seconds must be added to the TOD clock to get GMT.

SUMMARY

1. Calendars and clocks have evolved over the years to where there is a body of international agreements to say how timekeeping should be done. Computer people should not attempt to invent their own methods (and funny names), but should simply adopt what the rest of the world uses.

2. The TOD clock architecture is more than adequate for the needs of VM and MVS, but the present hardware implementation is inadequate for certain applications. Either hardware improvements will be made, or these applications will be done some other way.

3. Both MVS and VM make it much too easy to set the TOD clock wrong. However, customer run verification routines can be used to detect errors before any real damage has been done.

4. Verifying the time using an external clock is a reasonable way to insure that the TOD clock is set properly, but maintaining the time correct to better than a second a day requires hardware improvements to the TOD clock.

BIBLIOGRAPHY

IBM Corporation. *IBM System/370 Principles of Operation*, GA22-7000-9, May 1983, pp. 4.23-4.28

Kamas, George, and Howe, Sandra. "Time and Frequency Users' Manual", *National Bureau of Standards publication 559*, 1979

Landes, David. *Revolution in Time*, Belknap Press, Cambridge, Mass, 1983

Moyer, Gordon. "The Gregorian Calendar", *Scientific American*, (May 1982), pp. 144-152

Readhead, Anthony. "Radio Astronomy by Very-Long Baseline Interferometry", *Scientific American*, pp. 52-61 (June, 1982)

Ronan, Colin, et. al, "Calendar", *Encyclopaedia Britannica*, Macropaedia 3, 1986, pp. 595-612

Wardship, Schuyler, and Hocking, William. "NASA PTTI Programs: Present and Future", *Proc. 12th Annual Precise Time and Time Interval (PTTI) Conference*, Goddard Spaceflight Center, Greenbelt MD, Dec. 1980, pp. 151-198

Time Requirements for Various Applications

PRECISION	SOURCE	APPLICATION
1 minute	Radio, TV	Human schedules
1 second	Telephone	Celestial navigation
1 millisecond	HF radio (WWV)	Power company records
1 microsecond	LORAN-C	Radio-astronomy (VLBI)
10 nanosecond	Satellite/LASER	Spacecraft tracking

The precision of a clock measures how fast it gains or loses time. The precision listed here is not an absolute number, but rather an order of magnitude. Lets look at these applications.

People in today's world can get by with a clock that is off by 5 minutes, but a 10 minute error begins to be noticable. You miss planes, or you get out of a session late and all the Cokes are gone. If your clock loses much more than a few minutes a day, you will have problems.

In celestial navigation, an error of 1 second per day in the chronometer in the course of a 6 month voyage can amount to a 50 mile error. (I know that island is around here somewhere...)

Power companies use accurate clocks not only to keep your wall clock correct, but also to track down problems. A major outage can cause a cascading sequence of failures, and locating the cause can require knowing the exact sequence in which various parts of the system went down.

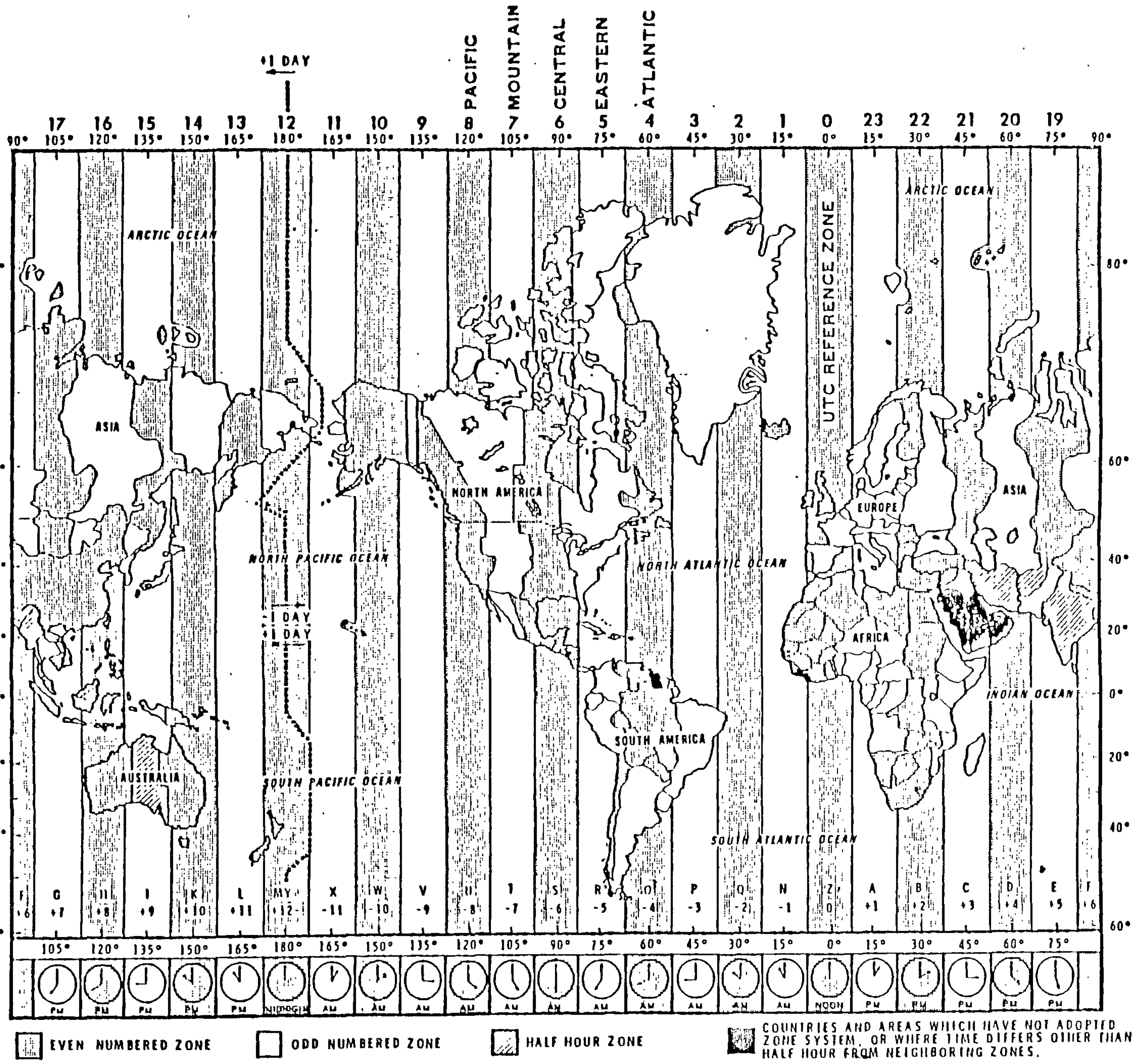
To me, one of the most fascinating uses of accurate times is in Very-Long-Baseline-Interferometry. Here signals from widely separated radiotelescopes are combined to provide an effective resolving power of a telescope nearly as big as the separation of the stations! Astronomers record what they receive at a rate of 4 million samples per second. They then combine the tapes from different stations using digital signal processing. Although final synchronization of the signals is done mathematically, the initial time stamp on a sample needs to be within a fraction of a microsecond to begin with.

Keeping track of satellites in orbit requires a worldwide network of radar stations. Apparently state-of-the-art equipment requires that each station be exactly on frequency. This correlates to being able to maintain the time at remote sites accurate to a few nanoseconds. Time transfer between remote sites at this level of precision can be done with LASER transmission from a satellite since the path time of the transmission is a constant.

FIGURE 1

FIGURE 2

Standard Time Zones



I

Example 3: Converting to TOD clock format

```
*   THE FOLLOWING VARIABLES ARE USED:
*        ARGYY    4 BYTE YEAR (BINARY)
*        ARGDDD   4 BYTE DAY OF YEAR (DDD BINARY)
*        ARGHUN   4 BYTE TIME OF DAY (BINARY, .01 SEC)
*        ARGTOD   8 BYTE OUTPUT TOD VALUE
*
*        FOR YEARS OTHER THAN ZERO, THE
*        DAY OF CENTURY = CEIL(YY*365.25)-1    + DDD
*        FOR YEAR 0, ITS JUST DDD
         L     R1,ARGYY
         LTR   R1,R1
         BE    TOD1                DAY OF YEAR 1 TOO HIGH UNLESS YEAR 0
         M     R0,=F'36525'
         A     R1,=F'99'           CEIL FUNCTION
         D     R0,=F'100'
*        SUBTRACT 1 TO GET CORRECT DAY OF CENTURY DUE TO YEARS
         BCTR  R1,0                (SINCE YEAR 0 ISNT A LEAP YEAR)
TOD1     DS    0H
         A     R1,ARGDDD
         BCTR  R1,0                ARGDDD IS 1 ORIGIN, WE NEED 0 ORIGIN
*   CONVERT TO MICROSECONDS: MULT BY 24  *  60  *  60  *  10**6
*   THIS FACTOR WONT FIT IN 32 BITS,
*   SO CHANGE IT TO:                3*8  * 15*4 * 15*4 * 5**6 * 2**6
*                OR: (3 * 15 * 15 * 15625) * 2**13
*   WHICH CAN BE DONE BY A MULTIPLY FOLLOWED BY A SHIFT OF 13 BITS
         SPACE 1
         M     R0,=A(3*15*15*15625)
         SLDL  R0,13
         L     R15,ARGHUN          TIME IN .01 SEC UNITS
         M     R14,=A(10000)       NOW IN MICROSECONDS
         SPACE 1
*   NOW ADD IN TODAYS TIME  (DOUBLE PRECISION ADD LOGICAL)
         AR    R0,R14              ADD HIGH ORDER PART
         ALR   R1,R15              ADD LOW ORDER PART
         BC    12,NOCARRY          CC=8:0,NOCARRY. CC=4:¬0,NOCARRY
         A     R0,=F'1'            PROPOGATE CARRY
NOCARRY  DS    0H
         SPACE 1
*   R0,R1 NOW HAS TIME IN MICROSECONDS
         SLDL  R0,12               CONVERT TO TOD FORMAT
         STM   R0,R1,ARGTOD        SAVE OUTPUT IN PARM LIST
```

FIGURE 3

Example 4. Converting from TOD clock format

The same variable names are used as in Example 3, but the routine runs the other way. This routine truncates the time of day to .01 second. To avoid that, save the bits dropped off in the first SRDL, and catenate that to the remainder following the divide by 10000.

```
*       CONVERT TOD value TO YY,DDD,.01SEC format
          LM    R0,R1,ARGTOD        GET INPUT VALUE
          SRDL  R0,12               CONVERT TO MICROSECONDS
          STM   R0,R1,ARGMIC        SAVE VALUE FOR LATER
          SRDL  R0,13               DO FIRST HALF OF DIVIDE
          D     R0,=A(3*15*15*15625) R1 IS DAYS IN CENTURY
*     This next statement is hard to convince yourself is true,
*     but if you keep close track of the units, you can see it is:
*        R0 AND THE LOW 13 BITS of ARGMIC IS MICSEC INTO THE DAY
*    DEAL WITH THE TIME OF DAY FIRST
          LR    R2,R0               GET HIGH PART INTO EVEN REGISTER
          L     R3,ARGMIC+4         GET LOW PART INTO ODD
          SLL   R3,(32-13)          BUMP THEM UP AGAINST EACH OTHER
          SRDL  R2,(32-13)          AND BACK DOWN TO LOW BITS
          D     R2,=A(10000)        CONVERT TO .01 SEC
          ST    R3,ARGHUN             (IGNORE REMAINDER)
*    NOW GET YYDDD FROM DAY PART IN R1
          A     R1,=F'1'            CONVERT FROM 0 ORIGIN TO 1 ORIGIN
          M     R0,=F'100'          AND DIVIDE BY 365.25 DAYS IN A
          D     R0,=F'36525'        YEAR. R1=#YEARS, R0 IS ROUGHLY
*                                     100*DAY OF YEAR.
          LR    R3,R0
          SR    R2,R2
          D     R2,=F'100'          R3 HAS # DAYS-1, EXCEPT FOR YY=0
          LTR   R1,R1               YEAR 0?
          BE    FRT1                YES, SKIP NONYR0 CORRECTION
          A     R3,=F'1'
FRT1      DS    0H
          ST    R1,ARGYY
          ST    R3,ARGDDD

ARGMIC    DS    D           WORKAREA TO HOLD TOD IN MICROSEC
```

FIGURE 4

Example 5a. Calculate date of last Sunday in October.

This code segment calculates the date of the last Sunday in October in the format yy.ddd. It uses a subroutine that converts a mm/dd/yy date into a yy.ddd date and also returns the day of week. Most people are familiar with code to convert from mm/dd/yy format to yy.ddd, so that code is not shown. Example 5b shows how to get day of week from the yy.ddd date.

```
          MVC   FALLBACK+6(2),CURYY  GET CURRENT YEAR INTO DATE
          LA    R1,FALLBACK               POINT TO PARM
          BAL   R14,MDYCNVT    GET YYDDD AND WEEKDAY OF 10/31/YY
*     NOW, R0 HAS YYDDD AND R15 HAS DAY OF WEEK (0=SUNDAY)
          ST    R0,OCTSUN
          CVD   R15,DBLWORK    CONVERT DAY # TO DECIMAL
          SP    OCTSUN,DBLWORK GET YYDDDC OF LAST SUNDAY IN OCT
          .


FALLBACK DC     C'10/31/YY'    DATE DST FALLS BACK TO STD TIME
CURYY    DC     C'86'          CURRENT YEAR
DBLWORK  DS     D              WORKAREA

OCTSUN   DC     F'0'           OUT: YYDDDC OF LAST SUN IN OCT
```

Example 5b. Calculate day of week given yy.ddd date

```
*    DAY OF WEEK IS MOD(DAYOFCENT,7)
*    DAY OF CENTURY IS CEIL(365.25*YY)+DDD-1
*          (EXCEPT FOR YY=0: DAY OF CENTURY = DDD  )
         L     R1,MDYYR
         LTR   R1,R1
         BE    MDY0                IF YEAR 0, ITS JUST DDD
         M     R0,=F'36525'
         LA    R1,99(R1)           DO CEIL
         D     R0,=F'100'
MDY0     DS    0H
         A     R1,MDYDDD
         BCTR  R1,0                LESS 1 (1900 NOT LEAP YR)
         SR    R0,R0               DIVIDE DAY OF CENTURY BY
         D     R0,=F'7'              7.  THE REMAINDER IS THE
         ST    R0,MDYDOW             DAY OF WEEK (SUNDAY=0)

MDYYR    DS    F                   BINARY YEAR  (YY)
MDYDDD   DS    F                   BINARY DAY OF YEAR (DDD)

MDYDOW   DS    F                   OUTPUT: DAY OF WEEK
```

FIGURE 5

Example 6a. Double precision add logical

Note: The result of an add logical is the same as a two's complement add except that overflow is not considered an error.

```
         LM     R0,R1,ADDEND1
         AL     R0,ADDEND2           ADD HIGH ORDER PART
         AL     R1,ADDEND2+4         ADD LOW ORDER PART
         BC     12,NOCARRY           CC=8:0,NOCARRY. CC=4:¬0,NOCARRY
         A      R0,=F'1'             PROPOGATE CARRY
NOCARRY  DS     0H
         STM    R0,R1,TOTAL          TOTAL <- ADDEND1 + ADDEND2

ADDEND1  DS     D                    FIRST ADDEND
ADDEND2  DS     D                    SECOND ADDEND

TOTAL    DS     D                    OUTPUT: ADDEND1 + ADDEND2
```

Example 6b. Double precision subtract logical.

Note: The result of a subtract logical is the same as a two's complement subtract except that overflow is not considered an error.

```
         LM     R2,R3,MINUEND
         SL     R3,SUBTHEND+4           SUBTRACT LOW BITS.
         BC     15-4,DELNOBOR   CC=4 NEEDS A BORROW
         SL     R2,=F'1'                BORROW ONE
DELNOBOR DS     0H
         SL     R2,SUBTHEND             SUBTRACT HI BITS
         STM    R2,R3,DIF               DIF <- MINUEND - SUBTHEND

MINUEND  DS     D                       FIRST NUMBER IN SUBTRACT
SUBTHEND DS     D                       SECOND NUMBER IN SUBTRACT

DIF      DS     D                       DIF <- MINUEND - SUBTHEND
```

FIGURE 6

TUCC Clock Installation

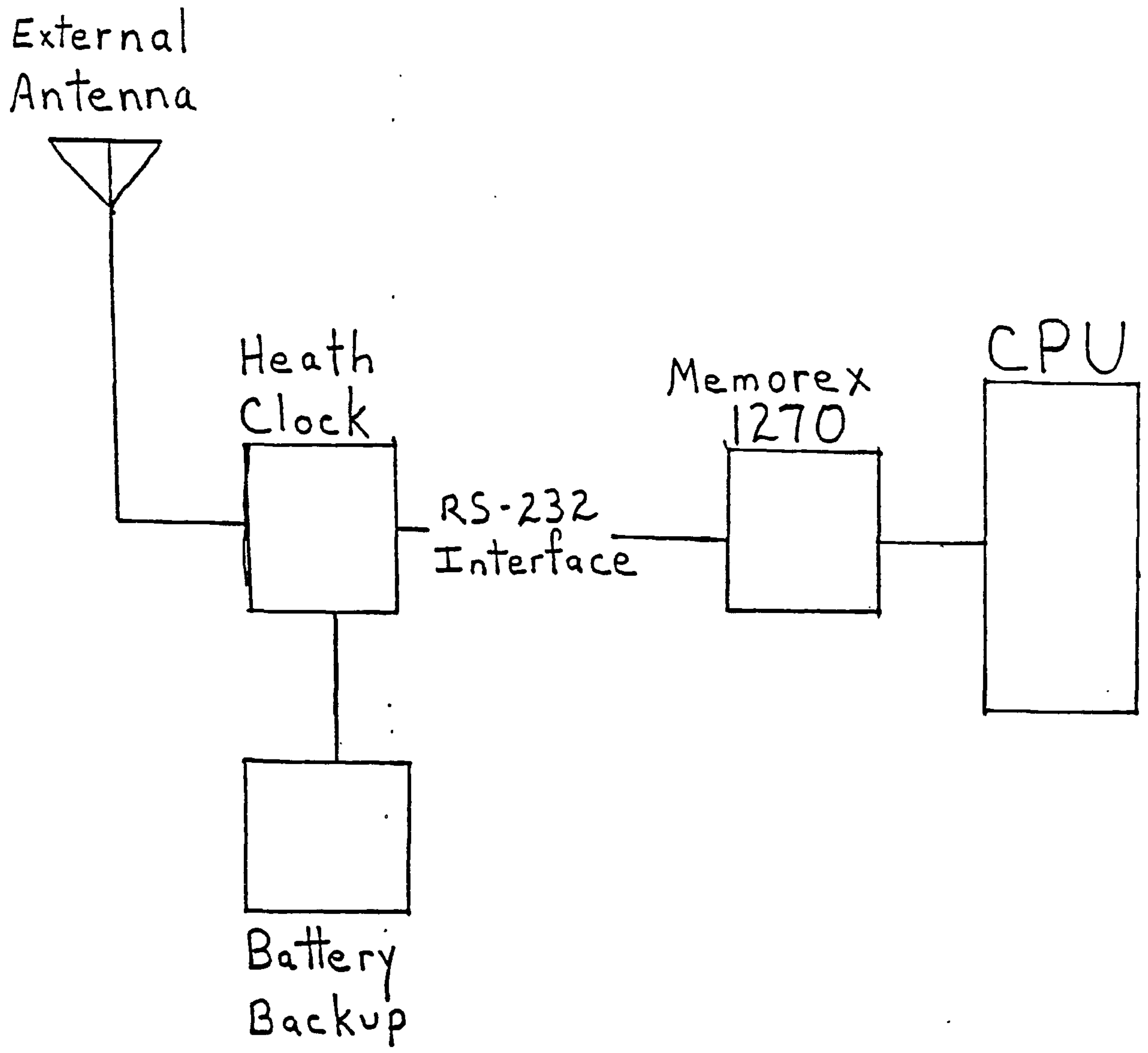


FIGURE 7

The TIMECHK program

The TIMECHK program checks the TOD clock agains the GMT clock and verifies that the local offset is correct. We run TIMECHK during IPL and each morning at 2:00 AM. The daily run insures that there will never be a big jump in the date, and allows detection of the daylight savings change as soon as it occurs.

Highlights of this program are:

* It reads the HEATH clock and verifies that the time is correct to within one minute. If the time doesn't check out, the operator must reply "USE BAD DATE" to continue.

* It records the date and time of execution and requires that time not go backwards, nor jump forward by more than 30 hours.

* When the external clock is not available, and in a couple of other conditions, the operator is asked to supply the date and day of week. They must be consistent to be considered valid.

* When the date changes from December to January, a new year is assumed, and if verified by the operator, it will be recorded for use in the next run. Following a year change, it requests that the switches in the clock be corrected.

* It verifies that the local offset is correct. The local offsets to be used for standard and daylight time are specified as assembly parms, and can be zero if local time is used in the TOD clock.

* TIMECHK runs as an MVS started task and can be run as a subsystem prior to starting JES.

FIGURE 8

The TODSET Program

TODSET is a stand-alone utility that is IPLed to set the clock. Once the time is set it can automatically IPL an MVS/370 or VM system.

Highlights are:

* It reads the HEATH clock and sets the TOD clock to the correct time. The operator must enable the TOD security switch when requested, but is not responsible for doing it at a precise instant.

* The TOD clock can be set using GMT or local time.

* After setting the clock, TODSET displays a suggested IPL unit and volume based on the CPU serial number. The operator can change the unit if desired, and then press "ENTER" to cause TODSET to IPL that system. The IPL unit change can be either temporary (for a test system), or permanent. TODSET makes the IPL unit making the selection based on CPU serial number, so only one copy of the program is required if multiple CPUs can access the same DASD volume.

* Console communication supports 3066 and 3270 consoles.

* When the date changes from December to January, a new year is assumed and is recorded for use by the next run.

* TODSET supports 370 mode CPUs, and will properly handle an MP with more than one TOD clock. It does not support the XA environment.

* It detects a VM system in need of a warm start, and immediately gives control to the VM system so that the warm start data is not damaged.

* TODSET does not require any modifications to the system, and can be bypassed if bugs are encountered.

FIGURE 9

TOD clock RPQ features

The system that NASA uses to keep their clocks synchronized with standard time requires two RPQs for their TOD clocks. As far as I can determine, these are available only on 303x and 308x CPUs. One RPQ simply adds a special DIAGNOSE code that allows the TOD clock to be set without operator intervention. The other has several features:

* Adds a READ EXTERNAL CLOCK instruction that reads the external clock value directly into memory without using normal I/O.
* Provides the ability to use an "on-time" pulse from the external clock so that the exact instant the TOD clock is set can be determined by the external clock
* Provides the ability to use an external frequency source to keep the TOD clock on time.

There is another TOD clock RPQ that is used by the airline industry for the Airline Control Program. This RPQ hooks up two CPUs and allows exact synchronization of their TOD clocks much like multiple TOD clocks in an MP. Unfortunately, this RPQ is mutually exclusive with the READ EXTERNAL CLOCK RPQ. The Airline RPQ is preferable to the READ EXTERNAL CLOCK RPQ when exact synchronization of the clocks is more important than being able to keep the clocks on time.

FIGURE 10