BITS,BYTES&PIXELS

LIMA 99/4A USERS GROUP

March 1994 Volume 10 #3

THE 1994 LIMA NUG CONFERENCE - REPORT #1
Friday evening/Saturday May 13&14
Reed Hall, The Ohio State University
Lima Campus, Lima Ohio

As in past years this all TI/Geneve event is TOTALLY FREE. There is no admission charge and no charge for tables in the exhibit area. Please note the correct dates listed above. Micropendium has in the past incorrectly listed the dates as May 14/15 1994. The Lima Campus is located just east of the city on the north side of State Route 309, 3 miles east of the junction of I75 and 309. There is ample free parking next to Reed Hall.

INFORMATION: To schedule a seminar, reserve free exhibit area tables, or for any additional information contact us in any of the following ways:

Write the Lima User Group at P.O. Box 647, Venedocia OH 45894.

Phone Dave Szippl evenings at 513-498-9713

Phone Charles Good evenings at 419-667-3131

Send internet messages to cgood@magnus.acs.ohio-state.edu

For a package of tourist information about the Lima area send a request to thre Lima/Allen county Convention & Visitors Bureau, 147 N. Main St., Lima OH 45801-4920.

TENTATIVE SCHEDULE:

Friday May 13 4PM-8PM. Equipment and table setup. User group representatives may copy software from the Lima software library at no charge. This is a good time to meet friends and socialize.

Saturday May 14 8AM-6PM. The main event. Seminars will be given more or less non stop during these hours. User group representatives may copy software from the Lima library. On site food service will be available in the middle of the day.

6PM-8PM. Clean up time.

8PM. After the show pizza party at a local restaurant.

HOTELS:

Call the hotel of your choice about prices and make your own reservation. Many hotels in Lima have nice rooms for under $40 per night. They are listed here in two groups; those at the most convenient location near I75 and route 309, and those a bit farther away.

Most convenient location, 5 minutes driving time to campus:

MOTEL 6 (This is the place where most people stay). 419-228-0456

HOLIDAY INN 419-222-0004

ECONOMY INN 419-222-1080

EAST GATE MOTEL 419-229-8085

DIELMAN'S MOTEL 419-225-2806

KNIGHT'S COURT 800-843-5644

Other hotels a bit farther away

RAMADA 419-228-4251

HOJO INN 419-228-7525

QUALITY INN 800-424-6423

DAY'S INN 419-227-6515

BEST WESTERN 800-528-1234

TRANSPORTATION:

The easiest way to get to Lima is to drive. The city is about mid way between Toledo and Dayton on I/S, a major north/south highway. From the west take US30 to route 309 and then 309 to Lima. From the east take US30 to I75 and then south to Lima.

Lima has no commercial air service, but we will TRY to arrange free transporation for you from the airports of nearby cities. Arrangements are made by members of the local user groups in these cities. If you are flying to DAYTON OHIO (most who fly to Lima use this airport) phone Dave Szippl evenings at 513-498-9713. If you are flying to COLUMBUS OHIO phone John Parkins at 614-891-4965. If you fly to FORT WAYNE INDIANA phone Homer Kipling at 219-483-8886. Unless arrangements can be made in advance with these people, you will have to rent a car and drive to Lima.

Lima is served by Greyhound bus.

PASCAL WITH THE TI 99
by Anders Persson

Charles Good wrote an article in the January issue of the Lima newsletter this year, concerning the p-code card for the TI 99/4A. When reviewing software, it's usually impossible to use it enough to really find out how to take the best advantage of the program. That's especially true when the review concerns something as complex as the UCSD p-system. Since I was mentioned in that article, I couldn't restrain from adding some of my own comments.

NEXT PAGE

The idea of the p-system

Charles asks what's portable with UCSD Pascal? As things has turned out, the answer to that question is, unfortunately, nothing. Due to the problem with different disk formats, there has never really been any portability between different implementations of UCSD Pascal. It's of course possible to transmit the source code via serial ports, with or without modems, between different computers, but that's about it.

However, there is something that really is portable with UCSD Pascal. The original Pascal definition, by Nicklaus Wirth, is practically useless for real world interactive programming. That led to different, more or less clever extensions to the original. Some extensions were targeted to improvement of the language's behavior in limited memories, while others improved the ability to program operating systems in Pascal.

Since features introduced with version IV.0 (the one implemented on the TI 99) has been an inspiration for various designers of Pascal compilers, a lot of Pascal code can be typed in and executed on the TI. This is true also for code neither originally intended for the UCSD system in general, nor for the TI in particular. I have myself transferred a substantial program (compiled EXE file on the PC is 55 kbytes) from the TI to Turbo Pascal version 4, with only minor source code changes. These were mainly related to the 80 character wide PC screen and somewhat different file handling procedures.

Since Turbo Pascal is continually being developed, in order to meet both new hardware (faster, bigger) and new operating environments (like Windows), the difference does of course grow with time. As Charles pointed out in his article, there is no current development of the p-system.

The TI implementation

The unique feature of the TI implementation is the p-code card itself. One reason for this card was probably the always apparent fear that, unless it was made impossible, people might copy software and execute it on their 4A's. By making the hardware p-code card an essential part of the Pascal system, that was much more difficult. But an advantage with this idea is that you get a ROM-disk, containing the operating system and the p-code interpreter, without wasting valuable RAM memory. We must remember that this was more than ten years ago, when 64 kbytes of RAM in a microcomputer was about as much as anyone had ever heard about. This was also a time, when running Pascal on a home computer, resulted in an impressed "Wow" from all computer friends that looked at it.

Since UCSD Pascal IV.0 wants to do a lot of things, it depends heavily on its memory management. There simply isn't room for everything at the same time. This results in a lot of disk activity when using the system. In the beginning, that was quite a nuisance, but since I made a RAM disk for my computer, that problem isn't so severe any longer. Adding four 360 kbyte disk drives (CorComp Controller) also makes the system run better, since all system software can be on line at the same time, together with the application under development.

With this system, I've done the major part of all software development on my computer. Built in memory management, the capabilities for structuring your programs that's inherent in Pascal, easy assembly interface and the adaptability of the system are some of the reasons. I've modified the p-system to include true pre-emptive multitasking, full screen turtle graphics and to take advantage of my redesigned 80 kbyte RAM console.

A disadvantage of the TI implementation is that it's not complete. The unit KERNEL, for example, isn't included on the disks, at least not with the interface section intact. That unit is essential for easy system programming. Another program that's missing is the Native Code Generator, which converts p-code programs to the assembly instructions of the processor used with the target machine, the 9900 in this case. This utility, when available, is used to speed execution of programs at the cost of code size.

To use the p-system

To make the system run faster, the SYSTEM.STARTUP program should copy all essential system files to a RAM disk, to begin with. That makes loading the Filer, a disk manager with features still difficult to find in later programs, fast enough.

I haven't had the chance to run any of the never released programs that Charles obviously has access to, but I can at least sort one thing out. A "fixed pitch printer" is one that prints like a typewriter, while a "variable pitch printer" is one capable of proportional spacing in its printout. Not an obvious feature for all low cost printers more than ten years ago.

It's not necessary to press "I" to initialize each time disks are replaced. The p-system allows you to refer to disks by name, not by number, and automatically tracks where a particular disk is located. If you move a disk to another drive, the system will look for it only once, and then remember where it's inserted. Myself, I rarely use the drive numbers, but refer to my disks with their names. Together with the ability to refer to the system disk with an asterisk and a prefixed disk with a dollar sign, you actually end up typing less than you do with the DSK# system.

It's true that you first have to format a disk and then Zero its directory before you can use it with the p-system, but you don't need an ordinary disk manager to accomplish that. The DFORMAT utility takes care of formatting a disk. Unfortunately, the original DFORMAT program, supplied on the Utilities disk, can't handle double sided disks, although it claims it can. I've developed an alternative, which handles every disk format supported by TI and CorComp controllers, including utilizing the CorComp variable interlacing feature. That means creating disks that are speed optimized for Pascal.

NEXT PAGE

The V(olume command displays all units currently available to the p-system. Unit 1, CONSOLE, is the keyboard and computer screen. Number 2 is SYSTERM (SYStem TERMinal), which is the same thing as CONSOLE, except that there is no implied echo on the screen of characters typed on the keyboard. Good if you want to read a key without displaying the character, since there is no equivalence to the CALL KEY statement in Pascal. Better still is to use an assembly routine that looks into the type ahead queue, to see if there is a new character stored there.

Units 7, REMIN (REMote INput), and 8, REMOUT (REMote OUTput), refer to the same physical port. Usually this is a serial port. On the TI, that's RS232/1 or RS232/2, dependent on where you have your printer connected. Setting the name of the physical port used for these units, as well as for unit 6, PRINTER, is done by the MODRS232 program on the Utilities disk. The source code of this program is included, together with the suggestion (in the manual) to use this procedure when you want to change the configuration within your own program. The REMOTE unit is mainly intended for communication purposes other than printing.

Unit 14, OS, contains the procedures that actually are the operating system. Most of these reside in a file called SYSTEM.PASCAL. From a hardware point of view, this is the GROM chips on the p-code card. The ROM chips contains the PME (P Machine Emulator, that interprets p-code) and low level I/O routines.

Pascal programs

Charles mentioned in his article that the p-system editor is an 80 column editor, using windowing left/right. That's true, but it's also true for the entire p-system. All programs executing under the p-system, has access to an 80 column screen.

It's a common misunderstanding that the eventual structure of a Pascal program should depend upon the structure of the language itself. That's absolutely untrue. It's perfectly possible to misuse Pascal to such an extent, that the resulting code couldn't be understood even by the original creator. The structure of a program is always a result of a structured programmer, not any particular language. However, Pascal delivers the tools a structured programmer may need to accomplish his task, of writing understandable code. The main benefit of Pascal is not the indentations that are allowed (but not compulsory), but the data structure concept. Unlike traditional BASIC, which knows nothing more complex than the array, Pascal allows a programmer to declare his own data structures, containing any mixture of data types required to handle a specific problem.

In conjunction with the separately compiled unit concept, designed mainly to facilitate memory management, it's also possible to declare a package of procedures and data structures. The benefit of this construction is, that these procedures work only with their intended data structures. That reduces the risk of hard to find errors, resulting from the correct procedure applied to wrong data. Apart from the fact that the unit is static (new copies can't be created during execution), this is similair to the class concept, found in many more modern and object oriented languages.

Charles states that he has found the p-code card as useful as the Thermal Printer, which I understand wasn't used too much. I fully agree with him in that there isn't much ready to run software available. Still I think it's the p-code card that has motivated the existence of my 99/4A. But that's only because of its overall performance as a development system for complex software. The p-system gives quite a lot of useful things, like a library system for commonly used procedures, memory management with automatic roll in and roll out of code segments, easy assembly language interface, floating point capability when needed and integers elsewhere and a lot of technical information about how it works. It also gives compatibility with , if not portability between, Pascal compilers designed for other computer systems.

Getting technical

The p-code card is located at CRU address 1F00. The reason for this is simply the fact that the p-system never releases control, once it's got it. In order to allow all other cards to execute their power up routines, the p-code card was best placed last in the search chain. What about the CorComp controller, then?

Well, since the disk controller is allocated CRU address 1100, another solution was used with the CorComp controller. That card takes command, but also takes responsibility for executing the power up routines of all other cards. The CorComp card assumes these cards to behave nicely, i.e. release control to the caller upon completition of their tasks. Usually, the only drawback with this scheme is that power up routines in GROMs are never found. These routines are rare, but do exist, for example in the Terminal Emulator II module.

When confronted with the p-code card, the usual takeover by the CorComp disk controller is prevented. The p-code card never returns from its power up routine. This explains how the p-code card appears to take control before the CorComp disk controller.

My solution to the somewhat egocentric behaviour of the CorComp controller is new EPROMs. These were once available on the market. Together with some other minor modifications, they put an end to the idea of taking control of the 99/4A before anything else. Definitely recommended.

If anyone are interested in commenting this article, you are welcomed. Either in Bits, Bytes & Pixels, or directly to me:

Anders Persson
Drottninggatan 35
S-341 36 LJUNGBY
Sweden

DONE

Assembler Executing #4

By Bob Carmany

I'm not sure whether it was out of friendship or revenge but Ron Kleinschafer sent along a great heap of A/L source code for me to examine. Not just a few files but an entire disk full of the stuff! Commented source code --this ought to be easy enough to understand ---WRONG!!! After a couple of hours (with my trusty elementary text in hand) a few rays of light penetrated the darkness! This was great stuff and I was even starting to understand a bit of what he did! It did make my head ache, though! Anyway, I learned a couple of things from reading all of that code --- It takes a heap of code to construct a meaningful program and there are several ways to get the same job done.

As an example, let's take a look at something simple --a loop. That is an easy and painless start! For the purposes of this illustration, we will assume that the guts of the loop are coded identically and only a few selected lines will change. You will see what I mean when we get into the program segment.

Let's start with the first one:

```
        LI    R6,4              Load the loop limit (4) into register 6
        :      :
LOOP    :
      loop contents
      and instructions
        :      :
        :      :
        AI    R6,-1    Add -1 to the loop counter in R6 after each pass

        JNE   LOOP     Compare the result of the arithmetic function to
                       zero and jump back to rerun LOOP if not equal to
                       zero
```

Easy enough so far! But, being a sneaky and devious language, there is another simple way to do the very same thing.

```
        LI    R6,4              Load loop limit (4) into register 6
        :      :
LOOP    :      :
      loop contents
      and instructions
        :      :
        :      :
        DEC   R6                Decrement R6 by one

        JNE   LOOP              Same as the first example
```

These are just two examples of how to code the very same idea in different ways. In the examples we used A(dd)I(mmediate) and DEC(rement) to effectively reduce the value in R6 by one in each iteration of the loop until a value of zero was reached at which time control was passed to another program segment.

Of course, the converse is true as well. By using a positive value after AI, we can create an incrementing loop. The AI instruction can be replaced just as easily by INC(rement). So we have created to instructions that perform the same function.

NEXT PAGE

```
        AI      R6,1
```

and

```
        INC     R6
```

An incrementing loop takes a bit more "engineering" to switch control when the loop limit has been reached but that is a minor inconvenience. In fact, even with my feeble experience in A/L programming I can think of four or five other ways to construct a basic loop --either decrementing or incrementing it as the case may be.

I guess that we are just about done with the preliminaries. We have covered addressing modes, number conversions, and figured out that there is no "right way" to write an A/L program --just different ways. Now we can start a bit of poking about!

Before we get started, there are a few definitions that we have to become familiar with so we know what we are doing (?).

Bit - a single binary digit

Niblet - two bits

Nibble - Four bits

Byte - eight bits

Word - two bytes (16 bits)

Ok, since I've hacked about a bit in A/L I have discovered that there are some instructions that are used a lot more than others. The whole lot of them fits into a series of categories. It's time to look at one of them. These are the instructions that move data about in a program

Name	OpCode	Comments
Move Word	MOV	This instruction can use any of the 5 general addressing modes
Move Byte	MOVB	Same as above
Swap Bytes	SWPB	Same as above
Load Immediate	LI	Register direct mode using immediate addressing
Load Wkspace poi	LWPI	Immediate addressing
Load Int Mask Im	LIMI	Immediate addressing
Store Wkspace P	STWP	Register direct mode
Store Status	STST	Register direct mode
Shift Rt Logical	SRL	Register direct mode with a count value
Shift Rt Arith	SRA	Same as above

NEXT PAGE

Shift Rt Circ. SRC Same as above

Shift Left Arith SLA Same as above

The most commonly used instructions are the first five and the last four in the list. Even amongst that group, there are a couple that do most of the work and we will take a look at them now.

MOV Source operand, Destination Operand

example:

MOV R1,R6 moves (or copies) the contents of R1 into R6

MOVB does the same thing except that it moves just 8 bits instead of 16 like MOV does.

LI Register, Operand

example:

LI R1,10 loads 10 into R1

LWPI functions the same except that the operand is loaded into a workspace.

SWPB Register

example:

SWPB R7
Swaps or switches the right and left bytes of the word in R7.

All of this has made me a bit thirsty. It's time to satisfy the inner man. I hope that you will see where all of this is headed as I try to cope with the rest of this A/L stuff that Tony and Ron got me started on. Oh well, jot down some notes and I'll continue this next month and hopefully it will lead somewhere besides insanity!

DONE

ASSEMBLER EXECUTING #3

By Bob Carmany

Have you ever heard of the "Riddle of the Sphinx"? It goes something like this: "What walks on four legs in the morning, two legs at mid-day, and three legs in the evening"? A real puzzler, huh? (answer later). Why did I bring that up? It's about the way I feel about this next topic -- addressing modes for registers which is where we will start. Let's have at it!

The first of the five general addressing modes is Register Direct Addressing. It is very simple (probably why I only had to read it once to understand it!). It simply takes the contents of one register and copies it to a second register. After the operation, both registers are identical. It looks like this:

```
MOV R10,R1
```

If R10 contained the value 3, both would contain 3 after the operation. No worries about this one!

WARNING: THIS NEXT SECTION SHOULD BE READ ONLY WHILE SOBER!!!

The second of the five general addressing modes is Register Indirect Addressing. Now things start to get complicated! This copies values from one register to the LOCATION contained in the second register. For example, if R1 (our example) contained a memory location, Indirect Addressing would copy our value 3 to the address contained in R1 instead of to R1 itself. I'm not sure I still understand this one fully! Anyway, it looks like this:

```
MOV R10,*R1
```

I think of it as moving a value to the location POINTED TO by the second register.

The third of the general addressing modes is Register Indirect Autoincrement Addressing. Here's where things start to ge bloody unbearable! For this example, we will use some hex values (more fun). R10 contains the hex value A062 and R3 contains the hex value B37E when we start this example. Ok, this operation moves the value A062 (from R10) to memory location B37E (pointed to by R1) and then increments R1 by two. So, when the operation is complete, R1 points to B380. The format looks like this:

```
MOV R10,*R1+
```

This is one I have problems with! In fact, it is my justification for keeping a book on elementary A/L programming next to the computer. Have Ron or Tony explain it to you at the next meeting.

Whew! Time for an easy one. The fourth general addressing mode is Symbolic Addressing. It can be used to move a value to a location in general memory that isn't in a register (ie. a memory location). It can also be used to move the value to a memory location specified by name. For example:

```
MOV R10,@>837C
```

moves the contents of R10 to the memory location >837C. Or:

```
MOV R10,@SCRN
```

moves the contents of R10 to the memory location named SCRN.

The last of the general addressing modes is Indexed Addressing. This is a combination of Symbolic Addressing and an address in a register. Let's take a look at this one! Suppose R10 contains our hex value A062 and R1 contains the value 0004 and SCRN contains the hex value B37E. The operation:

```
MOV @SCRN(R1),R10
```

Moves the value contained in R10 to address B382 (the memory location in SCRN - B37E plus 4 -- the value in R1).

I'm beginning to get the idea that I was in fact "sold a blind horse" by those two --Ron K and Tony McG. All of this has made my head hurt and bsides that, I'm thirsty! I'll be back in a few minutes to continue this (I hope).

Immediate Addressing is the next one. It is almost as easy as the first of the general addressing modes that we discussed. There are two instructions to use with it -- LI (LoadImmediate) and AI -- (AddImmediate). They do exactly that! Depending on which one you use, they either load or add a value to a register. For example:

```
LI R10,5
```

loads the value 5 into R10. And:

```
AI R10,4
```

Adds 4 to the current value of R10. The biggest advantage that they have is a savings in space when assembled.

The last of the addressing modes that we will attempt for the present is the PC (Program Counter)-relative Addressing mode. The discussion on this one is going to be brief --- find an elementary A/L book and read about jump instructions. There are 13 of them and the easiest thing to do is read about them at your own pace. I would suggest "Learning TI-99/4A Home Computer Assembly Language Programming" by Ira McComic. It's the one I use and where I got most of my examples from.

Now, for my very first attempt at an A/L program. This program runs from the E/A cartridge because I'm just not up

NEXT PAGE

to writing my own standalone routines for some of this stuff. It is much easier to use the REF table to access the routines, anyway. This program serves no utilitarian purpose. It just displays bits of text on the screen.

Oh yes, I only got 7 errors the first time I assembled this lot from the source code. Most of them were typos but I got some colorful screen displays at the start when I used the wrong register in the SCRWRT routine for displaying the text. I got the inspiration for this by examining bits of commented source code that came my way from Ron K (and his cohorts) in their undying effort to coerce me into trying A/L in the first place. Anyway, much to my surprise, this stupid program actually worked after I corrected the typos and the errant register access. It isn't the most elegant program but maybe Tony started this way once . . .naw, not a chance!

The answer to the riddle: a man. Crawls on all fours as a baby, walks on two legs in middle age, and with a cane (three legs) when aged. 'Til next month!

DONE


```
**********************************
*   BITS, BYTES & PIXELS         *
*   Published by Lima OH         *
*     99/4A User Group           *
*                                *
* Material contained herein      *
* may be copied by any user      *
* group as long as credit        *
* is given.  DV80 files of       *
* most articles in BB&P can      *
* be obtained by sending a       *
* disk and return postage.       *
*                                *
* ADDRESS-  P.O. Box 647         *
*           Venedocia Ohio       *
*           45894                *
* Published monthly except       *
*      July and August           *
* ------------------------       *
*      GROUP OFFICERS            *
* President-Susan Cummings       *
*               419-295-4239     *
* Vice Pres-Peter Harklau        *
*               419-234-8392     *
* Treasurer-Leonard Cummings     *
*               419-738-3770     *
* Newsletter editor and          *
*   Librarian-Charles Good       *
*               419-667-3131     *
**********************************
```


ARE NO TWO CONSOLES ALIKE?

By: Andy Frueh, Lima UG

These are some things I have observed concerning the TI consoles. I know it seems odd, but I have had experience with 3 black and silver consoles and 2 beige, and none of them are 100% identical to another one. Strange.

I suppose the thing that is the most different are the keyboards. Of the five consoles, I have noticed 3 distinct styles of keyboard. This is excluding color. I'm talking about "feel". The first console I had was fairly quiet (black) and mushy. The next black keyboard was VERY noisy and clanky. The beige console I am using now is very quiet and slightly firm. I have seen one beige keyboard that compares to the second clanky black model.

There are other little differences, too. For example, I sent my console (black & silver) to TI to get repaired. I was using one of the club's gray consoles. The on/off indicator was orange (a little orange strip shows up by the switch to show it is on) and the side shielding was silver. It is gold on all black models. Some peripherals will not work with silver shielding. However the console I got in exchange from TI was beige and had a blue on/off indicator and gold shielding.

Definately one of TI's problems, albeit a small one, is lack of consistency!

DONE