# Converting PicoWeb Projects to the New Development System

Version 2.00

July 22, 2000

Lightner Engineering
8551 La Jolla Shores Drive
La Jolla, California 92037-3044
Phone: +1-858-551-4011 FAX: +1-858-551-0777
Email: support@lightner.net
URL: http://www.picoweb.net/

 

## Table of Contents



 

## Introduction

This document describes the steps needed to convert an older PicoWeb project designed to use Atmel's AVR assembler to use the new PicoWeb development system which uses the GNU assembler and linker. The new v2.00 GNU-based PicoWeb development system was released on July 6, 2000.

Every attempt was made to maintain upward compatibility with the older PicoWeb development system software and firmware. However, because of certain basic syntax differences between the Atmel's MS-DOS assembler (i.e., `AVRASM.EXE`) and the GNU assembler, changes to a user's assembly language code may be required in order to use the new development tools. Also, because the new PicoWeb development system makes use of separately assembled object modules, and because a given PicoWeb project file is not longer a *single* assembly language source file, changes to project file #defines and directives may be required. Changes to PicoWeb *tags* and/or URL references in PicoWeb HTML files may also be required. Certain PicoWeb tags have been made obsolete as part of the switch to the new development system. Finally, changes to URLs which reference PicoWeb Web pages may be needed because of the fact that new PicoWeb kernel firmware now supports true file names in the PicoWeb's internal Web server "file system".

Each of these issues will be discussed in the sections which follow. Some changes are mandated while others are optional. The optional changes are usually due to the fact that the PicoWeb development system makes certain changes automatically when the project files are processed.

## Changes for GNU Assembler

The PicoWeb development system has switched from using Atmel's 16-bit MS-DOS assembler (i.e., `AVRASM.EXE`) to a public domain version of the GNU assembler which targets Atmel AVR machine code (i.e., `avr-as`). There were a number of factors that mandated this change. Here are a few of the advantages of the GNU-based assembler:

- Public domain source code (GPL license)
- No 16-bit MS-DOS file name restrictions (i.e., 8+3 names were required, including all directories in a source file's full path)
- No (known) interaction with McAfee virus protection software
- Support for object module linker (i.e., `avr-ld`)
- Better control over code and data placement (through the use of `.section` directives)
- Support for local labels through separately assembled modules (i.e., not all labels automatically "global")
- Better assembler macro support
- Supported under multiple operating systems, including Linux
- Support for C subroutines (using gcc)

The following is a list of items that can force changes to existing PicoWeb assembly code:

- The syntax of `.byte` and `.word` directives is very different (`AVRAVM.EXE` vs. `avr-as`)
- `.global` directives needed to reference labels in other object modules
- Program memory is now "byte addressable" (i.e., need `PM()` macro in certain cases)
- Certain complex relocatable expressions cause errors (related to required link step)
- Assembler macro declaration syntax is different
- Pcode routines must be listed in global table (i.e., `pcode.def`)
- New code/data section directives replace `.cseg`, `.dseg`, and `.eseg`

**GNU Assembler Documentation**
An HTML document title "Using as, The GNU Assembler" (January 1994) can be found in the PicoWeb development system's documents directory (`docs`) in a file named `as.html`. Please consult this file for a complete description of GNU assembler syntax. Note that all of your old *assembler macro* definitions will need to be changed to the new assembler syntax.

 

**Automatic Preprocessing**
Because a Perl-based preprocessor is used to prepare PicoWeb pcode instructions for assembly, this preprocessor can automatically map (i.e., "fix") certain older Atmel assembler directives before the assembly language source is sent to the GNU assembler. These automatic preprocessor mappings are listed in the following table:

| Assembler Directive | Preprocessor Mapping |
|---|---|
| `.device` | (line is deleted) |
| `.listmac` | (line is deleted) |
| `.dseg` | `.data` |
| `.cseg` | `.text` |
| `.eseg` | `.section cgi_seeprom`<br>(in a project file only, otherwise an error) |
| `eseg "string"` | `.section seeprom_cgi_strings` |
| `.dw` | `.word` |
| `.db` | `.byte` |
| `.equ X=y` | `.equ X,y`<br>`.equ x,y`<br>`.equ X,Y`<br>(Note: maps to both upper and lower case) |
| `.byte` *n* | not changed! |
| `.word` *n* | not changed! |

In general, if all of a project's code is located in a single project file or in source files included in that file using #include directives, then it is likely that no changes will be required to the project's assembly and/or pcode source. The only exceptions will be code that uses "`.byte`" or "`.word`" directives and code that makes use of a "`.eseg`" declaration to allocate storage in on-chip EEPROM memory.

**Global Labels**
If you project needs to make use of separately assembled object modules, then you will need to add .global assembler directives for any label that needs to be accessed by another object module. The following is an example:

```
.global mylabel
mylabel:
```

**Program Memory Addressed as Bytes**
Program memory is now treated as "byte addressable" by the assembler. Under the older Atmel assembler any reference to a memory location in flash-based program memory was treated as a "word address". For example, this meant that all strings in program memory needed to be aligned on a 16-bit word boundary. This is not longer the case with the GNU assembler. Note that AVR instructions still need to be aligned on a 16-bit word boundary.

Certain AVR instructions still need a "word address" (i.e., `icall`). In situations where a "word address" is needed the predefined assembler macro `PM()` can be used to convert a reference to a program memory "byte address" into a proper "word address".

**Pcode Declarations**
Pcode instructions which are defined in separately assembled object modules must be listed in the global pcode definition file (i.e., `pcode.def`). The PicoWeb preprocessor makes use of this file to identify legal pcode instructions when processing assembly source code files. A local copy of this file can be placed in the project directory. In this case, the pcode instructions listed in this file will be merged with (and override) the definitions in the file of the same name in the library directory (`lib`). Note that pcode instructions defined in the *project file*, and referenced exclusively in the project file, do not need to be listed elsewhere.

 

**Memory Management**
The hardware of the PicoWeb server contains two integrated circuits that have volatile and non-volatile storage which needs to be configured before the PicoWeb server can be used. One is the Atmel 90S8515 microcontroller and the other is an $I^2C$ serial EEPROM chip. The Atmel microcontroller has three kinds of memory, static RAM (512 bytes), flash-based program memory (8 Kbytes), and on-chip EEPROM (512 bytes). The $I^2C$ serial EEPROM chip holds between 16 and 32 Kbytes of data, depending upon the PicoWeb version. The configuration of the "initialized" portion on the static RAM, the two EEPROM memory blocks (on-chip and external) and the microcontroller's flash-based program storage is controlled by assembler *section* directives.

The various memory *sections* include the SRAM (e.g., `.bss` and `.data` sections), flash-based program memory (e.g., `.text` and `.cseg` sections) and on-chip EEPROM (e.g., `.section eeprom` section) in the Atmel microcontroller, as well as the external serial EEPROM chip (e.g., `.eseg` and `.section seeprom` sections). In fact, the PicoWeb development environment makes use of many other GNU linker *sections* to control the placement of code and data in the four basic memory blocks present in the PicoWeb server's hardware. The naming and order of these many linker sections is critical. The addition of new sections and/or modifications to the master linker control script (`picoweb.ld`) in not recommended without a thorough understanding of the issues involved.

The following table lists the basic assembler directives that are used to control the placement of code and data in the PicoWeb server many memory *sections*:

| **On-Chip Flash Program Memory (8 Kbytes)** | |
|---|---|
| **Assembler Directive** | **Meaning** |
| `.text` | AVR program code and data |
| `.section reset` | PicoWeb `#avr_reset` section |
| `.section fast` | PicoWeb `#avr_fast` section |
| `.section slow` | PicoWeb `#avr_slow` section |
| `.section strings` | `.ascii` and `.ascz` strings in program memory |
| **On-Chip SRAM (512 bytes)** | |
| **Assembler Directive** | **Meaning** |
| `.data` | initialized data |
| `.section .bss` | uninitialized data (zeroed at reset) |
| `.section COMMON` | named common blocks (alignment penalty!) |
| `.section uninitialized_data` | uninitialized data (not cleared at reset) |
| `.dseg` | mapped to "`.data`" by preprocessor |
| **On-Chip EEPROM (512 bytes)** | |
| **Assembler Directive** | **Meaning** |
| `.section eeprom*` | initialized data in EEPROM |
| `.section .eeprom*` | initialized data in EEPROM |
| **External Serial EEPROM (16-32 Kbytes)** | |
| **Assembler Directive** | **Meaning** |
| `.section seeprom` | pcode and data in serial EEPROM |
| `.section seeprom_strings` | strings in serial EEPROM |
| `.section seeprom_cgi` | pcode and data in serial EEPROM (linked high) |
| `.section seeprom_cgi_strings` | strings in serial EEPROM (linked high) |
| `.eseg` | mapped to "`.section seeprom_cgi`" in project file (error elsewhere) |

**Interrupt Vector Table**
Any code that makes use of the Atmel microcontrollers interrupt vector table will need to be changed. Under the new system, each entry in the Atmel microcontroller's interrupt vector table, located at location 0 in program memory, points to a dummy routine that consists of a single "`ret`" instruction. This is done using the assembler's "`.weak`" global directive.

The following shows the PicoWeb kernel's interrupt vector table at location 0:

```
.weak VEC_reset
.weak VEC_int0_isr
.weak VEC_int1_isr
.weak VEC_timer1cap_isr
.weak VEC_timer1compa_isr
.weak VEC_timer1compb_isr
.weak VEC_timer1ovf_isr
.weak VEC_timer0ovf_isr
.weak VEC_spistc_isr
.weak VEC_rx_isr
.weak VEC_udre_isr
.weak VEC_tx_isr
.weak VEC_ana_comp_isr
```

Any user-provided interrupt service routines need to override these table entries.

Here is a sample code fragment from the serial port's receive interrupt service routine (see firmware source code in the file "`lib/serial.asm`"):

```
.global VEC_rx_isr
VEC_rx_isr:
    push    r18
    in      r18,sreg
    push r18
    .
    .
    .
```

The assembler directive "`.global VEC_rx_isr`" overrides the default "weak" dummy interrupt service routine for the serial port's receive interrupt with a new user-supplied handler (i.e., label "VEC_rx_isr").

**Complex Relocatable Expressions**
Because a linker (i.e., `avr-ld`) is used to assemble multiple PicoWeb object modules into a single executable file, certain complex expressions involving relocatable variables (e.g., global labels) are no longer allowed. Please see assembler and/or linker documentation for details.

 

**Table 1 – PicoWeb Project Preprocessor Defines**

| Preprocessor Define | Default Value | Description |
|---|---|---|
| BANNER | "\r\nPicoWeb Server\r\n" | Text string that is printed to the serial port each time the PicoWeb server is reset |
| BAUD_RATE | 19200 | Defines the serial port baud rate. Note that #define CLOCK must be correctly set. Not all baud rates are available at all processor clock rates. *(Consult Atmel's AT90S8515 datasheet for details.)* |
| CLOCK | 7372000 | Defines the clock rate of Atmel microcontroller (Hz). This is used to set the UART baud rate divisor and for other timing-related calculations. |
| DEBUGGER | not defined | If defined, includes the serial port debugger firmware as part of the project build process |
| EEPROM_IP | not defined | If defined, specifies that the PicoWeb's IP address is stored in on-chip EEPROM (address will come from text file `ip`) |
| ENABLE_WATCHDOG | not defined | Enables the watchdog timer hardware on reset if defined |
| `NET_CONFIG_IP` | not defined | Define this if to allow the PicoWeb's IP address to be changed over the Ethernet using the program `setip` |
| NO_ETHER_ADDR_ON_BOOT | not defined | Define this to inhibit printout of the PicoWeb's Ethernet address to the serial port on reset |
| PKT_WATCHDOG_MAX | 250 | Idle network packet watchdog timer limit in "slow idle loop" trip counts (recommended range: 100-254) |
| SEEPROM_IP | not defined | Defines address in serial EEPROM which holds the PicoWeb's IP address. Define if you want the IP address stored in serial EEPROM memory |
| STATIC_IP_LSB<br>STATIC_IP_MSB | not defined | Define these two 16-bit constants if you want the PicoWeb's IP address to be "hard coded" into program memory (defines high and low part of 32-bit IP address) |
| USE_BOOTP | not defined | Use BOOTP protocol to get PicoWeb IP address |

## Changes to PicoWeb Project File

Besides the assembly language code changes detailed in the previous section about the GNU assembler, you should not be required to make any changes to the first section of your PicoWeb project file, namely everything up to the first `#avr` directive. This is the section of the project file where project-wide C-preprocessor defines are listed, along with the project's Web pages and CGI routines.

Nevertheless, you still may want to makes changes to your older project files in order to bring them up to date. Here is a list of things that are different and/or improved with respect to this first section of the project file:

- New #defines for processor clock rate and serial port baud rate
- Action of certain #define statements is different (i.e., some have no effect now)
- Web page file system uses "real names" (i.e., "`ii00`" and "`iu00`" URLs are gone)
- New options to set Web pages' *content type*
- The listing of CGI routines used in HTML *tags* is now optional
- Support added for linker directives (i.e., controls behavior of linker `avr-ld`)

**New Project File #defines**
Table 1 lists the PicoWeb preprocessor #define statements which affect the PicoWeb project build process. Note the addition of the three new preprocessor defines:

```
#define NET_CONFIG_IP     /* allow IP address reconfiguration via net */
#define CLOCK 7372000     /* clock rate of microcontroller (in Hz) */
#define BAUD_RATE 19200   /* serial port baud rate */
```

The first definition allows changes to a PicoWeb's IP address from a remote host computer connected to the PicoWeb via the Ethernet. The other two new definitions (i.e., `CLOCK` and `BAUD_RATE`) automatically calculate the proper value needed to get the serial port UART baud rate divisor, previously set using `#define SERIAL_BAUD_DIVISOR`. You will need to remove this older definition from your project file if you use the new definitions.

**Effect of Project File #defines**
It is important to note that the preprocessor definitions listed in the PicoWeb project file have a very different effect under the new development system. Under the old system a given PicoWeb project was assembled as a single assembly language source module (with many separate source files included using `#include` directives). Under the new development system, the only files these definitions affect are: assembly language source code in the project file, those assembly source files that the project file explicitly includes (i.e., with `#include`), the library file `main.asm`, and the linker script `picoweb.ld`.

**Web Page Content Type**
There is now more control over the content type delivered with a PicoWeb file. Note that this information is no longer specified in the HTML file, for example, by using the special PicoWeb tag "`t". As with the older development system software, the first section of a PicoWeb project file lists one or more files which contain HTML code and/or images which will be stored in the PicoWeb's serial EEPROM memory as part of the Web server's "file system". The first file in the list is the default Web server "home page". Any files listed must have one of the following file extensions:

`.htm` – HTML code (text file)
`.html` – HTML code (text file)
`.txt` – ASCII text file
`.jpg` – JPEG image
`.gif` – GIF image
`.png` – PNG image
`.js` – Javascript (text file)
`.cla` – Java applet (byte-codes)
`.class` – Java applet (byte-codes)

or they must also be followed by a HTTP *content type* field specification. (This is a new feature.) The following is a list of common "content type" field values:

```
none
text/html
text/plain
image/jpeg
image/gif
image/png
application/octet-stream
```

 

If no file content type specification is given, then an appropriate content type will be assigned according to the file's extension. Whenever a file is supplied by the PicoWeb server in response to a Web browser's GET request, the following HTTP header information will proceed the delivery of the file's contents:

```
HTTP/1.0 200¿
Content-type: content type¿
¿
```

where ¿ indicates a new-line character. Those familiar with low-level CGI programming will recognize the significance of the HTTP header, which tells a Web browser exactly what kind of document is being returned from the Web server in response to an HTTP GET request.

Any PicoWeb-supplied file may be assigned an arbitrary HTTP *content type* by providing a content type field after the file's name in the project file. The *content type* fields are not checked and can be any arbitrary string of characters.

Note that the special *content type* field "`none`" specifies that the PicoWeb server should not proceed the file delivered in response to an HTTP GET request with anything. In this case, the file itself (or the user-supplied CGI code that executes) *must* supply the needed HTTP header. Note that certain Web browsers (e.g., Netscape) will display without error retrieved Web pages which are missing the HTTP header. However, this is not always the case!

**True File Names in URLs**
Note that the Web pages listed in the project file are now *only* accessible by their full file name. The old naming scheme where the first Web file was referenced using "`ii00`" no longer works! As before, if no file name is given, the first Web page in the file list is returned, namely the PicoWeb server's "home page". However, if a PicoWeb URL file name *is* specified, and it does not match any of the listed files, then an error indication is returned. Under the older development system's firmware, the PicoWeb's "home page" would be returned, with no errors.

**Listing of CGI Routines Optional**
The listing of CGI routines in the first section of the PicoWeb project file is now optional. It used to be the case that all user-supplied CGI routines referenced by special PicoWeb tags needed to be listed there. This section now lists zero or more public *pcode entry point labels* for pcode routines. The extension "`.cgi`" must be appended to each pcode entry point label listed in this section.

The following is a sample project Web Page CGI Routine section:

```
//
// application-specific CGI routines
//
temperature.cgi // iu00 (returns ASCII temperature reading (deg-f)
```

The above line indicates that a CGI pcode routine named "`temperature`" exists somewhere in the PicoWeb server and that this routine can be accessed externally with an HTTP GET request using the following URL:

```
http://x.y.z.w/temperature.cgi
```

where `x.y.z.w` is the IP address assigned to the PicoWeb server.

Also, wherever the string `` `temperature.cgi` `` is found in any of the HTML source code files listed in the Web Page File List section as part of the project, a special *tag* will be inserted into those files. This special tag will cause the pcode routine "`temperature`" to be called at that point in the HTML code stream whenever an HTML file containing that tag is retrieved. Note that this tag processing applies to any global pcode subroutine label, not just those routines listed in this section.

 

**Linker Directives**
Support has been added for linker directives which control the behavior of the linker (`avr-ld`) The first section of the PicoWeb project file may include optional directives to the PicoWeb development system's linker. Allowed linker directives include:

```
link add object
link search library
```

where *object* is the name of a PicoWeb object file and *library* is the name of a PicoWeb library archive. The "link add" directive causes the linker to unconditionally include the specified object file. The "link search" directive causes the linker to search the specified archive file to resolve any pending unresolved global symbol references.

The following is a sample project Linker Directive section:

```
// linker directives (optional)
link add myobject.o
link search mylibrary.a
```

The "link add" directive will cause the linker to unconditionally include the object file "`myobject.o`" as part of the PicoWeb firmware image. The "link search" directive will cause the linker to search the library file "`mylibrary.a`" in order to resolve any global symbol references.

**Variable `putcok` Not Needed**
The SRAM variable "putcok" is no longer needed. This is now a dummy variable. Before, this variable had to be non-zero to enable output to the serial port and/or the network. User CGI routines no longer need pcode such as that shown in the following example:

```
my_cgi_routine:
    ppushn putcok,1                 ; save putchar enable state
    pmovbi putcok,0xff              ; putchar is now OK
    .
    .
    .
    ppopn putcok,1                  ; restore putchar enable state
    pret
```

## Changes to PicoWeb HTML Files

Depending upon exactly how you wrote the HTML code for your older PicoWeb Web pages, you may need to make changes to those older Web page to use the new development system. The biggest change to the PicoWeb development system's Web page file system is largely hidden. Under the new scheme, Web pages are now simple pcode routines. In the simplest case, a Web page may be a single pcode instruction that prints a string with the entire contents of the Web page. This is the case with Web pages that contain JPEG and GIF images. The other major change to the PicoWeb firmware was support for true file names in its "file system".

**Tag `` `t `` Obsolete**
The special PicoWeb tag "`` `t ``" is now obsolete, as mentioned above. However, the new PicoWeb development system will turn this tag into a "no operation" if it appears as the very first thing in a PicoWeb HTML file. We recommend that you remove these tags from your HTML files.

 

**Table 2 - PicoWeb HTML Tags**

| Tag | Meaning |
|---|---|
| `` `*routine*.cgi` `` | Call pcode routine "*routine*". |
| `` `*routine*.cgi?*parm*` `` | Set global parameter psetparm_parm to "*param*", then call pcode routine "*routine*". |
| `` `?*routine*.cgi?*parm*` *texteq* { *textneq* } `` | Set global parameter psetparm_parm to "*param*", then conditionally call pcode routine "*routine*". Conditionally output text according to the state of the pcode "Z flag". If Z=1, output text "*texteq*"; if Z=0, output text "*textneq*". |
| `` `?*routine*.cgi?*parm@label*.cgi` `` | Set global parameter psetparm_parm to "*param*", then conditionally jump to the label named "*label*" according to the state of the pcode "Z flag". If Z=1 then the jump is taken. |
| `` `=*label*` `` | Place a label named "*label*" in the pcode which outputs the HTML code which follows. May be "called" from other HTML code using `` `*label*.cgi` ``. |
| `` `@*label*.cgi` `` | Unconditional jump to pcode "*label*". Normally this is a location in an HTML code file labeled using the tag `` `*label*.cgi` ``. |
| `` `t `` | Removed if first item in a PicoWeb HTML file (compatibility) |

Notes:

1. The affect of the "Z flag" can be reversed in the conditional call/jump tags (i.e., `` `?... ``) by following the leading "?" with a "!" (i.e., `` `?!*routine*.cgi?*parm@label*.cgi` ``).
2. The value "*param*" shown in the table can take any of the following forms:
   - 0x*hhhh* A 16-bit hexadecimal value "*hhhh*" (e.g., 0x8000)
   - *ddddd* A 16bit decimal value "*ddddd*" (e.g., 32767)
   - "*string*" A text string delimited by double-quotes (e.g., "hello")

**Changes to PicoWeb Tags**

The format of the special PicoWeb HTML *tags* has been changed. However, *most* of the old tags should still give the same results with the new development system firmware. But, this is *not* true for tags of the form `` `*n* `` where *n* is a number.

All new PicoWeb tags have a back-tick (`) character enclosing the tag specification (i.e., a "back-tick" at the beginning and another one at the end). We strongly recommend that you change your HTML tags to use the equivalent "new format" tags. Table 2 shows a list of the new PicoWeb tags. As before, only tags in HTML files are recognized (i.e., .htm and .html files). Note that routines listed in PicoWeb tags still need to be written in pcode. However, they do not need to be listed in the PicoWeb project file in order to be referenced as part of a tag.

**No Automatic Processing of URL Parameters**

There is no longer any automatic processing of the HTTP GET request URL parameters for a parameter of the form "n=s". The PicoWeb sample project "webled" made use of this now obsolete feature. In order to get the old behavior, you must include the following tag at the beginning of your Web page's HTML code:

```
`pchk_port_url_parms.cgi`
```

**True File Names in URLs**

As noted above, the Web pages listed in the project file are now *only* accessible by their full file name. The old naming scheme where, for example, the first Web file was referenced using an index "ii00" no longer works! This means that if your HTML code explicitly referenced a Web page or CGI routine by its "index" then you

 

will need to make changes. Any references to "`ii`*hh*" or "`iu`*hh*" (where *hh* is the file's hexadecimal index number) will need to be changed to use the true name of the target file.

**Bad URL No Longer Returns Home Page**
As with the old firmware, if no file name is given in a URL, the PicoWeb server's "home page" is returned. However, if a PicoWeb URL file name *is* specified, and that name does not match any of the listed files, then an error indication is returned. This will be an issue if you have depended upon this behavior, for example in the ACTION statement in an HTML FORM. (For example, the old version of the sample project "webled" made use of this "feature".)

## Latent Pcode Bugs

As we switched our many PicoWeb sample projects to the new development system, we uncovered a number of latent bugs relating to the misuse of the pcode parameter syntax in places where a memory address was expected (i.e., not an "immediate" value). The following syntax:

```
[byte buf]
```

needed to be changed to:

```
[buf]
```

in many places because in these cases the pcode instructions' parameter was supposed to be an SRAM address. Note that `[byte buf]` specifies an indirect memory reference to memory location (`*buf & 0xff`). The second (correct) example does not discard the upper byte of the final target address (i.e., after indirection)! The first example worked under the old development system because the target SRAM location contained in `buf` happened to be below address 0x100. This was not the case under the new development system. Here is a list of sample project pcode instructions from which we needed to remove the "`byte`" modifier:

```
pmovb SD_CHAR,[byte pana_ptr]
pmovb buf,[byte pana_ptr]
pmovb buf,[byte rcs_ptr]
pputcb [byte DIG_PTR]
pputcb [byte rcs_ptr]
```

All of the above-listed pcode instructions are *wrong*!

The most common occurrence of this latent bug was in the often copied pcode routine named `decconv` which printed out a 16-bit number stored in `buf` as ASCII text. (Note that this is better done under the new development environment using the single pcode instruction "`pprintswi [buf]`".)

## New Pcode Instructions

**Processing URL FORM Parameters**
A number of new pcode instructions have been added which provide for simple processing of parameters passed as part of the HTTP GET "URL line". These parameters are typically passed to the PicoWeb when an HTML FORM button is pressed. If you have a project that needs to access parameters of this kind, then we strongly suggest that you take a look at the PicoWeb samples and see how this can be done with the new pcode instructions. (The sample project "serdev" makes use of these new instructions.)

Here is a sample CGI pcode routine that will print the string which follows the parameter "`S=`" in a URL line:

```
print_string:
   purlparm buf,"S="     ; search for command string
    pjumpne 1f           ; skip ahead if not found
    pprinturl buf,0      ; write text after M=
1:
    pret
```

 

Here is sample code that will convert to binary, then print the value of the decimal integer value after the URL parameter “I=” in a URL line:

```
print_int:
    purlparm buf,"I="    ; search for command string
    pjumpne 1f           ; exit if not found
    purl2int buf+2,buf   ; found...convert to integer (store at buf+2)
    pprintswi [buf+2]    ; print signed integers as text
1:
    pret
```

**Pcode Stack Frames**
The new PicoWeb development system has pcode instructions for using a “stack frame”. This is useful for passing parameters to pcode routines on the hardware stack, as well as for allocation of temporary SRAM storage for use in pcode routines. Please see the document “PicoWeb Pcode” for details.

## C Routines

The GNU C-compiler gcc will work with the PicoWeb development system. The document “How to Build a PicoWeb Project” has more details. Please consult the PicoWeb sample project “sieve” for an example.

7/22/00 11:08 PM

 