Creating a Multitasking Kernel for the COP8 Microcontroller

National Semiconductor
Application Note 1099
Richard F. Zarr
March 1998

INTRODUCTION

Everyone is familiar with multi-tasking operating systems such as Microsoft™ Windows® and UNIX®. These systems have the advantages of executing several tasks at once allowing a single CPU to do the job of several. These systems however, use very fast and sophisticated computers to run optimally. When it comes to the world of microcontrollers however, it seems very unlikely that one can take advantage of the features and benefits of a multitasking system. This is not necessarily true. With a little care and understanding of the limits of microcontrollers, a full-featured multitasking kernel (MTK) can be created. This MTK can allow many separate tasks to share the microcontroller's CPU and on-board peripherals and provide the added bonus of reduced time to market and lower development cost. In this application note, we will discuss the various types of multitasking systems, the best choice for a microcontroller, and how to implement an MTK in assembly language for the new COP8SGR microcontroller — a true system on a chip.

MULTITASKING SYSTEMS

The word "Multitasking" or "Multithreading" can have several meanings, but they generally refer to performing more than one task at a time. In reality, multitasking operating systems generally are sharing the CPU among many tasks. This sharing is done so fast that it provides the illusion that the computer is actually doing more than one thing at a time. This sharing technique assumes a single processor. Some computers however, actually have many processors in parallel and the operating system assigns an individual processor to each task. In the case of the COP8, there is only the single CPU and its peripherals to share among many tasks. This requires a mechanism to "switch" which task is actually running on the CPU. There are several task-switching methods in use today, two of which are "Preemptive" and "Cooperative".

The "Preemptive" method shown in *Figure 1* allows a task, such as calculating the number pi, to execute for some predetermined time. At the end of that time, the operating system or MTK suspends that task's ownership of the computer and enables the next task in a queue. The order in the queue can often be modified to allow higher priority tasks to get service faster. The problem with preemptive operating systems is that they require a great deal of resources for switching tasks. Since the task is stopped without warning — and usually in the middle of something complex, a complete, separate memory stack and variable space must be maintained for each task that is running. This also includes saving the address of the last instruction executed, and all the register states before the task was stopped. This is required to enable the operating system to restore the previous state of the task when it's time to start it again. This typically happens about every millisecond, but can happen faster or slower depending on real-time requirements. A characteristic of this type of task switching is that every task gets an equal amount of the system's CPU.

The "Cooperative" switching method shown in *Figure 2* handles tasks differently. The cooperative operating system uses a special understanding between the program task and itself to facilitate the sharing of resources. This "understanding" is as follows. Any task has full access to the CPU resource for as long as it requires, but must return the resource within a reasonable amount of time. Each thread must run from beginning to end, keeping track of its own state and then returning to the operating system. This implies that if the task does not return the CPU resource (i.e. gets stuck in an infinite loop), the system crashes or locks up. Sound familiar? The first versions of Microsoft Windows (Windows 3.x) used this scheme. If a program locked up, only an interrupt (such as the familiar ALT-CTRL-DEL sequence) could get the operating system's attention. Unix and Windows NT however, use a fully preemptive task switching method, which is much more robust.

Which method should be used for a microcontroller? The preemptive method is robust, but carries a heavy penalty in resources. The cooperative method uses much fewer resources, but can still lock up. A simple solution is to increase the robustness of the cooperative method by adding some of the qualities of the preemptive method. We'll call this method the "Supervised Cooperative" method and this will be the basis for COP8 based MTK. By adding some "watch-dog" features to the operating system, each task thread can be monitored for its usage of the CPU resource. If a task takes too long, it can be stopped and restarted — a nice feature.

THE SUPERVISED COOPERATIVE MULTITASKING KERNEL

The kernel used in this application note will use the supervised cooperative method of task switching. Since the COP8SGR has a watch-dog timer (Timer T0) that is always running, an interrupt can be generated periodically to check the status of each task. Every 4096 cycle of the instruction cycle clock, Timer T0 can generate an interrupt. At 10 MHz, that is about every 4 milliseconds. By using this "Tick", the operating system can "watch" threads execute. If a task is stuck, or locked up in a loop due to a coding error or other failure, the operating system can recover. Here's how it works. Every time a task is switched, a counter is loaded with a user selectable value (1–255). This number is the maximum task time allowed by the operating system. A zero value disables this feature by turning off the timer T0 interrupt. To calculate the worst case time, use the following calculation:

Tmax = n * Tcycle * 4096 where **n** is the value passed to **osSetMaxTaskTime**

If a task locks up in this example operating system, the OS transfers control to the first task loaded — the MAIN task. This task should have a message-processing loop to handle operating system messages from timers, UARTs, and the operating system itself. These messages are passed back to the loop in the accumulator on entry. If there are no mes-

UNIX® is a registered trademark of A T & T Bell Laboratories.
Microsoft™ is a trademark of Microsoft Corporation.
Windows® is a registered trademark of Microsoft Corporation.

sages for the message loop, the accumulator will be null (zero). If a task locks up, the main loop will receive a **mTaskFailure** message. The application can identify the task by looking at the 'B' register in the message loop. The main task can then restart it without affecting the other running tasks. To prevent locking up again, the OS stops the failed task. This requires the application to restart it implicitly. Task 1 (the first task added) is always the main task. If the main task fails, the OS will automatically restart the microcontroller since this is a fatal failure. A sample Main Task message loop is shown below.

```
Main:
     IFEQ  A, #mNone            ; Any messages?
     JP    MainContinue         ; No, continue with other main activities
     IFEQ  A, #mTime            ; Was it a timer message
     JP    MainTimer            ; Yes, go service the timer tick.
     IFEQ  A, #mUART            ; Was it a UART message?
     JP    MainUART             ; Yes, go service the UART
     IFEQ  A, #mTaskFailure     ; Did a task lock up?
     JP    MainRestartTask      ; Yes, go fix it

MainContinue:

;    (Do other main stuff here)

     JP    MainExit             ; Done, return to OS

MainTimer:
     LD    A, B                 ; 'B' holds which timer ticked...

;    (Do stuff with timer tick)

     JP    MainExit             ; Done, return to OS

MainUART:
     LD    A, B                 ; 'B' holds UART action required...

;    (Do UART stuff)

     JP    MainExit             ; Done, return to OS

MainRestartTask:
     LD    A, B                 ; 'B' holds which timer ticked...
     IFEQ  A, #2                ; Task 2?
     JSR   StartTask2           ; Yes, Reload data for task 2 and restart it.
     IFEQ  A, #3                ; Task 3?
     JSR   StartTask3           ; Yes, Reload data for task 3 and restart it.

;    (Continue checking for other failed tasks here)

MainExit:
     RET                        ; Return to OS
```

SOME CONCEPTS

We've already discussed the idea of a task or thread, but let's review the idea again. A "Task" or "Thread" is a distinct piece of code (software) that handles a specific task. This task might be scanning a keyboard for input, updating a multiplexed display, communicating on a network with other devices, reading external or internal peripherals for data, and much more. The task structure for the supervised cooperative OS requires the code to execute completely in very few passes — that is, very little looping. Loops require time, and hog the CPU resource. These tasks look very much like subroutines in that they are called by the OS, execute a function, and return back to the OS when completed. While the task is running however, no other task can have the CPU resource.

Another concept is the "Callback". Callbacks are a method of dynamically allocating resources such as timers, UARTs or other on-board microcontroller peripherals, and providing a direct handler for the resource — circumventing the main message loop. This concept works like this. A special piece of code is written to handle a specific task. This code's address is passed to the operating system as a location to "call" when this function is needed. This feature provides the benefit of not requiring the main task message loop to process the interrupt and figure out what happened. The interrupt is simply routed directly to the service routine. Callbacks can also augment or override a default function, or provide a completely new function. Let's examine an example of a callback implementation.

A timer resource can be dynamically allocated using a callback. The application can specify an interval for a timer, and a routine to call when each time tick occurs. The routine is "Called Back" after each tick of the timer. The timer can also be used once to time a specific event without "looping" to burn time. The timer is created using **osSetTimer**, which also assigns a callback routine when the time period ends. The callback in turn stops the timer and frees the timer resource using **osKillTimer**. The callback routine can then continue its operation following the time delay. This also illustrates the idea of a “Resource Pool”. The example MTK will use this concept to manage the COP8SGR UART and TIMER resources. Feel free to modify the MTK to extend this idea for handling other resources as well.

USING THE MULTITASKING KERNEL

The COP8SGR MTK architecture is shown in *Figure 3*. The COP8SGR has 4 pages of memory, each of which is 128 bytes long for a total of 512 bytes. They are selected by writing the page number into the segment register “S”. The COP8 uses the last 16 bytes of PAGE 0 for registers, so they cannot be used for data storage. There is a single stack that is located in PAGE 0, however it is available in any page when using the **POP** and **PUSH** instructions. This is very important for this MTK in that the stack is used for passing information between the operating system and the application.

We will use page 1 for the operating system, but any page other than zero will work fine. This reserves 128 bytes for the operating system. The COP8SGR has 3 general purpose timers (T1–T3), and a watchdog timer (T0), as well as a full-featured UART. The operating system will control access to these resources to allow sharing among many threads.

The initialization routine sets up all the COP8SGR memory and internal registers after reset and then calls a user setup routine called “INITIALIZE”. This is the first routine that gets control in the users code. Here the user should setup all the tasks that are initially active (more can be created later). This is done by pushing information onto the stack and calling the **osAddTask** routine. Here’s what the code looks like to add a task called “MAIN”.

```
Initialize:
     LD     A, #LOW(Main)      ; Add to task list
     PUSH   A                  ; LSB First
     LD     A, #HIGH(Main)     ;
     PUSH   A                  ; MSB Second
     JSR    osAddTask          ; Go add it...
     X      A, hMainTask       ; Save the task handle.
```

This should be done for each of the tasks the user needs to run. Each time a task gets control, the entry point is always the same. In the above example, the entry point was called MAIN. Each task is responsible for keeping track of itself and using as little time as required to perform its function. Each task is in effect “looping” along as the operating system passes control to it. Each task also must be started. This enters the task into the switching queue to await execution. This is accomplished as follows.

```
     LD      A, hMyTask        ; Tell OS which task to start
     JSR     osStartTask       ; Start the task
```

At the end of the INITIALIZE routine, the code should start the MTK as follows.

```
     JSR osStart
```

This begins the multitasking kernel and begins the task switching. An error recovery routine should follow this jump subroutine command. If the OS has a fatal error, it will return and execute instructions following this call. This allows the user to select a method of restarting the processes in the case of a catastrophic code failure. This is equivalent to the BLUE SCREEN OF DEATH seen on many PCs.

Once the OS has begun, each task will be executed in turn. The entry point for each routine will be the values passed to the OS by the **osAddTask** call. Resource callbacks will be executed whenever that resource requires attention — such as a received character in the UART RX Buffer. The application can suspend any task by calling **osStopTask**. Again, feel free to modify the MTK code given in this application note to add features you may need.

ALLOCATING RESOURCES

To acquire a microcontroller resource such as a timer, the application code should use the operating system as opposed to using the resource directly. What’s the advantage of that? Other tasks cannot tell if a particular resource is in use, the operating system can. For the case of the on-board timers, of which the COP8SGR has 3, the MTK has a special operating system routine that requests a timer resource. Since all the general purpose timers are identical, it doesn’t matter which timer a task uses. Therefore, the operating system decides which timer a task will get. This way, if another task needed a timer for awhile, and it received timer 1, the current task requesting a timer would get timer 2. Timer allocation (as well as UART and other peripheral allocation) is done using “handles”. Handles are nothing more than a number that indicates which entry in a table belongs to a particular task. The handle allows both the task and OS to know which items belong to whom. The handle is returned by the operating system if the task received the timer (or peripheral). If not, the handle will be NULL or zero. Otherwise it will be a number greater than 0. This number must be kept in order to free the resource when a task has finished using it. For instance, to request a timer a task would do the following.

```
     LD       A, #cPeriodLo             ; Setup time period
     PUSH     A                         ; Low order first
     LD       A, #cPeriodHi             ; followed by high order
     PUSH     A
     LD       A, #LOW(MyTimerCallBack)  ; Setup call back
     PUSH     A
     LD       A, #HIGH(MyTimerCallBack)
     PUSH     A
     JSR      osSetTimer                ; Go get a timer resource
     IFEQ     A, #0                     ; Was the resource available?
     JP       GetTimerFailure            ; No, bail and try later
     X        A, hMyTimer                ; Yes, save the timer handle
     (Continue)
```

To release the timer resource after the task is finished with it, the code should call the Kill Timer routine as follows.

```
  LD   A, hMyTimer     ; Free my timer
  JSR  osKillTimer
```

This routine will return zero in the accumulator if successful and non-zero if it fails. An invalid handle here can cause chaos. It may stop and release a timer being used by some other task that's critical to the application. Always store handles locally to the task. That is, keep a special variable that only that task uses for keeping the timer handle. No global variables for handles!

The timer allocation routine for this MTK is quite simple. It only allows for creating callbacks on a periodic basis. The general purpose timers in the COP8SGR however, are much more sophisticated. They have the ability to perform pulse width modulation (PWM) and capture the time between events as well. A more sophisticated callback handler would allow a mode select a particular timer and setup the timer for PWM operation, capture mode, or a periodic time. The routines in this application note can easily be expanded to include these features. Feel free to experiment and either modify these timer resource allocation routines or create your own.

WRITING TASKS

Obviously, the user of this MTK needs to add application specific code to make it useful. The code is basically made up of a collection of tasks that the application needs to perform. These tasks, as mentioned earlier, can handle many functions such as updating a multiplexed display, or scanning a keyboard. To write these task follow these simple rules.

1. Write the task as a subroutine — a single entry and exit point. This improves maintainability as well.
2. Keep tasks specific — perform single operations per task.
3. Avoid loops — The longer a task holds a resource such as the CPU, the less efficient the OS becomes.
4. Use callbacks for timers, UARTs, and other OS supported on-board peripherals.

A sample task that illustrates these points well is multiplexing an LED display. This example will use 4 common anode 7 segment displays. The D PORT of the COP8SGR will be used to drive the multiplexed segments directly, and the lower 4 F PORT bits will be used to drive PNP transistors to supply anode current to the 7 segment displays (8 segments including the decimal point). See *Figure 4* for the connection diagram. A callback from a timer will be used to maintain a constant refresh on the display. The period only needs to be about 4 mS, which provides a total refresh time of about 16 milliseconds for 4 digits. This is fast enough that the human eye will not see any flickering of the display. The PNP transistors require the pin to go low to turn on, so the code in the following example deals with these pins as active low or inverted — notice the digit drive table entries are inverted as well.

The variables in RAM for this routine are as follows

```
bDigitCount .DSB 1 ; Provide a counter to keep track of which digit is in use
bDispBuffer .DSB 4 ; Provide a buffer for the characters
```

The timer will be assigned in the INITIALIZE routine.

```
     LD       A, #cRefreshLo              ; Setup time period to refresh the display
     PUSH     A                           ; Low order first
     LD       A, #cRefreshHi              ; followed by high order
     PUSH     A
     LD       A, #LOW(ServiceDisplay      ; Setup a call back to service the display
     PUSH     A
     LD       A, #HIGH(ServiceDisplay)
     PUSH     A
     JSR      osSetTimer                  ; Go get a timer resource for this function
     IFEQ     A, #0                       ; Was the resource available?
     JP       GetTimerFailure             ; No, bail and try later
     X        A, hRefreshTimer            ; Yes, save the timer handle
     (Continue...)                        ; Continue to initialize other stuff...
```

The service routine for refreshing the 7 segment displays looks like this.

```
ServiceDisplay:
       LD      A, PORTFD         ; Get current value of port F data
       OR      A, #X'0F          ; Turn off all digits while updating (active low)
       X       A, PORTFD         ; Write new value to port F data

       LD      A, bDigitCount    ; Get current digit (0-3)
       ADD     A, bDispBuffer
       X       A, B              ; Use 'B' to point to buffer
       LD      A, [B]            ; Get the character to display
       XOR     A, #X'FF          ; Invert for active low drive
       X       A, PORTD          ; Write new value

       LD      A, bDigitCount    ; Get digit count to turn on new digit
       ADD     A, DGTable        ; Use the count to select the digit table entry
       LAID                      ; Load the accumulator with the digit data
       AND     A, PORTFD         ; Add what’s already on the upper port pins
       X       A, PORTFD         ; Write to F PORT data

       LD      A, bDigitCount    ; Get digit count and update
       INC     A                 ; Point to next digit
       IFGT    A, #cMaxDigits-1  ; Check if past last digit
       LD      A, #0             ; Yes, Reload to beginning
       X       A, bDigitCount    ; Save back for next pass
       RET                       ; Return to OS
DGTable:
       .DB     B’11111110        ; Digit 1
       .DB     B’11111101        ; Digit 2
       .DB     B’11111011        ; Digit 3
       .DB     B’11110111        ; Digit 4
```

As this above example illustrates, the task uses a callback from the timer and updates a new digit each time the routine gets called by the operating system. It uses a single entry and exit point, is brief and handles a single function. These rules can be broken with care. For example, a task could have a single entry point and multiple exit points, however, it complicates the maintenance of the software.

Another example task would be to scan a keyboard for a button closure. If a button is pressed, the task should load a variable with the scan code for that key. A scan code of 0xFF could be a no-key-pressed state. An example of this would be to use the L PORT of the COP8SGR for 16 keys connected in a matrix to the port. The lower 4 bits will be the drivers (active low) and the upper 4 will be the inputs (pulled-up). When a button is pushed it will short one of 4 lower port pins to one of the upper 4. See *Figure 5*. The task code would look like this.

```
ScanKeys:
        LD      rRowCount, #4        ; Scan 4 row (OK for only 4 passes in MTK)
        LD      bRowSel, #01         ; Use memory location for row select.
        LD      PORTLC, #X’0F        ; Lower 4 bits outputs, upper 4 bits inputs
        LD      PORTLD, #X’FF        ; Pull upper bits high, Clear driver bits
        LD      B, #01               ; Row to scan
ScanKeysLoop:
        LD      A, bRowSel           ; Get scan row
        XOR     A, #X’FF             ; Turn on (active low) scan row
        X       A, PORTLD            ; Send to port data
        LD      A, PORTLP            ; Check the pins on port L for change
        XOR     A, #X’FF             ; Reverse the polarity (Famous sci-fi quote)
        AND     A, #X’F0             ; Strip unused bits
        IFGT    A, #0                ; Hit?
        JP      ScanKeyHit           ; Yes, go see which key it was
        LD      A, bRowSel           ; Update the row pointer
        RC                           ; Clear the carry flag - just in case
        RLC     A                    ; Rotate left to next row
        X       A, bRowSel           ; Save it back
        DRSZ    rRowCount            ; Check the next row
        JP      ScanKeysLoop         ; Loop back
        LD      bScanCode, #X’FF     ; Scan code for no key pressed.
        JP      ScanKeysExit         ; No keys pressed... return to OS
ScanKeyHit:
        IFBIT   4, A                 ; Column 1?
        LD      bScanCode, #0        ;  Yes, must
        IFBIT   5, A                 ; Column 2?
        LD      bScanCode, #1        ;  Yes, must
        IFBIT   6, A                 ; Column 3?
        LD      bScanCode, #2        ;  Yes, must
        IFBIT   7, A                 ; Column 4?
        LD      bScanCode, #3        ;  Yes, must
;
        LD      A, #4                ; Find row
        SUB     A, rRowCount         ; by subtraction
        RC                           ; Reset the carry flag
        RLC     A                    ; Multiply X2
        RLC     A                    ; Multiply X2 (X4)
        ADD     A, bScanCode         ; Add to scan code
        X       A, bScanCode         ; Save it back
```

```
ScanKeysExit:
        LD      PORTLD, #X'00        ; Turn off all outputs to port
        RET                          ; Return to OS
```

Each time the MTK OS calls this routine, the keyboard is scanned. If a key is pressed, its scan code (0–15) is returned in the bScanCode variable. If not, the NO-KEY-PRESSED valued is loaded instead. Note here that the routine completes its function in a single call, much like a subroutine (which it is in reality — a subroutine of the application code). It also uses very little time to scan the entire keyboard. To reduce the time even further, each entry into the routine could scan a single row instead of the entire keyboard. This is why the current row was kept in a local variable. The changes to implement this are quite simple. Instead of looping within the task, you simply loop to the exit routine. Also the row counter must be preloaded elsewhere, not in the beginning of the routine since it is updated each pass of the task. The changes would be as follows.

```
       DRSZ     rRowCount           ; Check the next row
       JP       ScanKeysExit        ; Wait for next entry to check next row...
       LD       rRowCount, #4       ; Load new row count for next fresh start
       LD       bRowSel, #01        ; Use memory location for row select.
       LD       bScanCode, #X'FF    ; Scan code for no key pressed.
       JP       ScanKeysExit        ; No keys pressed... return to OS
ScanKeyHit:
       LD       rRowCount, #4       ; Reload the row counter
       LD       bRowSel, #01        ; Use memory location for row select.

       (CODE THE SAME FROM HERE)
```

CONCLUSIONS

There are many benefits gained by using a multitasking kernel such as that shown here. Time to market can be reduced by creating software tasks that can be used over and over. Once the kernel is stable, new applications can be created in a very short time. Also, sophisticated applications that require many functions to work together and simultaneously can be easily implemented. Use the MTK code in this application note as a starting point to add your own handlers or to develop your own application. Multitasking on the COP8 has never been easier. Enjoy...

OVERVIEW OF OPERATING SYSTEM ROUTINES

This section provides an overview of all the MTK operating system routines. This includes the routines that are accessible by the application code as well as those used internally by the operating system. All OS routines begin with the lower case letters "os". This will make it easy when looking at your code to determine if the routine belongs to the application or the operating system. These routines are grouped by function.

Routine:	**osPUSH**
Use:	This routine pushes the 'A' register onto the OS data stack
Entry:	Accumulator contains data byte to push
Returns:	Accumulator contains original data (Like real PUSH)
Comments:	Used for temporary application storage. This function provides another stack that can be used by the application for holding data.
Routine:	**osPOP**
Use:	This routine pops data from the OS data stack into the 'A' register.
Entry:	(none)
Returns:	Accumulator contains popped data from OS data stack
Comments:	Used for temporary application storage.
Routine:	**osSetMaxTaskTime**
Use:	This routine sets the maximum time any task can run (See text).
Entry:	Accumulator holds time selection if greater than 0. If zero, function disabled.
Returns:	Accumulator non-zero if successful, zero if fails
Comments:	Use this function to increase the robustness of the application. Prevents task lock-up.
Routine:	**osAddTask**
Use:	This routine will add a task entry point to the task list.
Entry:	The stack (SP) holds the entry point address. Push LSB of the task entry point address first and then the MSB. Next call *osAddTask.*
Returns:	Accumulator returns assigned task number if successful, zero if the task was not added.
Comments:	Use this routine to add the tasks to the task list. The application must register all task threads by calling this routine.

Routine:	**osStartTask**
Use:	This routine will start a task thread running.
Entry:	Accumulator contains the task number to start.
Returns:	Accumulator is zero if successful, non-zero if failed.
Comments:	For a task to execute, this routine must be called. Otherwise, the task is skipped and never executed.
Routine:	**osStopTask**
Use:	This routine will stop a task thread from running.
Entry:	Accumulator contains the task number to stop.
Returns:	Accumulator is zero if successful, non-zero if failed.
Comments:	Use this routine to suspend a task from running.
Routine:	**osStart**
Use:	This routine will start the operating system
Entry:	(None required)
Returns:	(Should not return unless fatal system error)
Comments:	This routine should be called after all tasks are added and started. Following the call to this routine, the application should contain an error handler to restart the system in the case of a fatal operating system error. This can occur if a call is made without the proper number of data values on the stack or other bad things...
Routine:	**osMain**
Use:	This routine is the main loop of the kernel. All dispatching of tasks is done from here. Any error handling is also done here.
Entry:	(None required)
Returns:	(Never unless really bad things have happened...)
Comments:	(Used by OS only)
Routine:	**osGetTaskAddress**
Use:	Get Task Address Subroutine
Entry:	Accumulator holds task-1
Returns:	B points to task entry address record
Comments:	(Used by OS only)
Routine:	**osGetFirstTask**
Use:	Gets the first running task number (first to start)
Entry:	(None)
Returns:	First running task number (1-n) in 'A' if successful, zero if it fails
Comments:	(Used by OS only)
Routine:	**osGetNextTask**
Use:	Gets the next running task number (next to start)
Entry:	(None)
Returns:	next running task number (1-n) in 'A' if successful, zero if it fails
Comments:	(Used by OS only)
Routine:	**osUnassigned**
Use:	Unassigned Callback Idle routine
Entry:	(None)
Returns:	(None)
Comments:	(Used by OS only). This routine is used as a safety net in the case a timer or other callback is unassigned and is executed. This will immediately return to the OS with no harm done.
Routine:	**osSetTimer**
Use:	This routine Start a timer with a call back routine.
Entry:	Stack has all data pushed on in this order:
	1. Push LSB of Timer value
	2. Push MSB of Timer value
	3. Push LSB of call back routine address (0 if main task)
	4. Push MSB of call back routine address (0 if main task)
Returns:	Accumulator holds handle number for timer (1-n) if successful
	Accumulator is zero (0) if resource unavailable
Comments:	If the callback is set to zero, Task 1 (first task added) will get the callback with the accumulator set to the system message *"mTimer"*, and 'B' with the timer handle. This facilitates a single routine to handle all operating system messages.
Routine:	**osKillTimer**
Use:	This routine stops a timer and frees its resource.
Entry:	Accumulator contains the non-zero timer handle.
Returns:	Accumulator is zero if successful, non-zero if failed.
Comments:	When a task is done using a timer resource, it must release this resource by calling this routine.
Routine:	**osSignal**
Use:	This routine will move a task to the top of the queue.
Entry:	Accumulator contains the task number to signal next.
Returns:	Accumulator is zero if successful, non-zero if failed.
Comments:	This is useful in real time conditions when a previous event requires faster service. For example, use this routine in a UART service callback to move the parser to the top of the list. It will then be executed next following the current task.
Routine:	**osSetUART**
Use:	This routine will capture a UART to a task and setup the baud rate, framing, data bits, and buffers.
Entry:	'A' holds COM channel request (1 = UART 1, 2 = UART 2, etc.)
	Data is pushed on the stack as follows:
	1. PUSH Configuration Byte (See *Tables 1, 2*)
	2. PUSH LSB of UART RX Call Back routine
	3. PUSH MSB of UART RX Call Back routine
Returns:	If successful, accumulator will return the handle to the UART (1-n).

If failure, accumulator will be zero (0).

Comments: If the callback address is zero (0), Task 1 (first task added) will receive the callback with the accumulator set to the system message mUART and the 'B' with the UART handle. This facilitates a single routine to handle all operating system messages.

TABLE 1. UART Configuration Byte

Bit	Description
0-3	Baud Rate — See BAUD Rate Table
4	Data Bits (0=7, 1=8)
5	Stop Bits (0=1, 1=2)
6	Parity (0=none, 1=On)
7	Parity Type (0=Odd, 1=Even)

TABLE 2. Baud Rate Bits

Bits 3-0	Baud Rate (Bits per Second)
0000	110
0001	134.5
0010	150
0011	300
0100	600
0101	1200
0110	2400
0111	4800
1000	7200
1001	9600
1010	19200
1011	Reserved — Defaults to 9600
1100	Reserved — Defaults to 9600
1101	Reserved — Defaults to 9600
1110	Reserved — Defaults to 9600
1111	Reserved — Defaults to 9600

Routine: **osUARTSend**

Use: This routine adds a byte to the transmit queue of the open UART resource.

Entry: Data byte is on the stack and 'A' holds the handle of the open UART resource.

Returns: If successful, returns zero in 'A'.
If failure, return non-zero in 'A'.

Comments: Use this routine to send data. The transmit queue has two pointers — a read and write pointer. If the pointers are different, the UART transmit routine reads data from the transmit queue and updates the read pointer until they are the same once again. Once they are the same, transmissions stop.

Routine: **osKillUART**

Use: Frees a UART resource

Entry: 'A' holds the handle to the open UART resource

Returns: If successful, returns zero in 'A'.

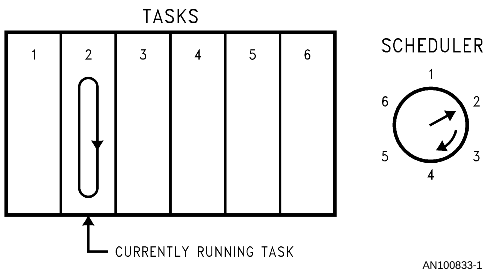
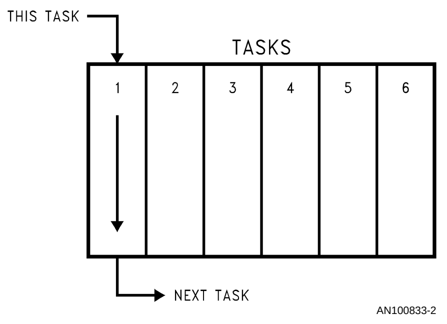


FIGURE 1. Preemptive Multitasking Structure



Scheduler

This Task	Next Task
1	2
2	3
3	4
4	5
5	6
6	1

FIGURE 2. Cooperative Multitasking Structure

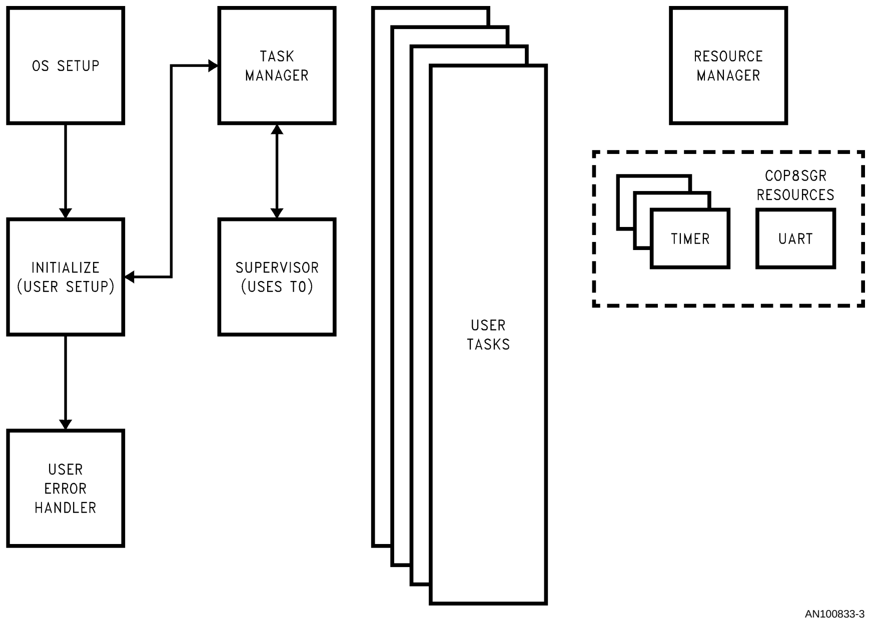


FIGURE 3. COP8SGR Multitasking Kernel Architecture

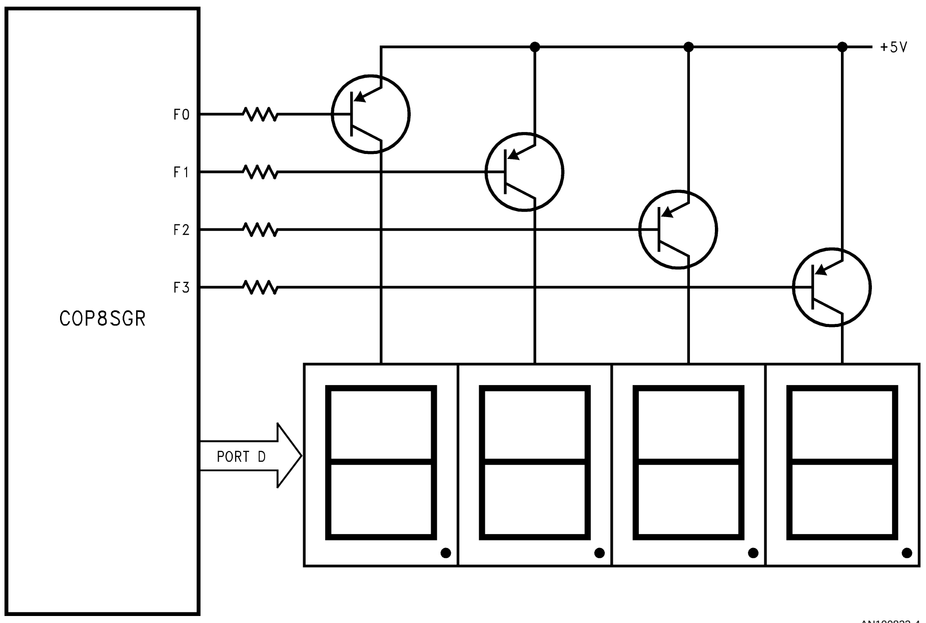


FIGURE 4. Multiplexed 7 Segment Display Task Hookup

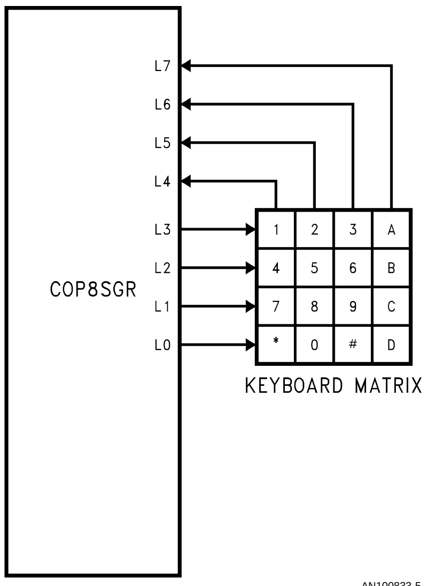


FIGURE 5. Keyboard Scanning Task Hookup

```
                .TITLE  MTK , 'Multi Tasking Kernal'
        ;*********************************************************************
        ; FILENAME: MTK88815.ASM
        ; Multi-Tasking Kernal for simple multithreaded applications
        ; Copyright (C) 1996, 1997 National Semiconductor Corporation
        ;*********************************************************************
        ;                      C O D E   H I S T O R Y
        ;
        ; Comment                                         Date         Version
        ;  -------------------------------------------------------------------
        ;  Code created by Rick Zarr                     06/03/96        1.00
        ;  Conversion to 888 core (COP8SAC)              02/27/97        1.10
        ;  Additions for use with article                08/07/97        1.20
        ;  Conversion to COP8SGR                         01/03/98        1.30
        ;  Additions for Supervised cooperative          01/07/98        1.40
        ;  Updates to fix bugs                           01/29/98        1.41
        ;  Additions to UART functions                   02/24/98        1.50
        ;  -------------------------------------------------------------------
        ;
        ;*********************************************************************
        ;  Notes:
        ;  A) Naming conventions:
     ;  The following naming methods are used through out this source code.
        ; This makes classifying the variables easier when looking through
        ;  the listing.  It also helps prevent association errors since the
     ; assembler cannot tell you if you've associated an incorrect variable
        ;  with an operation.  For example, if you wrote the following:
        ;
        ;       LD      A, xyz          ; Get the data in variable xyz
        ;
        ; and you made the mistake that xyz was in effect a constant, the
        ; assembler would load the location in memory that was pointed to by
        ; the constant. This means you might get something like this:
        ;
        ;       LD      A, 3            ; Get the data in variable xyz
        ;
        ; instead of the intended variable location. This naming convention
        ; can help prevent this.  Here is the same thing using the names.  I
        ; have defined two things, one is the variable name and size and the
    ;other is it's default value.  They can be named almost the same thing
        ; helping to keep track of what's what!
        ;
        ;       LD      A, cXYZ         ; Load the default value for xyz
        ;       X       A, bXYZ         ; Save it into the variable
        ;
        ;Here's my definitions used in most of the source code I write.
        ;  Feel free to adopt this methodology or create your own.
        ;
        ;  Variable prefix naming conventions:
        ;
  ;      a : ARRAY (Group of bytes.                       Example: aData)
  ;      b : BYTE VALUE (Single byte value.               Example: bData)
  ;      w : WORD  (Variable defined as DSW.              Example: wTest)
  ;      f : FLAG BIT (Bit within byte.                   Example: fWorking)
  ;      c : CONSTANT (ROM defined value                  Example: cValue)
  ;      p : POINTER (Byte long pointer into RAM.         Example: pBuffer)
  ;     pc : POINTER CONSTANT (Pointer into EEPROM.    Example: pcMaxMinV)
  ;      r : REGISTER (CPU register R0-RA             Example: rCounter)
  ;     io : I/O PIN (I/O pin on COP8 controller          Example: ioData)
  ;      i : INPUT PIN (Input pin on COP8 controller      Example: iSwitch)
  ;      o : OUTPUT PIN (Output pin on COP8 controller    Example: oLED1)
        ;
        ;*********************************************************************
                .INCLD  COP888EG.INC    ; Example 888 device for demo of MTK
        ;*********************************************************************
        ;
        ; CONSTANTS
```

```
                     ;
                     ; Example
                     ; cConstant = n ; Setup up constants like this
                     ;
                     ;*********************************************************************

        006F       cTopOfRAM       =     0x6F ; Top of RAM for COP87L88EG
        007F       cTopOfSysRAM    =     0x7F ; Top of OS RAM for COP87L88EG
        0001       cSystemSegment  =     1    ; Current system RAM segment
        0003       cRecSize        =     3    ; Size of task records in bytes
        000A       cMaxTasks       =     10   ; Maximum number of tasks that can be loaded

        0060       cBaudPrescale   =     0x60 ; UART Prescaler value 6.5 (01100) for 10MHz
        0010       cCOM1TxBufLen   =     16   ; COM1 Transmitter FIFO length
        0010       cCOM1RxBufLen   =     16   ; COM1 Receiver FIFO Length

                     ;*********************************************************************
                     ;
                     ; OPERATING SYSTEM MESSAGES
                     ;
                     ;*********************************************************************

        0000         mNone           =     X’00  ; Idle message
        0001         mTimer          =     X’01  ; Timer tick / service
        0002         mUART           =     X’02  ; UART Service
        000F         mTaskFailure    =     X’0F  ; Task monitor service

        0010         mUser           =     X’10  ; Beginning of user messages

   0000                    .SECT   OSRAM, SEG    ; Located in segment 1 of the COP87L88EG
                   ;*********************************************************************
                   ;
                   ; VARIABLES
                   ;
                   ; Example:
                   ;  wWord : .DSW 1 ; Word format for 16 bit location
                   ;  bByte : .DSB 1 ; Byte format for 8 bit location
                   ;
                   ;*********************************************************************
   0000            aStore          :     .DSB    8  ; Local store for working with data
   0008            pStack          :     .DSB    1  ; OS Data Stack Pointer

   0009            bTask           :     .DSB    1  ; Currently running task number (1-n)
   000A            bTaskNew        :     .DSB    1  ; Used by OS, Last task number added
   000B            bTaskNext       :     .DSB    1  ; Next active task to switch in
   000C            wAddress        :     .DSW    1  ; 16 bit address (MSB/LSB)
   000E            bMaxTaskTime    :     .DSB    1  ; Max tick count for task monitor
   000F            bTaskTime       :     .DSB    1  ; Remaining

   0010            fStatus         :     .DSB    1  ; Status byte for kernal
       0003        fUART           = 3              ;  Bit 3: UART 1 in use
       0002        fTimer3         = 2              ;  Bit 2 : Timer 3 in use
       0001        fTimer2         = 1              ;  Bit 1 : Timer 2 in use
       0000        fTimer1         = 0              ;  Bit 0 : Timer 1 in use

   0011            wPeriod         :     .DSW    1  ; Work areafor timer period
   0013            wTimerCallBack  :     .DSW    1  ; Work area for call back
   0015            wTimer1CallBack :     .DSW    1  ; Timer 1 callback address
   0017            wTimer2CallBack :     .DSW    1  ; Timer 2 callback address
   0019            wTimer3CallBack :     .DSW    1  ; Timer 3 callback address

   001B            wIntCallBack    :     .DSW    1  ; Hardware interruptcallback

   001D            bUARTSettings   :     .DSB    1  ; Current UART settings
       0004        fDataBits       = 4              ;  Data bits (0=7, 1=8)
        0005       fStopBits       = 5              ;  Stop Bits (0=1, 1=2)
        0006       fParitySel      = 6              ;  Parity Select (0=none, 1=On)
```

```
     0007         fParityType     = 7             ;  Parity Type (0=Odd, 1=Even)
0001E             wUARTRxCallBack :     .DSW    1  ; UART RX Call back address

0020              bCOM1RxBuffer   :     .DSB    cCOM1RxBufLen ; Setup buffer for RX
0030              pCOM1RxWrite    :     .DSB    1          ; Write pointer to RX buffer
0031              pCOM1RxRead     :     .DSB    1          ; Read pointer to RX buffer
0032              bCOM1TxBuffer   :     .DSB    cCOM1TxBufLen ; Setup buffer for TX
0042              pCOM1TxWrite    :     .DSB    1          ; Write pointer to TX buffer
0043              pCOM1TxRead     :     .DSB    1          ; Read pointer to TX buffer
0044              bCOM1Status     :     .DSB    1              ; Status byte for COM1
     0007          fTxBusy        = 7                          ;  Tx FIFO Full
     0006          fRxBusy        = 6                          ;  Rx FIFO Full

                  ;************************************************************************
                  ;
                  ;              T A S K  R E C O R D   S T R U C T U R E
                  ;
                  ;  fTaskStatus  : .DSB 1 ; Flags for each task
                  ;   fActive     = 0      ;  Bit 0 : State (0=Stopped, 1=Running)
                  ;  bAddrLSB     : .DSB 1 ; Task Address LSB
                  ;  bAddrMSB     : .DSB 1 ; Task Address MSB
                  ;
                  ;************************************************************************
0045         aTaskRecs       :       .DSB    cRecSize * cMaxTasks ; Array of task records
     0000         fActive         = 0                             ; Active flag for task

                  ;*************** I M P O R T   F U N C T I O N S ****************
                       .EXTRN Initialize:ROM  ; Inializes application code

                  ;*************** E X P O R T   F U N C T I O N S ****************
                       .PUBLIC osStart        ; Starts the MTK running
                       .PUBLIC osAddTask      ; Adds a task to the MTK task list
                       .PUBLIC osStartTask    ; Starts a task running
                       .PUBLIC osStopTask     ; Stops a task
       .PUBLIC osSetMaxTaskTime ; Sets the maximum time a task can run before being killed
                       .PUBLIC osSignal       ; Moves thread to top of queue
                       .PUBLIC osPUSH         ; PUSHes a byte onto the OS stack from ’A’
                       .PUBLIC osPOP          ; POPs a byte from the OS stack to ’A’
       .PUBLIC osSetCommChannel ; Captures and initializes a communications resource
                       .PUBLIC osUARTGetChar  ; Returns a byte from the UART RX queue
                       .PUBLIC osUARTSend     ; Sends a character to the UART
                       .PUBLIC osSetTimer     ; Captures and sets up a timer resource
                       .PUBLIC osKillTimer    ; Stops and frees a timer resource
                    ;*********************** C O D E   B E G I N S *************
CODE SECTION
                         .FORM  ’CODE SECTION’
0000                     .SECT   CODE, ROM, ABS=0x0000
                 ;
0000 DD6F    3   Start:  LD      SP, #cTopOfRAM          ; Top of Stack for COP87L88EG
                 ;
0002             ClearRAM:
0002 DF00    3           LD      S, #0                   ; Point to RAM segment 0
0004 9F6F    2           LD      B, #cTopOfRAM           ; Setup pointer into Main RAM
0006             ClearRAMLoop1:
0006 9E00    2           LD      [B], #000               ; Clear RAM location
0008 CE      3           DRSZ    B                  ; Update counter, skip jump if done
0009 FC      3           JP      ClearRAMLoop1           ; Continue until done
000A BC0000  3           LD      000 , #00               ; Clear last location
                 ;
000D DF01    3           LD      S, #1                   ; Point to RAM segment 1
000F 9F7F    2           LD      B, #cTopOfSysRAM        ; Setup pointer into OS RAM
0011             ClearRAMLoop2:
0011 9E00    2           LD      [B], #000               ; Clear RAM location
0013 CE      3           DRSZ    B                  ; Update counter, skip jump if done
0014 FC      3           JP      ClearRAMLoop2           ; Continue until done
0015 BC0000  3           LD      000 , #00               ; Clear last location

```

```
                    ;
0018 BC087F  3 R         LD      pStack, #cTopOfSysRAM     ; Setup OS Data Stack
                    ;
001B 9F15    2 R         LD      B, #wTimer1CallBack       ; Initialize callbacks to nil
001D 9ACF    2      LD      [B+], #LOW(osUnassigned)  ;  to prevent the OS from crashing
001F 9A00    2      LD      [B+], #HIGH(osUnassigned) ;  if the timer is active without
0021 9ACF    2           LD      [B+], #LOW(osUnassigned)  ;  the callback assigned.
0023 9A00    2           LD      [B+], #HIGH(osUnassigned) ;
0025 9ACF    2           LD      [B+], #LOW(osUnassigned)  ;
0027 9A00    2           LD      [B+], #HIGH(osUnassigned) ;

0029 DF00    3           LD      S, #0                   ; Point to main user RAM area

002B 9DB9    3           LD      A, RBUF                 ; Receive and rough characters
002D BCBA00  3           LD      ENU, #B'00000000        ; Clear UART registers
0030 BCBB00  3           LD      ENUR, #B'00000000       ;
0033 BCBC00  3           LD      ENUI, #B'00000000       ;

0036 BCEE80  3           LD      CNTRL, #B'10000000      ; Setup timer 1A
0039 BCC685  3           LD      T2CNTRL, #B'10000101    ; Setup timer 2A,B
003C BCB685  3           LD      T3CNTRL, #B'10000101    ; Setup timer 3A,B
003F BCE801  3           LD      ICNTRL, #B'00000001     ; Setup timer 0, 1B
0042 BCEF11  3           LD      PSW, #B'00010001        ; Setup PSW, Timer 1A, GIE

0045 AC0000  > 4 X       JP      Initialize              ; Go setup tasks

                    ;**********************************************************************
                    ;
                    ;    Multitasking kernal routines
                    ;
                    ;**********************************************************************
                    ;
                    ;    Routine: osAddTask
                    ;    Use: This routine will add a task entry point to the task list.
                    ;
                    ;    Entry: Stack (SP) holds return address and data. Push LSB first
                    ;             and then push MSB of task entry point callback routine.
                    ;             Then call osAddTask
                    ;    Returns: Accumulator returns assigned task number if successful,
                    ;             zero if the task was not added.
                    ;    Alters:  'A', 'B', and the SP
                    ;
                    ;**********************************************************************
0048                osAddTask:
0048 9DFF    3           LD      A, S                    ; Get current RAM segment
004A DF01    3           LD      S, #1                   ; Switch to OS data segment
004C 9C00    3 R         X       A, aStore+0        ; Save for later to restore segment
004E 9D0A    3 R         LD      A, bTaskNew             ; Update New task counter
0050 8A      1           INC     A                       ; Increment to next one
0051 930A    2           IFGT    A, #cMaxTasks           ; Check if larger than max task
0053 1C      3           JP      osAddTaskError          ;  yes, Report error
0054 9C0A    3 R         X       A, bTaskNew             ;  No, save it back
                    ;
0056 8C      3           POP     A                       ; Get Return Address MSB
0057 9C0D    3 R         X       A, wAddress+1           ; Save in MSB of wAddress
0059 8C      3           POP     A                       ; Get Return Address LSB
005A 9C0C    3 R         X       A, wAddress             ; Save in LSBof wAddress
                    ;
005C 9D0A    3 R         LD      A, bTaskNew             ; Get New Task pointer
005E 8B      1           DEC     A                       ; Correct for 0 reference
005F 30A4    5           JSR     osGetTaskAddress        ; Get address, pointed to by 'A'

0061 AA      2           LD      A, [B+]                ; Move past flags byte
0062 AA      2           LD      A, [B+]                ; Move past LSB of callback
0063 8C      3           POP     A                      ; Get callback LSB
0064 A3      2           X       A, [B-]                ; Store MSB, increment B
0065 8C      3           POP     A                      ; Get callback MSB
```

```
0066 A6      1           X       A, [B]                 ; Store LSB, increment B
                        ;
0067 9D0C    3 R         LD      A, wAddress            ; Restore return address
0069 67      3           PUSH    A                      ; Start with LSB
006A 9D0D    3 R         LD      A, wAddress+1          ;
006C 67      3           PUSH    A                      ; Now add MSB
006D 9D0A    3 R         LD      A, bTaskNew            ; Get the current task number
006F 01      3           JP      osAddTaskEnd           ; Return handle of task
0070                    osAddTaskError:
0070 64      1           CLR     A                      ; Report error
0071                    osAddTaskEnd:
0071 67      3           PUSH    A                      ; Save return value
0072 9D00    3 R         LD      A, aStore+0            ; Get old segment
0074 9CFF    3           X       A, S                   ; Restore old segment
0076 8C      3           POP     A                      ; Get return value
0077 8E      5           RET                            ; Return to calling routine


                        ;*************************************************************
                        ;
                        ;   Routine: osStartTask
                        ;   Use:     This routine will start running a task thread.
                        ;
                        ;   Entry:   Accumulator contains the task number to start.
                        ;   Returns: Accumulator is zero if successful, non-zero if failed.
                        ;
                        ;*********************************************************************
0078                    osStartTask:
0078 67      3           PUSH    A                      ; Save 'A' for now
0079 9DFF    3           LD      A, S                   ; Get current segment
007B DF01    3           LD      S, #1                  ; Switch to OS RAM Page
007D 9C00    3 R         X       A, aStore+0            ; Save old RAM segment
007F 8C      3           POP     A                      ; Get task number off main stack
0080 BD0A83  4 R         IFGT    A, bTaskNew            ; Is the task number valid?
0083 06      3           JP      osStartTaskError       ; No, don't start
                        ;
0084 8B      1           DEC     A                      ; Use corrected pointer (p-1)
0085 30A4    5           JSR     osGetTaskAddress   ; Get a pointer into the task buffer
0087 78      1           SBIT    fActive, [B]           ; Set Start Task flag in MSB
0088 64      1           CLR     A                      ; Clear 'A' to signal start
0089 02      3           JP      osStartTaskEnd         ; End routine
008A                    osStartTaskError:
008A 98FF    2           LD      A, #0xFF               ; Signal error (non-zero)
008C                    osStartTaskEnd:
008C 67      3           PUSH    A                      ; Save return value on main stack
008D 9D00    3 R         LD      A, aStore+0            ; Get old RAM segment from store
008F 9CFF    3           X       A, S                   ; Restore old RAM segment
0091 8C      3           POP     A                      ; Get return value
0092 8E      5           RET                            ; Return to calling task

                        ;*********************************************************************
                        ;
                        ;      Routine: osStopTask
                        ;      Use:     This routine will stop a task thread from running.
                        ;
                        ;      Entry:   Accumulator contains the task number to stop.
                        ;  Returns: Accumulator is zero if successful, non-zero if failed.
                        ;
                        ;*********************************************************************
0093                    osStopTask:
0093 DF01    3            LD     S, #1                  ; Switch to OS RAM Page
0095 BD0A83  4 R          IFGT   A, bTaskNew            ; Is the task number valid?
0098 06      3            JP     osStopTaskError        ; No, don't start
                        ;
0099 8B      1            DEC    A                      ; Use corrected pointer (p-1)
009A 30A4    5            JSR    osGetTaskAddress  ; Get a pointer into the task buffer
009C 78      1            SBIT   fActive, [B]           ; Start Task
```

```
009D 64      1            CLR    A                      ; Clear 'A' to signal start
009E 02      3            JP     osStopTaskEnd          ; End routine
009F                osStopTaskError:
009F 98FF    2            LD     A, #0xFF               ; Signal error (non-zero)
00A1                osStopTaskEnd:
00A1 DF00    3            LD     S, #0                  ; Switch back to user RAM area
00A3 8E      5            RET                           ; Return to calling task

                    ;*********************************************************************
                    ;
                    ;     Get Task Address Subroutine
                    ;     (Used by OS only)
                    ;
                    ; osGetTaskAddress : A is task-1, B returns address to first entry
                    ;
                    ;*********************************************************************
00A4                osGetTaskAddress:
00A4 9CFE    3            X      A, B                   ; Use 'B' to do math
00A6 9DFE    3            LD     A, B                   ; Get task pointer back
00A8 BDFE84  4            ADD    A, B                   ; Multiply x2 for pointer
00AB BDFE84  4            ADD    A, B                   ; Multiply x3 for pointer
00AE 9445    2 R          ADD    A, #aTaskRecs       ; Add base address of record buffer
00B0 9CFE    3            X      A, B                   ; Use 'B' now as pointer
00B2 8E      5            RET                           ;

                    ;*********************************************************************
                    ;
                    ;     Get First and Next running Task Subroutine
                    ;     (Used by OS only)
                    ;
           ;     osGetFirstTask : Returns first running task (1-n) in 'A' if successfull
           ;     osGetNextTask  : Returns next running task (1-n) in 'A' if successfull
                    ;
                    ;     If either call fails, 'A' is zero
                    ;
                    ;*********************************************************************
00B3                osGetFirstTask:
00B3 9800    2            LD     A, #0                  ; First task record
00B5 67      3            PUSH   A                      ; Save it on the stack
00B6 03      3            JP     osGetNextTaskLoop      ; go see what's enabled
00B7                osGetNextTask:
00B7 9D09    3 R          LD     A, bTask        ; Current Task+1 (Already incremented)
00B9 67      3            PUSH   A                      ; Save it for later
00BA                osGetNextTaskLoop:
00BA 30A4    5            JSR    osGetTaskAddress       ; Go get the address of the task
00BC 70      1            IFBIT   fActive, [B]          ; Active?
00BD 0C      3            JP      osGetNextTaskHit      ; Yes, we got the next active one
                    ;
00BE 8C      3            POP     A                     ; Get the old count
00BF 8A      1            INC     A                     ; Update it
00C0 BD0A83  4 R          IFGT    A, bTaskNew           ; Past last one?
00C3 64      1            CLR     A                     ; Yes, Start at beginnning
00C4 BD0982  4 R          IFEQ    A, bTask              ; Hit them all?
00C7 05      3            JP      osGetNextTaskError   ; Yes, Nothing is running. Bail...
00C8 67      3            PUSH    A                     ; Save it for later
00C9 F0      3            JP      osGetNextTaskLoop     ; Try again

00CA                osGetNextTaskHit:
00CA 8C      3            POP     A                 ; Get the task count off the stack
00CB 8A      1            INC     A                     ; Adjust task number to 1-n
00CC 01      3            JP      osGetNextTaskDone     ;
00CD                osGetNextTaskError:
00CD 64      1            CLR     A                     ; Return invalid handle
00CE                osGetNextTaskDone:
00CE 8E      5            RET                           ; Done

                    ;*********************************************************************
                    ;
                    ;     Unassigned Callback Idle routine
                    ;     (Used by OS only)
                    ;
             ;     This routine is used as a safty net incase a timer or other callback
             ;  is unassigned and is executed.  This will immediately return to the OS.
                    ;
                    ;*********************************************************************
00CF                osUnassigned:                         ; Do nothing and return
00CF 8E        5          RET                             ;

INTERRUPT ROUTINE
                          .FORM   ’INTERRUPT ROUTINE’
                    ;*********************************************************************
                    ;
                    ;     I N T E R R U P T   H A N D L E R
                    ;
                    ;     NOTES
                    ;     -----
           ;       This routine is the entry point for all vectors .  The VIS instruction
         ;       will find the correct vector for interrupt. It is the responsibility
                 ;       of each service routine to clear the cause of the interrupt. A
             ;       template is used for each of the existing interrupts.  The default
           ;       device for this source is the COP8SA family. The interrupts for this
           ;       device are all supported.  If this MTK is migrated to another feature
           ;       family device, simply replace the ’ServiceUnused’ routine with the
                    ;       missing routines.
                    ;
                    ;*********************************************************************
00FF                      .ORG  0x00FF                    ; Force to 0x00FF for interrupt
00FF                Interrupt:
00FF 67      3            PUSH    A                       ; Save ’A’
0100 9DFE    3            LD      A, B                    ; Save ’B’
0102 67      3            PUSH    A                       ;
0103 9DEF    3            LD      A, PSW                  ; Save ’PSW’
0105 67      3            PUSH    A                       ;
0106 9DFF    3            LD      A, S                    ; Save segment
0108 67      3            PUSH    A                       ;
0109 B4      5            VIS                             ; Find interrupt vector
                    ;*********************************************************************
010A                ServiceUnused:
010A B5      1            RPND                      ; Incase it’s set, clear SW pending
010B 2200  > 3            JP      Restore                 ; Just return
                    ;*********************************************************************
010D                ServiceWakeup:


                          ; *** Check which ’L’ Port pin caused the interrupt here ***

010D BCCA00  3            LD      WKPND, #0               ; Clear all pending flags
0110 2200  > 3            JP      Restore                 ; Return
                    ;*********************************************************************
0112                ServiceTimerT0:
0112 BDE86D  4            RBIT    T0PND, ICNTRL           ; Clear pending flag
0115 DF01    3            LD      S, #1                   ; Switch to OS segment
0117 9D0F    3 R          LD      A, bTaskTime            ; Decrement task watcher counter
0119 8B      1            DEC     A                       ;
011A 9300    2            IFGT    A, #0                   ; Did current running task die?
011C 1F      3            JP      ServiceTimerT0Exit      ; No, still running
                    ;
011D 9D09    3 R          LD      A, bTask                ; Did the main message loop die?
011F 9201    2            IFEQ    A, #1                   ;
0121 2000    3            JMP     Start                 ; Yes, restart the microcontroller
                                                  ; No, Stop the thread and report it...
0123 8B      1            DEC     A                     ; Adjust task number to get address
0124 30A4    5            JSR     osGetTaskAddress      ; Retrieve the task address.
0126 68      1            RBIT    fActive, [B]            ; Disable the task
```

```
                           ;
0127 9800    2                   LD      A, #LOW(Restore)      ; Push return address on stack
0129 67      3                   PUSH    A                     ;
012A 9802    2                   LD      A, #HIGH(Restore)     ;
012C 67      3                   PUSH    A                     ;
012D 9D46    3 R                 LD      A, aTaskRecs+1   ; Push address of main task onto stack
012F 67      3                   PUSH    A                     ;
0130 9D47    3 R                 LD      A, aTaskRecs+2        ;
0132 67      3                   PUSH    A                     ;
0133 9D09    3 R                 LD      A, bTask              ; Get active task that failed
0135 9CFE    3                   X       A, B                  ; Save in ’B’
0137 980F    2                   LD      A, #mTaskFailure      ; Setup message to task
0139 DF00    3                   LD      S, #0             ; Switch to User RAM for call back
013B 8E      5                   RET                           ; Call message loop.

013C                       ServiceTimerT0Exit:
013C 9C0F    3 R                 X       A, bTaskTime          ; Save new count back to RAM
013E 2200  > 3                   JP      Restore               ; Return
                           ;*********************************************************************
0140                       ServiceTimerT1A:
0140 BDEF6D  4                   RBIT    TPND, PSW             ; Clear Pending flag
0143                       ServiceTimerT1:
0143 DF01    3                   LD      S, #1                 ; Switch to OS RAM page
0145 9800    2                   LD      A, #LOW(Restore       ; Push return address on stack
0147 67      3                   PUSH    A                     ;
0148 9802    2                   LD      A, #HIGH(Restore)     ;
014A 67      3                   PUSH    A                     ;
014B 9D15    3 R                 LD      A, wTimer1CallBack    ; Push callback address on stack
014D 67      3                   PUSH    A                     ;
014E 9D16    3 R                 LD      A, wTimer1CallBack+   ;
0150 67      3                   PUSH    A                     ;
0151 DF00    3                   LD      S, #0             ; Switch to User RAM for call back
0153 8E      5                   RET                       ; Go service it, return to RESTORE

                           ;*********************************************************************
0143                       ServiceTimerT1:
0143 DF01    3                   LD      S, #1               ; Switch to OS RAM page
0154                       ServiceTimerT1B:
0154 BDE869  4                   RBIT    T1PNDB, ICNTRL       ; Clear Pending flag
0157 EB      3                   JP      ServiceTimerT1       ; Go service call-back
                           ;*********************************************************************
0158                       ServiceTimerT2A:
0158 BDC66B  4                   RBIT    T2PNDA, T2CNTRL      ; Clear Pending flag
015B                       ServiceTimerT2:
015B DF01    3                   LD      S, #1                ; Switch to OS RAM page
015D 9800    2                   LD      A, #LOW(Restore)     ; Push return address on stack
015F 67      3                   PUSH    A                    ;
0160 9802    2                   LD      A, #HIGH(Restore)    ;
0162 67      3                   PUSH    A                    ;
0163 9D17    3 R                 LD      A, wTimer2CallBack   ; Push callback address on stack
0165 67      3                   PUSH    A                    ;
0166 9D18    3 R                 LD      A, wTimer2CallBack+1 ;
0168 67      3                   PUSH    A                    ;
0169 DF00    3                   LD      S, #0                ; Switch to User RAM for call back
016B 8E      5                   RET                          ; Go service it, return to RESTORE
                           ;*********************************************************************
016C                       ServiceTimerT2B:
016C BDC669  4                   RBIT    T2PNDB, T2CNTRL      ; Clear Pending flag
016F EB      3                   JP      ServiceTimerT2       ; Go service call-back
                           ;*********************************************************************
0170                       ServiceTimerT3A:
0170 BDB66B  4                   RBIT    T3PNDA, T3CNTRL      ; Clear Pending flag
0173                       ServiceTimerT3:
0173 DF01    3                   LD      S, #1                ; Switch to OS RAM page
0175 9800    2                   LD      A, #LOW(Restore)     ; Push return address on stack
0177 67      3                   PUSH    A                    ;
0178 9802    2                   LD      A, #HIGH(Restore)    ;
```

```
017A 67      3           PUSH    A                    ;
017B 9D19    3 R         LD      A, wTimer3CallBack   ; Push callback address on stack
017D 67      3           PUSH    A                    ;
017E 9D1A    3 R         LD      A, wTimer3CallBack+1 ;
0180 67      3           PUSH    A                    ;
0181 DF00    3           LD      S, #0                ; Switch to User RAM for call back
0183 8E      5           RET                          ; Go service it, return to RESTORE
                   ;*****************************************************************
0184               ServiceTimerT3B:
0184 BDB669  4           RBIT    T3PNDB, T3CNTRL      ; Clear Pending flag
0187 EB      3           JP      ServiceTimerT3       ; Go service call-back
                   ;*****************************************************************
0188               ServiceUART_TX:
0188 DF01    3           LD      S, #1                ; Switch to OS RAM page
018A 9D43    3 R         LD      A, pCOM1TxRead  ; Get the transmit buffer read pointer
018C 9CFE    3           X       A, B                 ; Use ’B’ as the pointer
018E AA      2           LD      A, [B+]              ; Get the data
018F 9CB8    3           X       A, TBUF              ; Move to UART Tx register
0191 BD446F  4 R         RBIT    fTxBusy, bCOM1Status ; Update status, free FIFO
0194 9DFE    3           LD      A, B                 ; Get new pointer back
0196 9242    2 R         IFEQ    A, #bCOM1TxBuffer+cCOM1TxBufLen ; past buffer end?
0198 9832    2 R         LD      A, #bCOM1TxBuffer    ; Reload start of buffer
019A BD4282  4 R         IFEQ    A, pCOM1TxWrite      ; All data sent?
019D BDBC68  4           RBIT    ETI, ENUI            ;  Yes, Stop any new interrupts...
01A0 9C43    3 R         X       A, pCOM1TxRead       ; Save new pointer value
01A2 2200  > 3           JP      Restore              ; Return
                   ;*****************************************************************
01A4               ServiceUART_RX:
01A4 DF01    3           LD      S, #1                ; Switch to OS RAM page
01A6 9D30    3 R         LD      A, pCOM1RxWrite      ; Get the write pointer
01A8 9CFE    3           X       A, B                 ; Use ’B’ as pointer
01AA 9DB9    3           LD      A, RBUF              ; Get received character
01AC A2      2           X       A, [B+]              ; Save into receive buffer
01AD 9DFE    3           LD      A, B                 ; Update pointer
01AF 9230    2 R         IFEQ    A, #bCOM1RxBuffer+cCOM1RxBufLen; Past end of buffer
01B1 9820    2 R         LD      A, #bCOM1RxBuffer    ; Reload start of buffer
01B3 9C30    3 R         X       A, pCOM1RxWrite      ; Save pointer
01B5 9800    2           LD      A, #LOW(Restore)     ; Push returnaddress on stack
01B7 67      3           PUSH    A                    ;
01B8 9802    2           LD      A, #HIGH(Restore)    ;
01BA 67      3           PUSH    A                    ;
01BB 9D1E    3 R         LD      A, wUARTRxCallBack   ; Push callback address on stack
01BD 67      3           PUSH    A                    ;
01BE 9D1F    3R          LD      A, wUARTRxCallBack+1 ;
01C0 67      3           PUSH    A                    ;
01C1 DF00    3           LD      S, #0                ; Switch to User RAM for call back
01C3 8E      5           RET             ; Launch the call back thread to service data

                   ;*****************************************************************
01C4               ServiceSoftware:
01C4 B5      1           RPND                         ; Clear SW interrupt (NMI) flag
01C5 8C      3           POP     A                    ; Clear old return address
01C6 8C      3           POP     A                    ;
01C7 DF00    3           LD      S, #0                ; Point to users RAM space
01C9 8E      5           RET                          ; Return to Users osStart Call

                   ;*****************************************************************
01CA               ServiceMicrowire:
01CA BDE86B  4           RBIT    WPND, ICNTRL         ; Clear microwire pending flag
01CD 2200  > 3           JP      Restore              ; Return
                   ;*****************************************************************
01CF               ServiceExternal:
01CF BDEF6B  4           RBIT    IPND, PSW      ; Clear the external interrupt pending
01D2 2200  > 3           JP      Restore              ; Return

                   ;*****************************************************************
01E0                     .ORG    0x01E0               ; Vector table begins at 0x01E0
```

```
01E0 010A                 .ADDRW  ServiceUnused  ; Default vector (VIS with no interrupt)
01E2 010D                 .ADDRW  ServiceWakeup        ; Wakeup vector
01E4 0184                 .ADDRW  ServiceTimerT3B      ; Timer 3B vector
01E6 0170                 .ADDRW  ServiceTimerT3A      ; Timer 3A vector
01E8 016C                 .ADDRW ServiceTimerT2B       ; Timer 2B vector
01EA 0158                 .ADDRW  ServiceTimerT2       ; Timer 2A vector
01EC 0188                 .ADDRW  ServiceUART_TX       ; UART Transmit vector
01EE 01A4                 .ADDRW  ServiceUART_RX       ; UART Receive vector
01F0 010A                 .ADDRW  ServiceUnused        ;(Reserved for future use)
01F2 01CA                 .ADDRW  ServiceMicrowire     ; Microwire vector
01F4 0154                 .ADDRW  ServiceTimerT1B      ; Timer1B vector
01F6 0140                 .ADDRW  ServiceTimerT1A      ; Timer 1A vector
01F8 0112                 .ADDRW  ServiceTimerT0       ; Timer 0 vector
01FA 01CF                 .ADDRW  ServiceExternal      ; External pin vector
01FC010A                  .ADDRW  ServiceUnused        ; (Reserved for future use)
01FE 01C4                 .ADDRW  ServiceSoftware      ; NMI - Software interrupt
                    ;*********************************************************************
0200                Restore:
                    ;
0200 8C      3            POP     A                    ; Get old segment
0201 9CFF    3            X       A, S                 ; Restore segment register
0203 8C      3            POP     A                    ; Restore 'PSW' Carry flags
0204 BDEF6E  4            RBIT    C, PSW               ; Assume no carry
0207 6040    2            IFBIT   C, A                 ; Was there a carry?
0209 BDEF7E  4            SBIT    C, PSW               ;  Yes, set it up again
020C BDEF6F  4            RBIT    HC, PSW              ; Assume no half carry
020F 6080    2            IFBIT   HC, A                ; Was there a half carry?
0211 BDEF7F  4            SBIT    HC, PSW              ;  Yes, set it up again
0214 8C      3            POP     A                    ; Restore 'B'
0215 9CFE    3            X       A, B                 ;
0217 8C      3            POP     A                    ; Restore 'A'
0218 8F      5            RETI                         ; Return from interrupt

                    ;*********************************************************************

                    ;*********************************************************************
                    ;
                    ;     Routine:  osPUSH
              ;     Use:      This routine pushes the 'A' register onto the OS data stack
                    ;     Entry:    Accumulator contains data byte to push
                    ;     Returns:  Accumulator contains original data (Like real PUSH)
                    ;     Comments: Used for user temporary storage
                    ;
                    ;*********************************************************************
0219                osPUSH:
0219 67      3            PUSH    A                    ; Save 'A' for page switch
021A 9DFF    3            LD      A, S                 ; Get old segment
021C DF01    3            LD      S, #1                ; Switch to the OS Segment
021E 9C00    3 R          X       A, aStore+0          ; Save
0220 9DFE    3            LD      A, B                 ; Save 'B' for now
0222 9C01    3 R          X       A, aStore+1
0224 9D08    3 R          LD      A, pStack            ; Get the stack pointer
0226 9CFE    3            X       A, B                 ; Stick it into 'B' for pointer
0228 8C      3            POP     A                    ; Get the data from the main stack
0229 67      3            PUSH    A            ; Put it back on the main stack for later
022A A3      2            X       A, [B-]              ; Push data, Adjust pointer
022B 9DFE    3            LD      A, B                 ; Save new stack pointer
022D 9C08    3 R          X       A, pStack            ;  in pointer register
022F 9D01    3 R          LD      A, aStore+1          ; Restore 'B'
0231 9CFE    3            X       A, B                 ;
0233 9D00    3 R          LD      A, aStore+0          ; Get old RAM segment
0235 9CFF    3            X       A, S                 ; Restore the old segment
0237 8C      3            POP     A          ; return original 'A' value (like real PUSH)
0238 8E      5            RET                          ; Go do some more stuff...

                    ;*********************************************************************
                    ;
```

```
                        ;     Routine:  osPOP
                        ;     Use:      This routine pops data from the OS data stack into
                        ;               the 'A' register.
                        ;     Entry:    (none)
                        ;     Returns:  Accumulator contains poped data from OS data stack
                        ;     Comments: Used for user temporary storage
                        ;
                        ;*********************************************************************
0239                    osPOP:
0239 9DFF    3                LD      A, S                 ; Get current RAM segment
023B DF01    3                LD      S, #1                ; Switch to the OS Segment
023D 9C00    3 R              X       A, aStore+0          ; Save for later
023F 9DFE    3                LD      A, B                 ; Save 'B'
0241 9C01    3 R              X       A, aStore+1          ;  in store+1
0243 9D08    3 R              LD      A, pStack     ; Get the current OS Data Stack Pointer
0245 927F    2                IFEQ    A, #cTopOfSysRAM     ; At the top?
0247 0E      3                JP      osPOPError        ;  Yes, bad news... User screwed up.
0248 8A      1                INC     A                    ; Adjust Stack pointer
0249 9C08    3 R              X       A, pStack            ; Save it back
024B 9D08    3 R              LD      A, pStack            ; Reloadpointer
024D 9CFE    3                X       A, B                 ; Use 'B' as pointer into RAM
024F AE      1                LD      A, [B]               ; Get the data from the stack
0250 67      3                PUSH    A                    ; Save it for return value
0251 9D01    3 R              LD      A, aStore+1          ; Restore 'B'
0253 9CFE    3                X       A, B                 ;  to original value
0255 02      3                JP     osPOPExit             ; Done, return
0256                    osPOPError:
0256 64      1                CLR     A               ; Put '0' on stack for return value
0257 67      3                PUSH   A                     ; Save return value
0258                    osPOPExit:
0258 9D00    3 R              LD      A, aStore+0          ; Restore user RAM segment
025A 9CFF    3                X       A, S
025C 8C      3                POP     A                    ; Return value from OS Stack
025D 8E      5                RET                          ; Go do some more stuff...


                        ;*********************************************************************
                        ;
                        ;     Routine: osSetMaxTaskTime
                 ;     Use:     This routine enables the task watcher and sets the time limit.
                        ;
              ;     Entry:  Accumulator holds the time constant, if zero function disabled.
                        ;
                        ;     Returns: Accumulatorreturns assigned task number if successful,
                        ;              zero if the task was not added.
                        ;     Alters:  'A'
                        ;
                        ;*********************************************************************
025E                    osSetMaxTaskTime:
025E DF01    3                LD      S, #1                ; Switch to OS segment
0260 9200    2                IFEQ    A, #0                ; Turn function off?
0262 0A      3                JP      osSMTT_Off           ;  Yes, turn it off, return.
0263 9C0E    3 R              X       A, bMaxTaskTime      ; Save max task time
0265 9D0E    3 R              LD      A, bMaxTaskTime      ; Get it back
0267 9C0F    3 R              X       A, bTaskTime         ; Transfer to register for counter
0269 BDE87C  4                SBIT    T0EN, ICNTRL         ; Turn on the timer interrupt
026C 09      3                JP      osSMTT_Exit          ; Done, return
026D                    osSMTT_Off:
026D BDE86C  4                RBIT    T0EN, ICNTRL         ; Disable the T0 interrupt
0270 BC0F00  3 R              LD      bTaskTime, #0        ; Clear registers and counters
0273 BC0E00  3 R              LD      bMaxTaskTime, #0     ;
0276                    osSMTT_Exit:
0276 DF00    3                LD      S, #0                ; Restore application segment
0278 8E      5                RET                          ; Return

                        ;*********************************************************************
                        ;
```

```
                       ;     Routine: osSetTimer
                 ;     Use:     This routine Start a timer with a call back routine. If the
                   ;             call back is set to zero, Task 1 (Main task) will get the
                   ;             call with the accumulator set to cmTimer, and 'B' with the
                       ;             timer handle.
                       ;
                       ;     Entry:   Stack has all data pushed on in this order:
                       ;             1) Push LSB of Timer value
                       ;             2) Push MSB of Timer value
                   ;             3) Push LSB of call back routine address (0 if main task)
                   ;             4) Push MSB of call back routine address (0 if main task)
               ;     Returns: Accumulator holds handle number for timer (1-n) if successful
                       ;             Accumulator is zero (0) if resource unavailable
                       ;
                       ;*******************************************************************
0279                   osSetTimer:
0279 DF01    3               LD      S, #1                ; Switch Data RAM segment to OS
027B 8C      3               POP     A                    ; Get MSB of calling routine
027C 9C0D    3 R             X       A, wAddress+1        ; Save for now
027E 8C      3               POP     A                    ; Get LSB of calling routine
027F 9C0C    3 R             X       A, wAddress          ; Save for now
0281 8C      3               POP     A                    ; Get MSB of Call Back routine
0282 9C14    3 R             X       A, wTimerCallBack+1  ; Save for now
0284 8C      3               POP     A                    ; Get LSB of Call Back routine
0285 9C13    3 R             X       A, wTimerCallBack    ; Save for now
0287 8C      3               POP     A                    ; Get MSB of time period
0288 9C12    3 R             X       A, wPeriod+1         ; Save for now
028A 8C      3               POP     A                    ; Get LSB of time period
028B 9C11    3 R             X       A  wPeriod           ; Save for now
028D 9D0C    3 R             LD      A, wAddress          ; Restore the return address
028F 67      3               PUSH    A                    ; Save LSB first
0290 9D0D    3 R             LD      A, wAddress+1        ;
0292 67      3               PUSH    A                    ; Save MSB second
                       ;
0293 64      1               CLR     A                    ; Clear 'A'
0294 9F10    2 R             LD      B, #fStatus          ; Find a free timer
0296                   osSetTimer1:
0296 70      1               IFBIT   fTimer1, [B]         ; Timer 1 free?
0297 06      3               JP      osSetTimer2          ;  No, try timer 2
0298 78      1               SBIT    fTimer1, [B]         ; Capture timer
0299 9F15    2 R             LD      B, #wTimer1CallBack  ;  yes, load and start timer 1
029B 9801    2               LD      A, #1                ;
029D 10      3               JP      osSetTimerStart      ;
029E                   osSetTimer2:
029E 71      1               IFBIT   fTimer2, [B]         ; Timer 2 free?
029F 06      3               JP      osSetTimer3          ;  No, try timer 3
02A0 79      1               SBIT    fTimer2, [B]         ; Capture timer
02A1 9F17    2 R             LD      B, #wTimer2CallBack  ; yes, load and start timer 2
02A39802     2               LD      A, #2                ;
02A5 08      3               JP      osSetTimerStart      ;
02A6                   osSetTimer3:
02A6 72      1               IFBIT   fTimer3, [B]         ; Timer 2 free?
02A7 22FB  > 3               JP      osSetTimerExit       ;  No timers available, sorry...
02A9 7A      1               SBIT    fTimer3, [B]         ; Capture timer
02AA 9F17    2 R             LD      B, #wTimer2CallBack  ;  yes, load and start timer 2
02AC 9802    2               LD      A, #2                ;
                       ;
02AE                   osSetTimerStart:
02AE 67      3               PUSH    A                    ; Save timer handle
02AF A0      1               RC                           ; Check if zero
02B0 9D13    3 R             LD      A, wTimerCallBack    ; Get return address
02B2 BD1480  4 R             ADC     A, wTimerCallBack+1  ; See if all zeros
02B5 88      1               IFC                     ; If there was a carry, good address
02B6 0A      3               JP      osSetTimerAddress    ;
02B7 9900    2               IFNE    A, #0                ; If not zero, also good address
02B9 07      3               JP      osSetTimerAddress    ;
                       ;
```

```
02BA 9D46    3 R          LD      A, aTaskRecs+1 ; Point to TASK 1 LSB (Default callback)
02BC A2      2            X       A, [B+]              ; Save LSB in call back record
02BD 9D47    3 R          LD      A, aTaskRecs+2       ; Now get MSB
02BF A6      1            X       A, [B]               ; Save MSB in call back record
02C0 06      3            JP      osSetTimerPeriod     ; Now go fire it up!
02C1                osSetTimerAddress:
02C1 9D13    3 R          LD      A, wTimerCallBack    ; Get LSB of call back routine
02C3 A2      2            X       A, [B+]              ; Save in LSB of call back record
02C4 9D14    3 R          LD      A, wTimerCallBack+1  ; Get MSB of call back routine
02C6 A6      1            X       A, [B]               ; Save in MSB of call back record
02C7                osSetTimerPeriod:
02C7 8C      3            POP     A                    ; Get timer handle back
02C8 67      3            PUSH    A                    ; Save for later
                    ;
02C9 9FEA    2            LD      B, #TMR1LO           ; Point to timer 1 as default
02CB9202     2            IFEQ    A, #2                ; Timer 2?
02CD 9FC0    2            LD      B, #TMR2LO           ;  Yes, use timer 2
02CF 9203    2            IFEQ    A, #3                ; Timer 3?
02D1 9FB0    2            LD      B, #TMR3LO           ;  Yes, use timer 3
                    ;
02D3 9D11    3 R          LD      A, wPeriod           ; Get LSB of period
02D5 A2      2            X       A, [B+]              ; Save it
02D6 9D12    3 R          LD      A, wPeriod+1         ; Get MSB of period
02D8 A2      2            X       A, [B+]              ; Save it
02D9 9D11    3 R          LD      A, wPeriod           ; Get LSB of period
02DB A2      2            X       A, [B+]              ; Save it
02DC 9D12    3 R          LD      A, wPeriod+1         ; Get MSB of period
02DE A2      2            X       A, [B+]              ; Save it
                    ;
02DF 8C      3            POP     A                ; Check if timer 1 (FIX FOR MEM MAP)
02E0 67      3            PUSH    A                    ;  memory is scrambled for timer 1
02E1 9201    2            IFEQ    A, #1                ; Timer 1?
02E3 9FE6    2            LD      B, #T1RBLO           ;  Yes, fix location
                    ;
02E5 9D11    3 R          LD      A, wPeriod           ; Get LSB of period
02E7 A2      2            X       A, [B+]              ; Save it
02E8 9D12    3 R          LD      A, wPeriod+1         ; Get MSB of period
02EA A6      1            X       A, [B]               ; Save it
                    ;
02EB 8C      3            POP     A                    ; Get timer handle back
02EC 9201    2            IFEQ    A, #1                ; Start timer 1?
02EE BDEE7C  4            SBIT    T1C0, CNTRL          ;  Yes, start it up
02F1 9202    2            IFEQ    A, #2                ; Start timer 2?
02F3 BDC67C  4            SBIT    T2CO, T2CNTRL        ; Yes, start it up
02F6 9203    2            IFEQ    A, #3                ; Start timer 3?
02F8 BDB67C  4            SBIT    T3CO, T3CNTRL        ;  Yes, start it up

02FB                osSetTimerExit:
02FB DF00    3            LD      S, #0           ; Switch data RAM segment back to user
02FD 8E      5            RET                          ; Return

                    ;*********************************************************************
                    ;
                    ;     Routine: osKillTimer
                    ;     Use:     This routine stop a timer and free it.
                    ;
                    ;     Entry:   Accumulator contains the non-zero timer handle
                    ;     Returns: Accumulator is zero if successful, non-zero if failed.
                    ;
                    ;*********************************************************************
02FE                osKillTimer:
02FE DF01    3            LD      S, #1                ; Switch data segment to OS
0300 9201    2            IFEQ    A, #1                ; Kill Timer 1?
0302 09      3            JP      osKillTimer1         ;  Yes, stop it.
0303 9202    2            IFEQ    A, #2                ; Kill Timer 2?
0305 0D      3            JP      osKillTimer2         ;  Yes, stop it.
0306 9203    2            IFEQ    A, #3                ; Kill Timer 3?
```

```
0308 11      3           JP      osKillTimer3         ;  Yes, stop it.
0309 98FF    2           LD      A, #0xFF             ; Failure, incorrect handle.
030B 15      3           JP      osKillTimerExit      ; Return
030C                osKillTimer1:
030C BD1068  4 R         RBIT    fTimer1, fStatus     ; Free timer
030F BDEE6C  4           RBIT    T1C0, CNTRL          ; Stop the timer
0312 0D      3           JP      osKillTimerOK        ;
0313                osKillTimer2:
0313 BD1069  4 R         RBIT    fTimer2, fStatus     ; Free timer
0316 BDC66C  4           RBIT    T2CO, T2CNTRL        ; Stop the timer
0319 06      3           JP      osKillTimerOK        ;
031A                osKillTimer3:
031A BD106A  4 R         RBIT    fTimer3, fStatus     ; Free timer
031D BDB66C  4           RBIT    T3CO, T3CNTRL        ; Stop the timer
0320                osKillTimerOK:
0320 64      1           CLR     A                    ; Report timer dead...
0321                osKillTimerExit:
0321 DF00    3           LD      S, #0                ; Switch data segment back to user
0323 8E      5           RET                          ; Return

                    ;**********************************************************************
                    ;
                    ;     Routine: osSignal
                    ;     Use:     This routine will move a task to the top of the queue.
                    ;
                    ;     Entry:   Accumulator contains the task number to signal next.
                    ;     Returns: Accumulator is zero if successful, non-zero if failed.
                    ;
                    ;*********************************************************************
0324                osSignal:
0324 DF01    3           LD      S, #1                ; Switch to OS RAM Page
0326 BD0A83  4 R         IFGT    A, bTaskNew          ; Is the task number valid?
0329 0A      3           JP      osSignalError        ;  No, don’t start
                    ;
032A 8B      1           DEC     A                    ; Use corrected pointer (p-1)
032B 30A4    5           JSR     osGetTaskAddress ; Get a pointer into the task buffer
032D 70      1           IFBIT   fActive, [B]         ; Check if task running
032E 01      3           JP      osSignalOK           ;  Yes, go move to high priority
032F 04      3           JP      osSignalError        ;  No, Go error out
                    ;
0330                osSignalOK:
0330 9C0B    3 R         X       A, bTaskNext         ; Next task goes to top!
0332 64      1           CLR     A                    ; Clear ’A’ to signal OK
0333 02      3           JP      osSignalEnd          ; End routine
0334                osSignalError:
0334 98FF    2           LD      A, #0xFF             ; Signal error (non-zero)
0336                osSignalEnd:
0336 DF00    3           LD      S, #0                ; Switch back to user RAM
0338 8E      5           RET                          ; Return to calling task

                    ;*********************************************************************
                    ;
                    ;     Routine: osSetCommChannel
                    ;     Use:     This routine will capture a UART to a task
             ;              and setup the baud rate, framing, data bits, and buffers.
                    ;
             ;     Entry:   ’A’ holds requested COM channel (1=UART 1, 2=UART 2, etc.)
                    ;              Data is pushed on the stack as follows:
             ;              * PUSH baud rate (see tables below), data bits, and parity
                    ;              * PUSH LSB of UART RX Call Back routine
                    ;              * PUSH MSB of UART RX Call Back routine
                    ;     Returns: If successful, returns handle to UART (1-n) in ’A’
                    ;              If failure, return zero (0) in ’A’
                    ;
                    ;     Table 1: 76543210 (Setup byte for UART - pushed on stack first)
                    ;              ||||''''- Baud rate:
                    ;              |||'----- Data bits (0=7, 1=8)
                    ;              ||'------ Stop bits (0=1, 1=2)
                    ;              |'------- Parity (0=none, 1=On)
                    ;              '-------- Parity Type(0=Odd, 1=Even)
                    ;
                    ;     Table 2: Baud Rate (bits 0-3)
                    ;
                    ;              0000 = 110   BPS
                    ;              0001 = 134.5 BPS
                    ;              0010 = 150   BPS
                    ;              0011 = 300   BPS
                    ;              0100 = 600   BPS
                    ;              0101 = 1200  BPS
                    ;              0110 = 2400  BPS
                    ;              0111 = 4800  BPS
                    ;              1000 = 7200  BPS
                    ;              1001 = 9600  BPS
                    ;              1010 = 19200 BPS
                    ;              1011 = (Reserved*)
                    ;              1100 = (Reserved*)
                    ;              1101 = (Reserved*)
                    ;              1110 = (Reserved*)
                    ;              1111 = (Reserved*)
                    ;
                    ;              Note * Defaults to 9600 BPS
                    ;
                    ;*********************************************************************
0339                osSetCommChannel:
0339 9DFF    3            LD      A, S                 ; Save old segment
033B DF01    3            LD      S, #1                ; Switch Data RAM segment to OS
033D 9C00    3 R          X       A, aStore+0 ; Save for later to restore (in OS Segment)

033F 8C      3            POP     A                    ; Get MSB of calling routine
0340 9C0D    3 R          X       A, wAddress+1        ; Save for now
0342 8C      3            POP     A                    ; Get LSB of calling routine
0343 9C0C    3 R          X       A, wAddress          ; Save for now
0345 8C      3            POP     A                    ; Get MSB of Call Back routine
0346 9C1F    3 R          X       A, wUARTRxCallBack+1 ; Save for now
0348 8C      3            POP     A                    ; Get LSB of Call Back routine
0349 9C1E    3 R          X       A, wUARTRxCallBack   ; Save for now
034B 8C      3            POP     A                    ; Get MSB of time period
034C 9C1D    3 R          X       A, bUARTSettings     ; Save for now
                    ;
034E 9D0C    3 R          LD      A, wAddress          ; Restore the return address
0350 67      3            PUSH    A                    ; Save LSB first
0351 9D0D    3 R          LD      A, wAddress+1        ;
0353 67      3            PUSH    A                    ; Save MSB second
                    ;
0354 BD1073  4 R          IFBIT   fUART, fStatus       ; is UART already in use?
0357 23D1  > 3            JP      osSetUARTFail        ;  Yes, bail out
                    ;
0359 BD107B  4 R          SBIT    fUART, fStatus       ; Take control of UART
035C 9D1D    3 R          LD      A, bUARTSettings     ; Setup UART
035E 950F    2            AND     A, #0x0F             ; Clear unused bits for now
0360 A0      1            RC                           ; Multiply by 2
0361 A8      1            RLC     A                    ;
0362 9476    2            ADD     A, #LOW(osBaudTable) ; Get table offset
0364 A4      3            LAID                         ; Get first byte
0365 9CBD    3            X       A, BAUD              ; Put into baud rate register
                    ;
0367 9D1D    3 R          LD      A, bUARTSettings     ; Setup UART
0369 950F    2            AND     A, #0x0F             ; Clear unused bits for now
036B A0      1            RC                           ; Multiply by 2
036C A8      1            RLC     A                    ;
036D 9477    2            ADD     A, #LOW(osBaudTable)+1 ; Get table offset+1
036F A4      3            LAID                           ; Get first byte
0370 9760    2            OR      A, #cBaudPrescale      ; Add prescaler value
0372 9CBE    3            X       A, PSR                 ; Put into Prescaler register
```

```
0374 2396  > 3            JP      osSetUART2
0376               osBaudTable:
0376 6903                 .DW     873                       ; 110   BPS
0378 CA02                 .DW     714                       ; 134.5 BPS
037A 8002                 .DW     640                       ; 150   BPS
037C 3F01                 .DW     319                       ; 300   BPS
037E 9F00                 .DW     159                       ; 600   BPS
0380 4F00                 .DW     79                        ; 1200 BPS
0382 2700                 .DW     39                        ; 2400 BPS
0384 1300                 .DW     19                        ; 4800 BPS
0386 0E00                 .DW     14                        ; 7200 BPS
0388 0900                 .DW     9                         ; 9600 BPS
038A 0400                 .DW     4                         ; 19200 BPS
038C 0900                 .DW     9                         ; 9600 BPS (Just in case)
038E 0900                 .DW     9                         ; 9600 BPS (Ditto)
0390 0900                 .DW     9                         ; 9600 BPS (Ditto)
0392 0900                 .DW     9                         ; 9600 BPS (Ditto)
0394 0900                 .DW     9                         ; 9600 BPS (Ditto)
                   ;
0396               osSetUART2:
0396 BCBA08  3            LD      ENU, #B’00001000          ; No parity, 7 data bits
0399BCBB00   3            LD      ENUR, #B’00000000         ; Clear attention mode
039CBCBC20   3            LD      ENUI, #B’00100000         ; 1 stop bit, TDX pin enabled,
                   ;                                        ; Async mode, Int Clks
039F BDD16B  4            RBIT    3, PORTLC                 ; RX pin as input
03A2 BDD17A  4            SBIT    2, PORTLC                 ; TX Pin as output
03A5 9D1D    3 R          LD      A, bUARTSettings          ; Get the settings again
03A7 6010    2            IFBIT   fDataBits, A              ; 7 or 8 data bits?
03A9 BDBA6B  4            RBIT    CHLO, ENU                 ; 8, Set to 8 data bits
03AC 6020    2            IFBIT   fStopBits, A              ; 1 or 2 stop bits
03AE BDBC7F  4            SBIT    STP2, ENUI                ;  2, Set to 2 stop bits
03B1 6040    2            IFBIT   fParitySel, A             ; Parity?
03B3 BDBA7F  4            SBIT    PEN, ENU                  ;  Yes, turn it on
03B6 6080    2            IFBIT   fParityType,A             ; Odd/Even?
03B8 BDBA7D  4            SBIT    PSEL0, ENU                ; Even, set it up.
                   ;
03BB BC4232  3 R          LD      pCOM1TxWrite, #bCOM1TxBuffer ; Setup Tx Write pointer
03BE BC4332  3 R          LD      pCOM1TxRead, #bCOM1TxBuffer ; Setup Tx Read pointer
03C1 BC3020  3 R          LD      pCOM1RxWrite, #bCOM1RxBuffer ; Setup Rx write pointer
03C4 BC3120  3 R          LD      pCOM1RxRead, #bCOM1RxBuffer ; Setup Rx Read pointer
                   ;
03C7 9D00    3 R          LD      A, aStore+0               ; Restore old RAM segment
03C9 9CFF    3            X       A, S                      ;
03CB 9801    2            LD      A, #1               ; Only one UART so return handle=1
03CD BDBC79  4            SBIT    ERI, ENUI                 ; Turn on the receiver interrupt
03D0 05      3            JP      osSetUARTExit             ; Done
03D1               osSetUARTFail:
03D1 9D00    3 R          LD      A, aStore+0               ; Restore old RAM segment
03D3 9CFF    3            X       A, S                      ;
03D5 64      1            CLR     A                         ; Returnbad handle value
03D6               osSetUARTExit:
03D6 8E      5            RET                               ; Return

                   ;**********************************************************************
                   ;
                   ;      Routine: osUARTGetChar
                   ;      Use:    This routine gets one character from the receive queue
                   ;
                   ;      Entry:   ’B’ holds comm channel handle
                   ;      Returns: ’A’ hold received character
                   ;               ’B’ is zero if fail, otherwise handle remains.
                   ;               ’X’ is altered.
                   ;
                   ;**********************************************************************
03D7               osUARTGetChar:
03D7 9DFF    3            LD      A, S                      ; Save old segment
03D9 DF01    3            LD      S, #1                     ; Switch Data RAM segment to OS
```

```
03DB 9C00    3 R          X       A, aStore+0 ; Save for later to restore (in OS Segment)
                      ;
03DD 9DFE    3            LD      A, B                   ; Get comm channel handle
03DF 9201    2            IFEQ    A, #1             ; Check handle to see if valid UART
03E1 01      3            JP      osUARTRxCh1            ;  UART on channel 1
03E2 18      3            JP      osUARTRxFail           ; Not a valid UART
03E3                  osUARTRxCh1:
03E3 9D31    3 R          LD      A, pCOM1RxRead         ; Get the read pointer
03E5 BD3082  4 R          IFEQ    A, pCOM1RxWrite        ; Any data to receive?
03E8 12      3            JP      osUARTRxFail           ;  No, bail...
03E9 9CFC    3            X       A, X                   ; Use ’X’ as pointer into buffer
03EB BA      3            LD      A, [X+]                ; Get the new character
03EC 67      3            PUSH    A                      ; Save on stack for now
03ED 9DFC    3            LD      A, X                   ; Get new pointer value
03EF 9230    2 R          IFEQ    A, #bCOM1RxBuffer+cCOM1RxBufLen; Past end of buffer
03F1 9820    2 R          LD      A, #bCOM1RxBuffer      ; Reload start of buffer
03F3 9C31    3 R          X       A, pCOM1RxRead         ; Save pointer
03F5 9D00    3 R          LD      A, aStore+0            ; Restore old RAM segment
03F7 9CFF    3            X       A, S                   ;
03F9 8C      3            POP     A                      ; Get data back
03FA 05      3            JP      osUARTRxExit           ; Done
03FB                  osUARTRxFail:
03FB 9D00    3 R          LD      A, aStore+0            ; Restore old RAM segment
03FD 9CFF    3            X       A, S                   ;
03FF 5F      1            LD      B, #0                  ; Clear handle
0400                  osUARTRxExit:
0400 8E      5            RET                            ; Done...

                      ;*********************************************************************
                      ;
                      ;      Routine: osUARTSend
           ;      Use:     This routine queues a byte of data for transmission by the UART.
                      ;
                      ;      Entry:   ’A’ holds data byte, ’B’ holds comm channel handle
                      ;      Returns: If successful, returns zero ’A’
                      ;               If failure, returncnon-zero in ’A’
                      ;*********************************************************************
0401                  osUARTSend:
0401 67      3            PUSH    A                      ; Save data
0402 9DFE    3            LD      A, B                   ; Get comm channel handle
0404 9201    2            IFEQ    A, #1             ; Check handle to see if valid UART
0406 02      3            JP     osUARTSendCh1           ;  UART on channel 1
0407 2438  > 3            JP     osUARTSendError         ; Not a valid UART
0409                  osUARTSendCh1:
0409 9DFF    3            LD      A, S              ; Get the current segment register
040B DF01    3            LD      S, #1                  ; Switch to OS segment
040D 9C00    3 R          X       A, aStore+0            ; Save for later on OS page
040F BD4477  4 R          IFBIT   fTxBusy, bCOM1Status   ; Is transmitter busy?
0412 1D      3            JP      osUARTSendBusy         ;  Yes, bail out
0413 9D42    3 R          LD      A, pCOM1TxWrite        ; Get the write pointer to COM1
0415 9CFE    3            X       A, B                   ; Use ’B’ to write the data
0417 8C      3            POP     A                 ; Get the data byte off the stack
0418 A2      2            X       A, [B+]                ; Write into buffer
0419 9DFE    3            LD      A, B                   ; Get pointer back
041B 9242    2 R          IFEQ    A, #bCOM1TxBuffer+cCOM1TxBufLen ;Past end of buffer?
041D 9832    2 R          LD      A, #bCOM1TxBuffer      ; Reload at start of buffer
041F BD4382  4 R          IFEQ    A, pCOM1TxRead         ; Have we overrun the FIFO?
0422 BD447F  4 R          SBIT    fTxBusy, bCOM1Status   ;  Yes, use later for test.
0425 9C42    3 R          X       A, pCOM1TxWrite  ; Save the new COM1 write pointer
0427 BDBC78  4 SBIT  ETI, ENUI  ; Enable transmitter interrupt (start transmitter)
042A 9D00    3 R          LD     A, aStore+0             ; Get old segment pointer back
042C 9CFF    3            X      A, S                    ; Return old segment
042E 64      1            CLR    A                       ; Return with zero (success)
042F 0B      3            JP     osUARTSendExit          ; Done, return
0430                   osUARTSendBusy:
0430 8C        3            POP     A                    ; Adjust Stack
0431 9D00      3 R          LD      A, aStore+0          ; Get old segment pointer back
```

```
0433 9CFF      3            X       A, S                 ; Return old segment
0435 98FE      2            LD      A, #X'FE             ; Return non-zero value (Busy)
0437 03        3            JP      osUARTSendExit       ; Done, return
0438                osUARTSendError:
0438 8C        3            POP     A                    ; Adjust stack pointer
0439 98FF      2            LD      A, #X'FF             ; Return non-zero value (Error)
043B                osUARTSendExit:
043B 8E        5            RET                          ; Return

                    ;*********************************************************************
                    ;
                    ;      Routine: osStart
                    ;      Use:     This routine will start the MTK Operating System (OS)
                    ;
                    ;      Entry:   (None required)
              ;      Returns: Accumulator is zero if successful, non-zero if failed.
                    ;
                    ;*********************************************************************
043C                osStart:
043C DF01      3            LD      S, #1                ; Switch to OS RAM page
043E 9D0A      3 R          LD      A, bTaskNew          ; Check if any tasked loaded.
0440 9200      2            IFEQ    A, #0                ; Any tasks loaded?
0442 2466    > 3            JP      osStartError         ;  No, Return with error
                    ;
0444 30B3      5            JSR     osGetFirstTask       ; Get the first enabled task
0446 9200      2            IFEQ    A, #0                ; Anything to run?
0448 1D        3            JP      osStartError         ;  No, Return with error
0449 9C09      3 R          X       A, bTask             ; Save in currently running task
044B 30B7      5            JSR     osGetNextTask        ; Get the next task number
044D 9200      2            IFEQ    A, #0                ; Anything to run next?
044F 16        3            JP      osStartError         ;  No, Return with error
0450 9C0B      3 R          X       A, bTaskNext         ; Save in next task variable
                    ;
0452 986B      2            LD      A, #LOW(osMain)      ; Setup return Address
0454 67        3            PUSH    A                    ; First LSB
0455 9804      2            LD      A, #HIGH(osMain)     ;
0457 67        3            PUSH    A                    ; Next MSB
                    ;
0458 9D09      3 R          LD      A, bTask           ; Get first task that's enabled
045A 8B        1            DEC     A                    ; Yes, get the address
045B 30A4      5            JSR     osGetTaskAddress     ; Use to get the next task
045D AA        2            LD      A, [B+]              ; Move past flags, Point to LSB
045E AA        2            LD      A, [B+]              ; Get LSB, Point to MSB
045F 67        3            PUSH    A                    ; Put LSB on stack first (FILO)
0460 AE        1            LD      A, [B]               ; Get MSB, Point at LSB
0461 67        3            PUSH    A                    ; Put on stack

0462 BDEF78    4            SBIT    GIE, PSW             ;
0465 02        3            JP      osStartEnd           ; Go begin
0466                osStartError:
0466 98FF      2            LD      A, #0xFF             ; Make non-zero
0468                osStartEnd:
0468 DF00      3            LD      S, #0                ; Switch back to user RAM
046A 8E        5            RET                          ; Start first task (Cool!)

                    ;*********************************************************************
                    ;
                    ;      Routine: osMain
           ;      Use:     This routine is the main loop of the kernal.  All dispatching
               ;               is done from here.  Any error handling is also done here.
                    ;
                    ;      Entry:   (None required)
                    ;      Returns: (Never)
                    ;
                    ;*********************************************************************
046B                osMain:

```

```
1203                     ; *********** DO ERROR CHECKING HERE ***************
1204
1205 046B DF01      3            LD      S, #1               ; Switch to OS data segment
1206 046D 9D0B      3 R          LD      A, bTaskNext     ; Get the pointer to the next task
1207 046F 9200      2            IFEQ    A, #0       ; Something very bad has happened!
1208 0471 2495    > 3            JP      osMainExit      ; Let the user decide how to fix it.
1209 0473 986B      2            LD      A, #LOW(osMain)     ; Put return address on stack
1210 0475 67        3            PUSH    A                   ; First LSB
1211 0476 9804      2            LD      A, #HIGH(osMain)    ;
1212 0478 67        3            PUSH    A                   ; Next MSB
1213                     ;
1214 0479 9D0B      3 R          LD      A, bTaskNext        ; Update current task
1215 047B 9C09      3 R          X       A, bTask            ;
1216 047D 30B7      5            JSR     osGetNextTask       ; Get the next one
1217 047F 9C0B      3 R          X       A, bTaskNext
1218
1219 0481 9D09      3 R          LD      A, bTask            ; Start next task thread
1220 0483 8B        1            DEC     A            ; Adjust task number for table look-up
1221 0484 30A4      5            JSR     osGetTaskAddress    ; Look up task address
1222 0486 70        1            IFBIT   fActive, [B]   ; Check to make sure it’s still active
1223 0487 08        3            JP      osMainSwitch        ;  Yes, still active - go switch
1224 0488 30B7      5            JSR     osGetNextTask   ;  No, find one that is OK to start
1225 048A 9C0B      3 R          X       A, bTaskNext        ; Save it
1226 048C 8C        3            POP     A                   ; Adjust stack
1227 048D 8C        3            POP     A                   ;
1228 048E 246B    > 3            JP      osMain              ; Go test it again
1229 0490                osMainSwitch:
1230 0490 AA        2            LD      A, [B+]         ; Load ’A’ with Flags, point to LSB
1231 0491 AA        2            LD      A, [B+]         ; Load ’A’ with LSB, point to MSB
1232 0492 67        3            PUSH    A                   ; Save LSB
1233 0493 AE        1            LD      A, [B]              ; Load ’A’ with MSB
1234 0494 67        3            PUSH    A                   ; Save MSB
1235 0495                osMainExit:
1236 0495 DF00      3            LD      S, #0        ; Switch to user base RAM data segment
1237 0497 8E        5            RET
1238                     ;       .ENDSECT                    ; Section ends here
1239 0498                        .END    Start
 .sect_CODE_1 . . .  0000 Abs Null ROM

 ADRSLT . . . . . .  00CC Abs Byte
        -4
 ANYCOP . . . . . .  0000 Abs Null
       -64
 ATTN . . . . . . .  0002 Abs Null
        -4
 aStore . . . . . .  0000 Rel Byte SEG
      -107  247  280  299  313  648  650  658  660  678  680  690  697 964 1043 1049
      1069 1086 1091 1115 1129 1135
 aTaskRecs  . . . .  0045 Rel Byte SEG
      -158     356     473     475     801     803
 BAUD . . . . . . .  00BD Abs Byte
        -4     992
 BUSY . . . . . . .  0002 Abs Null
        -4
 bCOM1RxBuffer  . .  0020 Rel Byte SEG
      -138     567     568    1040    1041    1083    1084
 bCOM1Status  . . .  0044 Rel Byte SEG
      -144     551    1116    1126
 bCOM1TxBuffer  . .  0032 Rel Byte SEG
      -141     553     554    1038    1039    1123    1124
 bMaxTaskTime . . .  000E Rel Byte SEG
      -114     719     720     727
 bTask  . . . . . .  0009 Rel Byte SEG
      -110     376     387     461     477    1163    1174    1215   1219
 bTaskNew . . . . .  000A Rel Byte SEG
      -111     248     252     259     274     301     329     385    899    1156
 bTaskNext  . . . .  000B Rel Byte SEG
```

```
     -112     909    1167    1206    1214    1217    1225
bTaskTime  . . . .  000F Rel Byte SEG
     -115     456     484     721     726
bUARTSettings  . .  001D Rel Byte SEG
     -131     975     986     994    1028
C  . . . . . . . .  0006 Abs Null
       -4     622     623     624
CHL1 . . . . . . .  0004 Abs Null
       -4
CHLO . . . . . . .  0003 Abs Null
       -4    1030
CKO  . . . . . . .  0007 Abs Null
       -4
CKX  . . . . . . .  0001 Abs Null
       -4
CMP1OE . . . . . .  0003 Abs Null
       -4
CMP1EN . . . . . .  0001 Abs Null
       -4
CMP1INN  . . . . .  0001 Abs Null
       -4
CMP1INP  . . . . .  0002 Abs Null
       -4
CMP1OUT  . . . . .  0003 Abs Null
       -4
CMP1RD . . . . . .  0002 Abs Null
       -4
CMP2OE . . . . . .  0006 Abs Null
       -4
CMP2EN . . . . . .  0004 Abs Null
       -4
CMP2INN  . . . . .  0004 Abs Null
       -4
CMP2INP  . . . . .  0005 Abs Null
       -4
CMP2OUT  . . . . .  0006 Abs Null
       -4
CMP2RD . . . . . .  0005 Abs Null
       -4
CMPSL  . . . . . .  00B7 Abs Byte
       -4
CNTRL  . . . . . .  00EE Abs Byte
       -4     219     842     873
COP820 . . . . . .  0002 Abs Null
      -64
COP820CJ . . . . .  0005 Abs Null
      -64
COP840 . . . . . .  0003 Abs Null
      -64
COP840CJ . . . . .  0006 Abs Null
      -64
COP8620  . . . . .  0007 Abs Null
      -64
COP8640  . . . . .  0008 Abs Null
      -64
COP8720  . . . . .  0009 Abs Null
      -64
COP8780  . . . . .  000A Abs Null
      -64
COP880 . . . . . .  0004 Abs Null
      -64
COP888BC . . . . .  001B Abs Null
      -64
COP888CF . . . . .  0014 Abs Null
      -64
COP888CG . . . . .  0015 Abs Null
      -64
```

```
COP888CL . . . . .  0016 Abs Null
      -64
COP888CS . . . . .  0017 Abs Null
      -64
COP888EB . . . . .  001C Abs Null
      -64
COP888EG . . . . .  0018 Abs Null
      -64       4
COP888EK . . . . .  0019 Abs Null
      -64
COP888EW . . . . .  001D Abs Null
      -64
COP888FH . . . . .  001E Abs Null
      -64
COP888GD . . . . .  001F Abs Null
      -64
COP888GG . . . . .  0020 Abs Null
      -64
COP888GW . . . . .  0021 Abs Null
      -64
COP888HG . . . . .  0022 Abs Null
      -64
COP888KG . . . . .  0023 Abs Null
      -64
COP8ACC  . . . . .  001A Abs Null
      -64
COP8SAA  . . . . .  0024 Abs Null
      -64
COP8SAB  . . . . .  0025 Abs Null
      -64
COP8SAC  . . . . .  0026 Abs Null
      -64
COP912C  . . . . .  0001 Abs Null
      -64
COP943 . . . . . .  000B Abs Null
      -64
COPCHIP  . . . . .  0018 Abs Null
       -4
ClearRAM . . . . .  0002 Abs Null ROM
     -184
ClearRAMLoop1  . .  0006 Abs Null ROM
     -187     190
ClearRAMLoop2  . .  0011 Abs Null ROM
     -195     198
cBaudPrescale  . .  0060 Abs Null
      -80    1000
cCOM1RxBufLen  . .  0010 Abs Null
      -82     138     567    1083
cCOM1TxBufLen  . .  0010 Abs Null
      -81     141     553    1123
cMaxTasks  . . . .  000A Abs Null
      -78     158     250
cRecSize . . . . .  0003 Abs Null
      -77     158
cSystemSegment . .  0001 Abs Null
      -76
cTopOfRAM  . . . .  006F Abs Null
      -74     182     186
cTopOfSysRAM . . .  007F Abs Null
      -75     194     202     682
DOE  . . . . . . .  0007 Abs Null
       -4
ENAD . . . . . . .  00CB Abs Byte
       -4
ENI  . . . . . . .  0001 Abs Null
       -4
ENTI . . . . . . .  0004 Abs Null
```

```
       -4
ENU  . . . . . . .  00BA Abs Byte
       -4     215    1022    1030    1034    1036
ENUI . . . . . . .  00BC Abs Byte
       -4     217     556    1024    1032    1046    1128
ENUR . . . . . . .  00BB Abs Byte
       -4     216    1023
ERI  . . . . . . .  0001 Abs Null
       -4    1046
ERR  . . . . . . .  0002 Abs Null
       -4
ETDX . . . . . . .  0005 Abs Null
       -4
ETI  . . . . . . .  0000 Abs Null
       -4     556    1128
FE . . . . . . . .  0006 Abs Null
       -4
fActive  . . . . .  0000 Abs Null
     -159     306     334     380     467     904    1222
fDataBits  . . . .  0004 Abs Null
     -132    1029
fParitySel . . . .  0006 Abs Null
     -134    1033
fParityType  . . .  0007 Abs Null
     -135    1035
fRxBusy  . . . . .  0006 Abs Null
     -146
fStatus  . . . . .  0010 Rel Byte SEG
     -117     769     872     876     880     982     985
fStopBits  . . . .  0005 Abs Null
     -133    1031
fTimer1  . . . . .  0000 Abs Null
     -121     771     773     872
fTimer2  . . . . .  0001 Abs Null
     -120     778     780     876
fTimer3  . . . . .  0002 Abs Null
     -119     785     787     880
fTxBusy  . . . . .  0007 Abs Null
     -145     551    1116    1126
fUART  . . . . . .  0003 Abs Null
     -118     982     985
GIE  . . . . . . .  0000 Abs Null
       -4    1183
HC . . . . . . . .  0007 Abs Null
       -4     625     626     627
ICNTRL . . . . . .  00E8 Abs Byte
       -4     222     454     504     591     722     725
IEDG . . . . . . .  0002 Abs Null
       -4
INT  . . . . . . .  0000 Abs Null
       -4
INTR . . . . . . .  0000 Abs Null
       -4
IPND . . . . . . .  0003 Abs Null
       -4     595
Initialize . . . .  **** Rel Null ROM   Ext
      162     225
Interrupt  . . . .  00FF Abs Null ROM
     -431
LPEN . . . . . . .  0006 Abs Null
       -4
MSEL . . . . . . .  0003 Abs Null
       -4
mNone  . . . . . .  0000 Abs Null
      -90
mTaskFailure . . .  000F Abs Null
      -93     479
```

```
mTimer . . . . . .  0001 Abs Null
      -91
mUART  . . . . . .  0002 Abs Null
      -92
mUser  . . . . . .  0010 Abs Null
      -95
osAddTask  . . . .  0048 Abs Null ROM   Pub
      166    -244
osAddTaskEnd . . .  0071 Abs Null ROM
      275    -278
osAddTaskError . .  0070 Abs Null ROM
      251    -276
osBaudTable  . . .  0376 Abs Null ROM
      990     998   -1003
osGetFirstTask . .  00B3 Abs Null ROM
     -371    1160
osGetNextTask  . .  00B7 Abs Null ROM
     -375    1164    1216    1224
osGetNextTaskDone   00CE Abs Null ROM
      395    -398
osGetNextTaskError  00CD Abs Null ROM
      388    -396
osGetNextTaskHit .  00CA Abs Null ROM
      381    -392
osGetNextTaskLoop   00BA Abs Null ROM
      374    -378     390
osGetTaskAddress .  00A4 Abs Null ROM
      261     305     333    -351     379     466     903    1176   1221
osKillTimer  . . .  02FE Abs Null ROM   Pub
      177    -861
osKillTimer1 . . .  030C Abs Null ROM
      864    -871
osKillTimer2 . . .  0313 Abs Null ROM
      866    -875
osKillTimer3 . . .  031A Abs Null ROM
      868    -879
osKillTimerExit  .  0321 Abs Null ROM
      870    -884
osKillTimerOK  . .  0320 Abs Null ROM
      874     878    -882
osMain . . . . . .  046B Abs Null ROM
     1169    1171   -1201    1209    1211    1228
osMainExit . . . .  0495 Abs Null ROM
     1208   -1235
osMainSwitch . . .  0490 Abs Null ROM
     1223   -1229
osPOP  . . . . . .  0239 Abs Null ROM   Pub
      172    -675
osPOPError . . . .  0256 Abs Null ROM
      683    -693
osPOPExit  . . . .  0258 Abs Null ROM
      692    -696
osPUSH . . . . . .  0219 Abs Null ROM   Pub
      171    -644
osSMTT_Exit  . . .  0276 Abs Null ROM
      723    -728
osSMTT_Off . . . .  026D Abs Null ROM
      718    -724
osSetCommChannel .  0339 Abs Null ROM   Pub
      173    -961
osSetMaxTaskTime .  025E Abs Null ROM   Pub
      169    -715
osSetTimer . . . .  0279 Abs Null ROM   Pub
      176    -749
osSetTimer1  . . .  0296 Abs Null ROM
     -770
osSetTimer2  . . .  029E Abs Null ROM
```

```
      772    -777
osSetTimer3  . . .  02A6 Abs Null ROM
      779    -784
osSetTimerAddress   02C1 Abs Null ROM
      797     799    -806
osSetTimerExit . .  02FB Abs Null ROM
      786    -848
osSetTimerPeriod .  02C7 Abs Null ROM
      805    -811
osSetTimerStart  .  02AE Abs Null ROM
      776     783    -791
osSetUART2 . . . .  0396 Abs Null ROM
     1002   -1021
osSetUARTExit  . .  03D6 Abs Null ROM
     1047   -1052
osSetUARTFail  . .  03D1 Abs Null ROM
      983   -1048
osSignal . . . . .  0324 Abs Null ROM   Pub
      170    -897
osSignalEnd  . . .  0336 Abs Null ROM
      911    -914
osSignalError  . .  0334 Abs Null ROM
      900     906    -912
osSignalOK . . . .  0330 Abs Null ROM
      905    -908
osStart  . . . . .  043C Abs Null ROM   Pub
      165   -1154
osStartEnd . . . .  0468 Abs Null ROM
     1184   -1187
osStartError . . .  0466 Abs Null ROM
     1158    1162    1166   -1185
osStartTask  . . .  0078 Abs Null ROM   Pub
      167    -295
osStartTaskEnd . .  008C Abs Null ROM
      308    -311
osStartTaskError .  008A Abs Null ROM
      302    -309
osStopTask . . . .  0093 Abs Null ROM   Pub
      168    -327
osStopTaskEnd  . .  00A1 Abs Null ROM
      336    -339
osStopTaskError  .  009F Abs Null ROM
      330    -337
osUARTGetChar  . .  03D7 Abs Null ROM   Pub
      174   -1066
osUARTRxCh1  . . .  03E3 Abs Null ROM
     1073   -1075
osUARTRxExit . . .  0400 Abs Null ROM
     1089   -1094
osUARTRxFail . . .  03FB Abs Null ROM
     1074    1078   -1090
osUARTSend . . . .  0401 Abs Null ROM   Pub
      175   -1106
osUARTSendBusy . .  0430 Abs Null ROM
     1117   -1133
osUARTSendCh1  . .  0409 Abs Null ROM
     1110   -1112
osUARTSendError  .  0438 Abs Null ROM
     1111   -1139
osUARTSendExit . .  043B Abs Null ROM
     1132    1138   -1142
osUnassigned . . .  00CF Abs Null ROM
      205     206     207     208     209     210    -410
PE . . . . . . . .  0005 Abs Null
       -4
PEN  . . . . . . .  0007 Abs Null
       -4    1034
```

```
PORTCC . . . . . .  00D9 Abs Byte
       -4
PORTCD . . . . . .  00D8 Abs Byte
       -4
PORTCP . . . . . .  00DA Abs Byte
       -4
PORTD  . . . . . .  00DC Abs Byte
       -4
PORTGC . . . . . .  00D5 Abs Byte
       -4
PORTGD . . . . . .  00D4 Abs Byte
       -4
PORTGP . . . . . .  00D6 Abs Byte
       -4
PORTI  . . . . . .  00D7 Abs Byte
       -4
PORTLC . . . . . .  00D1 Abs Byte
       -4    1026    1027
PORTLD . . . . . .  00D0 Abs Byte
       -4
PORTLP . . . . . .  00D2 Abs Byte
       -4
PSEL0  . . . . . .  0005 Abs Null
       -4    1036
PSEL1  . . . . . .  0006 Abs Null
       -4
PSR  . . . . . . .  00BE Abs Byte
       -4    1001
PSW  . . . . . . .  00EF Abs Byte
       -4     223     435     488     595     622     624     625    627    1183
pCOM1RxRead  . . .  0031 Rel Byte SEG
     -140    1041    1076    1085
pCOM1RxWrite . . .  0030 Rel Byte SEG
     -139     562     569    1040    1077
pCOM1TxRead  . . .  0043 Rel Byte SEG
     -143     547     557    1039    1125
pCOM1TxWrite . . .  0042 Rel Byte SEG
     -142     555    1038    1118    1127
pStack . . . . . .  0008 Rel Byte SEG
     -108     202     651     657     681     685     686
RBFL . . . . . . .  0001 Abs Null
       -4
RBIT9  . . . . . .  0003 Abs Null
       -4
RBUF . . . . . . .  00B9 Abs Byte
       -4     214     564
RCVG . . . . . . .  0000 Abs Null
       -4
RDX  . . . . . . .  0003 Abs Null
       -4
Restore  . . . . .  0200 Abs Null ROM
      443  451  469  471  485  491  493  511  513  530  532  558  570 572  592  596
     -617
S  . . . . . . . .  00FF Abs Byte
       -4  185  193  212  245  246  281  297  298  314  328  340  437 455  480  490
      499  510  519  529  538  546  561  578  586  620  646  647  661 676  677  698
      716  729  750  849  862  885  898  915  962  963  1044  1050 1067 1068 1087 1092
     1113    1114    1130    1136    1155    1188    1205    1236
S0 . . . . . . . .  0000 Abs Null
       -4
S1 . . . . . . . .  0001 Abs Null
       -4
SI . . . . . . . .  0006 Abs Null
       -4
SIO  . . . . . . .  00E9 Abs Byte
       -4
SIOR . . . . . . .  00E9 Abs Byte
```

```
       -4
SK . . . . . . . .  0005 Abs Null
       -4
SO . . . . . . . .  0004 Abs Null
       -4
SSEL . . . . . . .  0004 Abs Null
       -4
STP2 . . . . . . .  0007 Abs Null
       -4    1032
STP78  . . . . . .  0006 Abs Null
       -4
ServiceExternal  .  01CF Abs Null ROM
     -594     613
ServiceMicrowire .  01CA Abs Null ROM
     -590     609
ServiceSoftware  .  01C4 Abs Null ROM
     -582     615
ServiceTimerT0 . .  0112 Abs Null ROM
     -453     612
ServiceTimerT0Exit  013C Abs Null ROM
      459    -483
ServiceTimerT1 . .  0143 Abs Null ROM
     -489     505
ServiceTimerT1A  .  0140 Abs Null ROM
     -487     611
ServiceTimerT1B  .  0154 Abs Null ROM
     -503     610
ServiceTimerT2 . .  015B Abs Null ROM
     -509     524
ServiceTimerT2A  .  0158 Abs Null ROM
     -507     605
ServiceTimerT2B  .  016C Abs Null ROM
     -522     604
ServiceTimerT3 . .  0173 Abs Null ROM
     -528     543
ServiceTimerT3A  .  0170 Abs Null ROM
     -526     603
ServiceTimerT3B  .  0184 Abs Null ROM
     -541     602
ServiceUART_RX . .  01A4 Abs Null ROM
     -560     607
ServiceUART_TX . .  0188 Abs Null ROM
     -545     606
ServiceUnused  . .  010A Abs Null ROM
     -441     600     608     614
ServiceWakeup  . .  010D Abs Null ROM
     -445     601
Start  . . . . . .  0000 Abs Null ROM
     -182     463    1239
T0EN . . . . . . .  0004 Abs Null
       -4     722     725
T0PND  . . . . . .  0005 Abs Null
       -4     454
T1A  . . . . . . .  0003 Abs Null
       -4
T1B  . . . . . . .  0002 Abs Null
       -4
T1C0 . . . . . . .  0004 Abs Null
       -4       4     842     873
T1C1 . . . . . . .  0005 Abs Null
       -4       4
T1C2 . . . . . . .  0006 Abs Null
       -4       4
T1C3 . . . . . . .  0007 Abs Null
       -4       4
T1ENB  . . . . . .  0000 Abs Null
       -4
```

```
T1PNDB . . . . . .  0001 Abs Null
       -4     504
T1RAHI . . . . . .  00ED Abs Byte
       -4
T1RALO . . . . . .  00EC Abs Byte
       -4
T1RBHI . . . . . .  00E7 Abs Byte
       -4
T1RBLO . . . . . .  00E6 Abs Byte
       -4     833
T2A  . . . . . . .  0004 Abs Null
       -4
T2B  . . . . . . .  0005 Abs Null
       -4
T2C1 . . . . . . .  0005 Abs Null
       -4
T2C2 . . . . . . .  0006 Abs Null
       -4
T2C3 . . . . . . .  0007 Abs Null
       -4
T2CNTRL  . . . . .  00C6 Abs Byte
       -4     220     508     523     844     877
T2CO . . . . . . .  0004 Abs Null
       -4     844     877
T2ENA  . . . . . .  0002 Abs Null
       -4
T2ENB  . . . . . .  0000 Abs Null
       -4
T2PNDA . . . . . .  0003 Abs Null
       -4     508
T2PNDB . . . . . .  0001 Abs Null
       -4     523
T2RAHI . . . . . .  00C3 Abs Byte
       -4
T2RALO . . . . . .  00C2 Abs Byte
       -4
T2RBHI . . . . . .  00C5 Abs Byte
       -4
T2RBLO . . . . . .  00C4 Abs Byte
       -4
T3A  . . . . . . .  0006 Abs Null
       -4
T3B  . . . . . . .  0007 Abs Null
       -4
T3C1 . . . . . . .  0005 Abs Null
       -4
T3C2 . . . . . . .  0006 Abs Null
       -4
T3C3 . . . . . . .  0007 Abs Null
       -4
T3CNTRL  . . . . .  00B6 Abs Byte
       -4     221     527     542     846     881
T3CO . . . . . . .  0004 Abs Null
       -4     846     881
T3ENA  . . . . . .  0002 Abs Null
       -4
T3ENB  . . . . . .  0000 Abs Null
       -4
T3PNDA . . . . . .  0003 Abs Null
       -4     527
T3PNDB . . . . . .  0001 Abs Null
       -4     542
T3RAHI . . . . . .  00B3 Abs Byte
       -4
T3RALO . . . . . .  00B2 Abs Byte
       -4
T3RBHI . . . . . .  00B5 Abs Byte
```

```
       -4
T3RBLO . . . . . .  00B4 Abs Byte
       -4
TBMT . . . . . . .  0000 Abs Null
       -4
TBUF . . . . . . .  00B8 Abs Byte
       -4     550
TC1  . . . . . . .  0007 Abs Null
       -4
TC2  . . . . . . .  0006 Abs Null
       -4
TC3  . . . . . . .  0005 Abs Null
       -4
TDX  . . . . . . .  0002 Abs Null
       -4
TMR1HI . . . . . .  00EB Abs Byte
       -4
TMR1LO . . . . . .  00EA Abs Byte
       -4     815
TMR2HI . . . . . .  00C1 Abs Byte
       -4
TMR2LO . . . . . .  00C0 Abs Byte
       -4     817
TMR3HI . . . . . .  00B1 Abs Byte
       -4
TMR3LO . . . . . .  00B0 Abs Byte
       -4     819
TPND . . . . . . .  0005 Abs Null
       -4     488
TRUN . . . . . . .  0004 Abs Null
       -4
WDOUT  . . . . . .  0001 Abs Null
       -4
WDSVR  . . . . . .  00C7 Abs Byte
       -4
WEN  . . . . . . .  0002 Abs Null
       -4
WKEDG  . . . . . .  00C8 Abs Byte
       -4
WKEN . . . . . . .  00C9 Abs Byte
       -4
WKPND  . . . . . .  00CA Abs Byte
       -4     450
WPND . . . . . . .  0003 Abs Null
       -4     591
wAddress . . . . .  000C Rel Word SEG
     -113  255  257  270  272  752  754  763  765  967  969  977  979
wIntCallBack . . .  001B Rel Word SEG
     -129
wPeriod  . . . . .  0011 Rel Word SEG
     -123     760     762     821     823     825     827     835    837
wTimer1CallBack  .  0015 Rel Word SEG
     -125     204     495     497     774
wTimer2CallBack  .  0017 Rel Word SEG
     -126     515     517     781     788
wTimer3CallBack  .  0019 Rel Word SEG
     -127     534     536
wTimerCallBack . .  0013 Rel Word SEG
     -124     756     758     794     795     807     809
wUARTRxCallBack  .  001E Rel Word SEG
     -136     574     576     971     973
XBIT9  . . . . . .  0005 Abs Null
       -4
XMTG . . . . . . .  0001 Abs Null
       -4
XRCLK  . . . . . .  0003 Abs Null
       -4
```

```
 XTCLK  . . . . . .  0002 Abs Null
       -4
 **** Errors:    0, Warnings:    0
Checksum:     0xF108
Byte Count:   0x045D (1117)
Input File:   mtk88815.asm
Output File:  mtk88815.obj
Memory Model: Large
Chip:         888EG
```