The remainder of this file will be in a format acceptable to TASM for direct assembly. Note there may be errors in the code, it is not intended that it be cut up and included in students files. It is meant only as an example of addressing modes and instructions. The first example is a prime number finder. The second example is a set of subroutines to maintain a multitasking system.

;==============================================================================
;                Advanced Coding Examples for the Students of AER201S
;==============================================================================

.ORG $E000
SEI ; INITIALIZING THE STACK POINTER
LDX #$FF
TXS
;
LDX #$00
LDY #$00
Delay DEX
BNE Delay
DEY
BNE Delay
;
;==============================================================================
;    Prime Number Finder
;==============================================================================

; This Prime Number Finder uses the sieve method to find the primes up to 255
; and then uses those primes to find the primes up to 65535. Note that this
; is of course not THE most efficient way to find primes but it makes a good
; demonstration.
; It would be neat to stack this code up against a casually written/optimized
; compiled C prime number finder on a raging 386. I have a feeling there will
; be less than a factor of ten difference on execution speed. You may be
; surprised just how fast the 6502 is on simple problems.
;
Test_num = $00 ; Test Number to Eliminate non-primes
Array = $00 ; Base Address for the array of primes
lda #$01
sta $a003
lda #$01
sta $a001 ; Turns on an LED on bit zero of port A of VIA 1
        ; to let you know it has started looking for primes

ldx #$01 ; Initialize the array of numbers
Init_Loop txa
    stx Array,x
    inx
    bne Init_Loop

lda #$02 ; Initialize the Test_num = 2
sta Test_num
lda #$04 ; Put the square of 2 in the accumulator
        ; as the first non-prime

; Start Setting the Multiples of the Test_num to zero
Start_num
Got_Mult tax
    stz Array,x ; Set multiples of Test_num to zero since they
    clc "; are not prime.
    adc Test_num ; Calculate the next multiple
    bcs Next_num ; Until the Multiples are outside the array
    jmp Got_Mult

Next_num inc Test_num ; Go on to the next Test_num
ldx Test_num
cpx #$10 ; Until Test_num => sqrt(largest number)
beq More_Primes
lda Array,x
beq Next_num ; Don't use Test_num if Test_num is not prime
txa
; Got a valid new Test_num, now find its square because all non-primes
; multiples less than its square are eliminated already
dex
clc
Square adc Test_num
dex
bne Square
; OK Got the square of Test_num in the accumulator
; lets start checking
jmp Start_num

; More_Primes
;
; Ok now we have all the primes up to 255 in the memory locations $01-$FF
; Lets repack them more neatly into an array with no spaces to make our
; life easier
ldx #$00 ; .X is a pointer into the loose array
ldy #$01 ; .Y is a pointer into the packed array
Repack inx
beq Done_packing
lda Array,x
beq Repack
sta Array,y
iny
jmp Repack

Prime_Ptr = $F0 ; This is a points into the list of primes greater
; than $FF and less that $10000

Poss_Prime = $F2 ; Possible prime
Temp = $F4 ; A Temporary Number used to find modulus
Shift = $F6 ; Number of Places that .A is shifted
TempArg = $F7 ; A temporary number; argument of modulus

; Done packing
lda #$00 ; Store a $00 at the end of the array of short
sta Array,y ; primes so we know when we have reached the end
lda #$00
sta Prime_ptr ; Set the Prime Pointer (for primes >$FF)
lda #$02 ; pointing into $0200. The found primes will be
sta Prime_ptr+1 ; recorded sequentially from there on.

1da #$01 ; Start with $0101 as the first possible prime
sta Poss_Prime
sta Poss_Prime+1

Next_PP ldy #$02
Next_AP lda Array,y
beq Prime
jsr Mod
beq Next_Poss_prime ; it was a multiple of Array,y
; and therefore not prime
iny
jmp Next_AP

Prime ldx #$00
lda Poss_prime ; Store prime away in the array of primes
sta (Prime_ptr,x)
inx
lda Poss_prime+1
sta (Prime_ptr,x)
clc
da Prime_ptr ; Increment the pointer in the array of primes
adc #$02
sta Prime_ptr
lda Prime_ptr+1
adc #$00
sta Prime_ptr+1

Next_Poss_prime
clc ; Increment Poss_Prime to look at the next
lda Poss_Prime ; number
adc #$01
sta Poss_Prime
lda Poss_Prime+1
adc #$00
sta Poss_Prime+1
bcc Next_PP ; Carry will be set when we reach $10000
lda #$00
sta $a001 ; Turns off the LED after the code finishes
DONE      JMP DONE ; Endless loop at end to halt execution

; Find the Modulus Remainder of Poss_Prime and number in A
; Input Regs: .A Number being divided into the Possible Prime
; Poss_Prime contains the number being tested for primeness
; Output Regs: .A Modulo remainder

Mod       ldx Poss_Prime ; Transfer Poss_Prime to Temp
          stx Temp
          ldx Poss_Prime+1
          stx Temp+1
          ldx #$00 ; Set the bit shifting counter to #$00
          stx Shift

; Compare A to the upper byte of Temp
Compare   sec             ; Compare to see if the .A is greater than
          cmp Temp+1 ; (equal to) the high byte of Temp
          bcs A_Bigger

; If the accumulator is smaller than the upper byte of Temp then shift it
; until it is bigger or it overflows the highest bit
          clc
          rol a
          bcc Not_off_end

; It has overflowed the highest bit, unroll it by one position
          ror a
          sta TempArg
          jmp Start_Mod

; Not overflowed yet, go and compare it to Temp+1 again
Not_off_end inc Shift
          jmp Compare

; If the accumulator is bigger and it has been shifted then unshift by one
; bit
A_Bigger    ldx Shift
          cpx #$00
          sta TempArg
          beq Start_Mod
          clc
          ror a
dec Shift
sta TempArg

; If the accumulator was smaller than the highest byte of Temp it now
; has been shifted to strip off the high bit at least
; If the accumulator was larger than the highest byte then proceed with the
; regular modulus shift and subtracts
;
Start_Mod     lda Temp+1
sec
cmp TempArg
bcc Dont_Subt
;
; Subtract as a stage of division
;
sbc TempArg
sta Temp+1
;
Dont_Subt
;
; We would now like to shift the TempArg relative the Temp
; 1) Shift is greater than zero - accumulator was shifted - unshift it
; 2) Shift Temp - if shift reaches -8 then we are out of Temp and
; what we have left is the modulus --RTS
;
lda Shift
bmi Sh_Temp   ; Case 2
beq Sh_Temp
;
Case 1
clc
ror TempArg
dec Shift
jmp Start_Mod
;
Sh_Temp      cmp #$f8
bne Continue
lda Temp+1       ; This is the Modulus
rts

Continue      dec Shift
clc
rol Temp
rol Temp+1
jmp Start_Mod

; The Multitasking 6502    - See you 6502 do several things at once
This relies on the assumption that there is a source of IRQ's out there that is repetitive and each task is allotted time between each IRQ. Process 1 is started automatically by the RESET signal. Any process can extend its life for a while (if it is doing something important) by setting the SEI and then CLI after the important section.

.ORG $E000

SEI ; INITIALIZING THE STACK POINTER
LDX #$FF
TXS

LDX #$00
LDY #$00
Delay DEX
BNE Delay
DEY
BNE Delay

Each Process has a reserved space in memory starting with process 1 at $0200-$03FF, process 2 at $0400-$05FF. With this model, an 8K RAM can support 15 such processes provided none of the RAM outside zero page and stack is used during the execution of a particular process.

M_box = $F0 ; A Mailbox used to communicate between processes
Com1 = $F8 ; User Communications Channel to other processes
Com2 = $F9
Temp = $FA ; A temporary variable used during SWAPS and SPAWNS
Proc_Ptr = $FB ; Pointer to the reserved space of the current process
Proc = $FC ; Current process number
Proc_N = $FE ; Actual Number for active Processes
Proc_M = $FF ; Maximum Number of Processes that have been concurrent

A Process Record Consists of:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Priority</td>
</tr>
<tr>
<td>01</td>
<td>Priority Counter</td>
</tr>
<tr>
<td>02</td>
<td>Accumulator</td>
</tr>
<tr>
<td>03</td>
<td>X Register</td>
</tr>
<tr>
<td>04</td>
<td>Y Register</td>
</tr>
<tr>
<td>05</td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>10-FF</td>
<td>Zero Page Memory from $00-$EF</td>
</tr>
<tr>
<td>100-1FF</td>
<td>System Stack Space</td>
</tr>
</tbody>
</table>

lda #$01 ; Initialize the start up process as 1
sta Proc
sta Proc_N ; Set the number of processes to 1
sta $0200 ; Set the priority of process 1 to 1
lda #$00
sta $0201 ; Set the priority counter of process 1 to 0
lda #$00
sta Proc_Ptr ; Initialize the process pointer to point to
lda #$02 ; Process 1 reserved space $0200-$03FF
sta Proc_Ptr+1
JMP Start_Code
IRQ Subroutine to Swap Tasks

IRQ_VECT    sta Temp              ; Store .A Temporarily

; If there is only one active process currently then just return

lda Proc_N
cmp #$01
bne Cont_Swap1
lda Temp
rti

; Continue there is more than one Process

Cont_Swap1   tya
pha

; Check process priority counter. If it equals the priority of the process
; then attempt to swap in another process

ldy #$00
lda (Proc_Ptr),y    ; Load Priority Number
beq Swap_In        ; If 'killed' process then just swap in another
iny
inc (Proc_Ptr),y    ; Increment Priority Counter
cmp (Proc_Ptr),y
beq Cont_Swap2

; Not done this Process, Return

pla
tay
lda Temp
rti

; Other Processes available and this one is done:  S W A P   O U T

Cont_Swap2    pla

ldy #$04
sta (Proc_Ptr),y    ; Save .Y
dey
taxa
sta (Proc_Ptr),y    ; Save .X
dey
lda Temp
sta (Proc_Ptr),y    ; Save .A
ldy #$05
tsx
taxa
sta (Proc_Ptr),y    ; Save .SP

; Swap Zero Page ($00-$EF) to (Proc_Ptr + $10-$FF)

ldy #$00
lda #$10
sta Proc_Ptr
Out_Zero    lda $00,y
sta (Proc_Ptr),y
iny
cpy #$f0
bne Out_Zero
;
;   Swap System Stack
;
lda #$00
sta Proc_Ptr
inc Proc_Ptr+1
tsx
taxa
tay
Out_Stack   iny
beq Swap_In
lda $0100,y
sta (Proc_Ptr),y
jmp Out_Stack
;
;
;   Look for the next process to swap in
;
Swap_In    lda Proc           ; Looking for another process to Swap in
cmp Proc_M
bne Not_End
;
;   Go back to Process #1
;
lda #$01
sta Proc
lda #$02
sta Proc_Ptr+1
jmp Check_Proc
;
;   Go to the next Process
;
Not_End     clc
lda Proc_Ptr+1
adc #$02
sta Proc_Ptr+1
inc Proc
;
;   Check this Process if Non-Active, go try another
;
ldy #$00
lda (Proc_Ptr),y
beq Another
;
;   Found an Acceptable Process:  S W A P    I N
;
;
;   Get the Stack Pointer
;
ldy #$05
lda (Proc_Ptr),y  ; Restore .SP
tax
txs
; Swap In Zero Page ($00-$EF) to (Proc_Ptr + $10-$FF)

    ldy #$00
    lda #$10
    sta Proc_Ptr

In_Zero    lda (Proc_Ptr),y
    sta $00,y
    iny
    cpy #$f0
    bne In_Zero

; Swap System Stack

    lda #$00
    sta Proc_Ptr
    inc Proc_Ptr+1
    tsx
    txa
    tay

In_Stack  iny
    beq Restore_Regs
    lda (Proc_Ptr),y
    sta $0100,y
    jmp In_Stack

; Restore all of the system registers

Restore_Regs

    ldy #$00
    sta Proc_Ptr
    dec Proc_Ptr+1
    ldy #$01          ; Set Priority Counter to 0
    sta (Proc_Ptr),y
    iny
    lda (Proc_Ptr),y ; Temporarily store .A
    sta Temp
    iny
    lda (Proc_Ptr),y ; Restore .X
    tax
    iny
    lda (Proc_Ptr),y ; Restore .Y
    tay
    lda Temp          ; Restore .A
    rti

;--------------------- Done the Swap ----------------------

;----------------------------------------------------------

; Spawn a New Process

;----------------------------------------------------------

; PHA    Process PCH
; PHA    Process PCL
; PHA    Process Priority
; JSR    Spawn High
;     Spawn Low

;
Spawn
  lda Proc_Ptr+1 ; Store Current Process Pointer
  sta Temp
  lda Proc ; Store Current Process Number
  pha
  lda #$01 ; Set Process Pointer and Number to 1
  sta Proc
  lda #$02
  sta Proc_Ptr+1

; Free_Check ; See if there is an old process number no longer
  ldy #$00 ; in use
  lda (Proc_Ptr),y
  beq Got_Free
  inc Proc
  clc
  lda Proc_Ptr+1
  adc #$02
  sta Proc_Ptr+1
  lda Proc_M
  sec
  cmp Proc
  bcs Free_Check
  inc Proc_M ; Have to create an extra Process
  inc Proc_N

; Ok we are clear, Create this Process
;
Got_Free
  tsx ; Get the current stack pointer
  txa
  clc
  adc #$05
  tax ; Set x to point at Priority

  ldy #$00
  lda $0100,x ; Transfer Priority to Process Space
  sta (Proc_Ptr),y

  ldy #$05 ; Set .sp = #$FC
  lda #$FC
  sta (Proc_Ptr),y

  ldy #$02 ; Set the accumulator to 1 to indicate: START
  lda #$01 ; to the new process
  sta (Proc_Ptr),y

  inc Proc_Ptr+1 ; To point into stack swap space for this process

  lda #$00 ; Processor Status Register, for this process
  ldy #$FD
  sta (Proc_Ptr),y

  inx
  lda $0100,x ; Load PCL
  iny
  sta (Proc_Ptr),y ; Put into (swapped) Stack

  inx
  lda $0100,x ; Load PCH
sta (Proc_Ptr),y ; Put into (swapped) Stack
;
lda Temp           ; Set Pointer back to original (Spawner) process
sta Proc_Ptr+1
;
lda Proc           ; Take Spawned Process number and put in Temp
sta Temp
;
pla                  ; Restore Spawned Process number
sta Proc
;
pla                  ; Pull 'Spawn' return address from stack
tax
pla
tay
;
pla                  ; Pull Spawn data out of the stack pla pla
pla

;tya                  ; Push the Return Address back to the stack pha txa pha
lda Temp           ; Return Spawned Process Number
rts

;------------------ Done Spawn ------------------
;
;
=============================================================
; Kill a Process  
=============================================================
;
; Input Registers : NONE
; Output Registers: NEVER RETURNS IF KILL IS SUCCESSFUL
;
Kill      lda Proc_N
        cmp #$01            ; Can't Clear Last Process
        bne Ok_More
        rts

Ok_More  ldy #$00            ; OK Kill the Process, put a 0 in Priority
        tya
        sta (Proc_Ptr)
;
dec Proc_N       ; One Less Process
;
lda Proc            ; If we are clearing 'Maximum' Process then
cmp Proc_M       ; then reduce maximum
        beq Reduce_Max
        jmp Swap_In         ; Otherwise Go swap another in
;
Reduce_Max
        dec Proc
dec Proc_M
dec Proc_Ptr+1
dec Proc_Ptr+1
lda (Proc_ptr),y
        beq Reduce_Max
jmp Swap_In
;-------------------- Done Clear a Process ----------------------------
;
;
;=======================================================================
;
; An Example Spawnable Process
;=======================================================================
;
; Input Registers: .A  =  #$00 Means that we just want the address of
;   (JSR Child)                      this process so that the process swapper
;                                           will know where to start.
; ;
; ; (RTI to CHILD1) .A  =  #$01 Means that the process swapper has signalled
; the process to actually start
; ;
Child    jsr Child1
Child1  cmp #$00
   bne Go_For_It
;
; Process was called to get its start up address
;
pla ; Grab Child1 start up address
clc
adc #$01 ; Remember that an RTS return address points at the
tax ; last byte of the JSR statement.
pla ; RTI return addresses point to the first byte of the
adc #$00 ; next instruction to be executed
tay
;
pla ; Save Return Address to program calling Child
sta Temp
pla
sta Proc_Ptr
;
tya ; Push Child1 RTI address
pha
tax
pha
;
lda Proc_Ptr ; This Pushes the calling program's return address
pha ; back into the stack
lda Temp
pha
;
lda #$00 ; Returns Proc_Ptr(low) to #$00 after its use as a
sta Proc_Ptr ; Temporary variable
rts
;
; Spawned Process actually starts:
; Note that PLA's are not required to get rid of the JSR Child1 start up
; address since the RTI address pushed in points to Child1 NOT Child
Go_For_It

Body of the spawned process
;
;
; An Example of a Kill of the present Process
;----------------------------------------------------------
;
{ User Code }

sei
jsr Kill ; This should kill the process unless it is the
    ; only process
cli

; This is the only process
;
{ More user code }

;----------------------------------------------------------

Start of User Code
;----------------------------------------------------------
Start_Code
{ Your first process goes here }
;
;
Example Spawn of Process 'Child'
;
sei ; Prevent swap attempts during process creation  
lda #$00  
jsr Child ; Request Address for Child1
;
lda #Priority  
pha ; Push Priority into the stack
;
jsr Spawn ; Ask the Process Swapper to set 'Child1' up in
    ; the swap schedule
    rol a  
    sta Ptr+1 ; Set pointer to the Child process zero page
    lda #$10 ; reserved area
    sta Ptr
;
; The Spawn call returns the process number. If there is some initial data
; or a pointer that this process would like to pass to 'Child1' then the
; address of its ZERO PAGE reserved data space is pointed to by '(Ptr),y'.
; Once the data has been transferred:
        cli ; Re-enable swap attempts
;
;
;----------------------------------------------------------

Example of Taking full control of execution temporarily
;----------------------------------------------------------
;
sei ; Disable swaps
{ User Code }
cli ; Re-enable swaps
;
;
;----------------------------------------------------------
Example of taking full control by Killing all other processes

; -------- Full control example --------

Ptr = $00
K_Proc = $02

sei ; Disable swaps

lda #$00 ; Set Pointer to $0200
sta Ptr
lda #$02
sta Ptr+1

lda #$01 ; Set Kill Process counter to 1
sta K_Proc

Top lda Proc
cmp K_Proc
beq Don_t_Kill
ldy #$00
tyazsta (Ptr),y

Don_t_Kill
cmp Proc_M
beq Done_Kill
inc Ptr+1
inc Ptr+1
inc K_Proc
jmp Top

Done_Kill
lda #$01
sta Proc_N
lda Proc
sta Proc_M
cli ; Note that this is optional, if we know that there
; are no other processes we could prevent swap decisions
; by not clearing the IRQ mask.

{ More code that will not be swapped out }

; .ORG $FFFC
; .WORD $E000
; .WORD IRQ_VECT
; .END

; ------------------- Done Multitasking example -------------------