```
; *******************************************************
; * Title:    Microprocessor Assignment One             *
; *           Reading/Writing from/to PortA and Port B  *
; * Author:   Thomas Adam                               *
; * Version:  2.0                                       *
; *******************************************************

; This is useful, stops us having to define our own.
$Include 'c:\pemicro\ics08gpz\gpregs.inc'

; The HC08 allows us a lot of lattitude to decide what will be program space
; and program calculation. This is usually situation dependant and so we had
; better ensure that we are able to have plenty of room. Since the HC08 is
; backwards compatible with the HC05, any code we might have designed on the
; HC05 should work on the HC08, but *not* vice-versa.

Ram          EQU  $0040         ; Start at this address
Rom          EQU  $E000         ; This is valid ROM on the GP20 and GP32
Vector       EQU  $FFDC         ; Memory for our vector tables and such.

; This next section needs indenting for some reason. I was previously unaware
; that the ASM file was sensitive to whitespace. This was certainly something
; not mentioned in the lectures or anyother resource I have read.
  org Ram                              ;\
  org Rom                              ;| Reserve memory space.
                                       ;/

; Start program here.

; Use of the MAIN label a good thing
MAIN    sei                          ; disables interrupts
        bset    0,CONFIG1            ; disables the watchdog

        clra                         ; It is usually good practice to clear the
                                     ; accumulator.

        clrx                         ; processing doesn't stop with
                                     ; uninitialized register warning
                                     ; when push A,X on the stack

                                     ; I fell foul of this little indiscrepency
                                     ; intially, until I realised from reading
                                     ; "HC05 to HC08 Optimisation", Motorola that
                                     ; I need to do this to surpress such
                                     ; warnings.

        ; This first section prsented no issues, and was fairly straight
        ; forward. This was not covered in the lectures, and so required me to
        ; do some research as to how it all related. The main 86HC08 PDF
        ; available from the Motorola website (subsequentally mirrored on
        ; Learnwise comes in handy).

        jsr     led          ;->|    ; jump to subroutine. This will read and
                             ;  |    ; write from both portA and portB,
                             ;  |    ; displaying each LED in turn. When this
                             ;  |    ; subroutine ends, the PC (program counter)
                             ;  |    ; will point to the next instruction (*)
                             ;  |    ; below (think OO return status).
                             ;  |    ;
                             ;  |    ; Initially I did have all of this in one
                             ;  |    ; huge listing, but having attended the
                             ;  |    ; lectures, assembly programs follow a
```



```
                             ;  |    ; format of:
                             ;  |    ;
                             ;  |    ; declaration -> initialisation ->
                             ;  |    ; sub-routines
                             ;  |    ;
                             ;  |    ; While this is not an enforced policy, it
                             ;  |    ; is good style
                             ;  |    ;
        cli                  ;<-|    ; * Allow interrupts to happen
        sta     PORTB                ; clear portB pins
        lda     #$FF                 ; nice high value for loop, and for our
                                     ; PORTs.

        ; The following is messy -- very messy. I make no excuses for it. I
        ; had lots of trouble trying to make this neater. I am not sure why
        ; this was....
        ;
        ; This section of code is executed *AFTER* the jsr from the counting
        ; from both PORTs has completed. The design for this, is that the LEDs
        ; (some of them), should flash in a certain sequence, and for a
        ; specific number of iterations. The sequence as defined in the
        ; assignment is:
        ;
        ; green -> yellow -> red -> yellow and green
        ;
        ; These will therefore flash seven times, since my name:
        ; TOM ADAM == seven characters in my name.
        ;
        ; Using port B as the output port, the same rationale applies to the
        ; last problem of counting to the output port, all one needs to do is
        ; to take each port in turn and flash it. This can be achieved by
        ; calling a sub-routine to turn the LED on, in that subroutine loop to
        ; keep it on, and then when rts from subroutine is complete, loop
        ; again with the delay.
        ;
        ; Flowchart would be:
        ;
        ;    +>--1. Turn on the appropriate LED (jsr)
        ;    ^   2. Delay again.
        ;x 7 ^   3. Turn off.
        ;    |<--4. Delay

        jsr fgreen           ;\  Turn on the appropriate LED
        jsr loop1            ;|  Delay
        stx PTB              ;|  Turn off.
        jsr loop1            ;/  Delay

        jsr fgreen           ; repeat as above
        jsr loop1
        stx PTB
        jsr loop1

        jsr fgreen
        jsr loop1
        stx PTB
        jsr loop1

        jsr fgreen
        jsr loop1
        stx PTB
        jsr loop1
```

```
        jsr fgreen
        jsr loop1
        stx PTB
        jsr loop1

        jsr fgreen
        jsr loop1
        stx PTB
        jsr loop1

        jsr fgreen
        jsr loop1
        stx PTB
        jsr loop1

        ; now for yellow

        jsr fyellow
        jsr loop1
        stx PTB
        jsr loop1

        jsr fyellow
        jsr loop1
        stx PTB
        jsr loop1

        jsr fyellow
        jsr loop1
        stx PTB
        jsr loop1

        jsr fyellow
        jsr loop1
        stx PTB
        jsr loop1

        jsr fyellow
        jsr loop1
        stx PTB
        jsr loop1

        jsr fyellow
        jsr loop1
        stx PTB
        jsr loop1

        jsr fyellow
        jsr loop1
        stx PTB
        jsr loop1

        ; Now for Red.
        jsr fred
        jsr loop1
        stx PTB
        jsr loop1

        jsr fred
        jsr loop1
        stx PTB
        jsr loop1
```



```
        jsr fred
        jsr loop1
        stx PTB
        jsr loop1

        jsr fred
        jsr loop1
        stx PTB
        jsr loop1

        jsr fred
        jsr loop1
        stx PTB
        jsr loop1

        jsr fred
        jsr loop1
        stx PTB
        jsr loop1

        jsr fred
        jsr loop1
        stx PTB
        jsr loop1

        ; Flash Both colours.
        jsr fry
        jsr loop1
        stx PTB
        jsr loop1

        jsr fry
        jsr loop1
        stx PTB
        jsr loop1

        jsr fry
        jsr loop1
        stx PTB
        jsr loop1

        jsr fry
        jsr loop1
        stx PTB
        jsr loop1

        jsr fry
        jsr loop1
        stx PTB
        jsr loop1

        jsr fry
        jsr loop1
        stx PTB
        jsr loop1

        jsr fry
        jsr loop1
        stx PTB
        jsr loop1
```

```
        ; First task, is to be able to read and write from both ports. My
        ; initial theory behind this is port B as the output port. This is
        ; also the same as DDRB (DDR is "Direct Data Read" and allows for
        ; direct access to the port via the main microcontroller). It is
        ; divided into two, eight bits thus:
        ;
        ; 128 64 32 16  8  4  2  1
        ; +-----------------------+
        ; |  |  |  |  |  |  |  |  |
        ; +-----------------------+
        ;
        ; Addressing in this case though is done in hexadecimal, and so really
        ; the values need are:
        ;
        ;  $80 $40 $20 $10 $8  $4  $2  $1
        ; +-------------------------------+
        ; |   |   |   |   |   |   |   |   |
        ; +-------------------------------+
        ;
        ; Since there are two ports, each with two sections, the port's
        ; ability to act as both I/O port is defined by the DDR being set to
        ; all 1's for output (write) and all 0's for input (read).
        ;
        ; The following statement then, sets Port B as thus:
        ;
        ;
        ;                         $F
        ;                 ---------------
        ;  $80 $40 $20 $10 $8  $4  $2  $1
        ; +-------------------------------+
        ; | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
        ; +-------------------------------+
        ;
        ; The "$F" above is defined for the sequence that the LEDs will light
        ; up in. The sequence (spread over the two portd is):
        ;
        ; R -> G -> R and G -> A -> R and A and Y and G -> R -> G -> A -> R ->
        ; R and A and G and R and R and A and G and R
        ;
        ;
        ; Recall that the previous statement "lda #$FF" will give us eight
        ; 1's, which are then stored in DDRB (Port B).

        sta     DDRB        ;set Port B as output

        ;
        ; This now starts the main 'led' subroutine which will light each led
        ; in turn, on both ports.

led     lda     #$FF        ; store eight 1's into accumulator A
        sta     DDRB        ; Send what we have in accumulator A to the output
                            ; port.
                            ;
        ldx     #$FF        ; Set the Index Register to the same value.
        stx     PTB         ; Keep PTB to 0
        sta     DDRA        ; Set DDRA to Output Port.
                            ;
                            ; The GP32 board we are using has two ports. A and
                            ; B. Both of which can be programmed independantly
                            ; of eachother. The formt is thus:
                            ;
                            ; +--------------------+--------------------+
```



```
                            ; |        Port B      |        Port A      |
                            ; |--------------------|--------------------|
                            ; |                    |      <---- $F ---->|
                            ; |$80 $40 $20 $10     |  $8  $4  $2  $1    |
                            ; | .R  .A  .G  .R     |  .R  .A  .G  .R    |
                            ; +--------------------+--------------------+
                            ;
                            ; Both these ports have mappings assigned to them.
                            ; While it is not essential, what I ought to have
                            ; done is define some EQUATES (EQU) values for:
                            ; PORTA, PORTB, DDRA, DDRB. This would certainly
                            ; have made things clearer. But, these are
                            ; inherently present anyway, so we do not need to
                            ; define them.

        ; and so we start. The design behind this is that we'll go through
        ; each LED in turn, lighting it, before turning it off and moving to
        ; the next one on the board. This is essentially 'counting', or so I
        ; assume. It is not clear from the assignment.
        ;
        ; The way the LEDs are turned on and off is a repeating pattern. Since
        ; the way the instructions are processed on the board is quick, one
        ; has to use a sizeable delay to make it appear to turn on and them
        ; off.

        ; The Index Register is used to communicate with the port. The flow
        ; chart the applies in each case is:
        ;
        ; |-->1. Load in IR the value of the PORT needing to be manipulated
        ; ^   2. Store the IR value in PTB
        ; |   3. Turn on the LED (defined from step 1.) by sending it a '1'
        ; +-<-4. Loop (jsr, and then go back to step 1).

        ldx     #$01        ;\  load into the IR the value of the current LED
        stx     PTB         ; |  store that value in PTB (see EQU comment
                            ; |  above).
        ldx     #$FF        ; |  Load into the IR a 1 (this will turn it on)
        jsr     loop1       ;/  Keep it on for a while (40 milliseconds?)

        ldx     #$02        ;\  load into the IR the value of the current LED
        stx     PTB         ; |  store that value in PTB (see EQU comment
                            ; |  above)
        ldx     #$FF        ; |  Load into the IR a 1 (this will turn it on)
        jsr     loop1       ;/  Keep it on for a while (40 milliseconds?)

        ldx     #$03        ; etc, etc.
        stx     PTB
        ldx     #$FF
        jsr     loop1

        ldx     #$04
        stx     PTB
        ldx     #$FF
        jsr     loop1

        ldx     #$08
        stx     PTB
        lda     #$FF
        jsr     loop1

        ldx     #$0F      ; Here, turn on all LEDs for the PORT.
        stx     PTB
```

```
        ldx     #$FF
        jsr     loop1

        ldx     #$10      ; Continue.
        stx     PTB
        ldx     #$FF
        jsr     loop1

        ldx     #$20
        stx     PTB
        ldx     #$FF
        jsr     loop1

        ldx     #$40
        stx     PTB
        ldx     #$FF
        jsr     loop1

        ldx     #$80
        stx     PTB
        ldx     #$FF
        jsr     loop1

        ldx     #$FF
        stx     PTB
        rts
;*******************************************************************************
;

; Start of flashing light sequence subs.......

; Same ideas as before. This flashes green.

fgreen  ldx     #$02      ;\  Load the correct LED
        stx     PTB       ;|  Store the value of IR in PTB
        jsr     loop1     ;|  Delay
        rts               ;/  return from this subroutine.

; Yellow.

fyellow ldx     #$04      ;\  Load the correct LED
        stx     PTB       ; |  Store the value of the IR in PTB
        jsr     loop1     ; |  Delay.
        rts               ;/  Return from this subroutine.

; Red
fred    ldx     #$01      ; etc....
        stx     PTB
        jsr     loop1
        rts

; Yello and Green
fry     ldx     #$06      ; $04 + $02 = $6
        stx     PTB
        jsr     loop1
        rts

; This loop acts as an effective delay to keep the LED on. This was achieved
; by creating an inner and outer loop, the inner loop, dictated by the outer
; loop. This loops for approximately 40 milliseconds, which visually works,
; even if it is slow in simulation.
```



```
loop1   deca                    ; a = a - 1
        bne     loop1           ; while != 0
        decx                    ; wait for a while.
        bne     loop1           ; Inner/Outer loop allows for greater delay.
        rts                     ; Return from subroutine.

;*************************************************************
;* Vectors - Timer Interrupt Service Routine.                *
;*           after a RESET.                                  *
;*************************************************************

; These act as our interrupts to the LED port.
   org  Vector

        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main
        dw      main

; ****************************************************************************
; Personal Development
;
; I have found this assignment to be both interesting and challenging. I have
; found it very difficult at times to grasp certain concepts, particularly
; with how the microprocessor is made up, and how each individual part
; inter-relates, etc. This required a lot of reading around the subject.
;
; Microprocessors is one of those subjects that can be split into two easy
; parts, numbering systems, and actual microprocessors. I found the numbering
; systems to be of particular use, since it has many applications besides being
; useful for microprocessors. The ability and need to understand numbering
; systems was made clear, through this assignment. If we had not been taught it,
; I would say that we would not have been able to do the assignment.
;
; I did not find too much difficulty with numbering systems, other than
; division and multiplication via binary. It took me a number of attempts before
; I was able to fully understand it. I again, had to do more research into how
; one calculates the sums. This research is beneficial not only in seeking an
; alternate method of working to solve a problem, but it also means that one
; often comes across other related material to do with assembly programming in
; general which is often interesting. This helped me a lot in being able to do
; this assignment.
;
; The assembly programming compliments the work learnt from numbering systems
; well, and it helps to perhaps 'visualise' this via the programming. I found a
; lot of the programming difficult, and certainly I did not get anywhere near to
; programming anything on the LCD display. This is down to lack of
; understanding, and lack of demonstrations, primarily from the tutorial. Not
```

```
; once was it mentioned to us about how to implement such a feature, and while
; I did research into the situation, even by application of my research I was
; unable to successfully program to the LCD display.
;
; As a general improvement I would recommend that the time spent doing the
; numbering systems during tutorials is reduced. Spending approximately seven
; to eight weeks on them is just far too long. It is the programming of the
; microcontroller which is the hardest to understand, and this should be
; addressed where possible. Certainly, while self-learning is a nice tool to
; use, it is not a means of learning throughout the unit. I found a distinct
; lack of examples and demonsrations in tutorials to suppliment the material
; being taught in the lectures to suppliment our learning. These kinds of
; activities are not best left to individual/group learning in this instance.
;
; If I had realised just how challenging the programming section of the unit
; was, I would have spent much more time, getting the basics correct so that I
; have more of the fundamental building blocks to then look further at the LCD
; issue.
;
; Despite some of the criticisms mentioned, this assignment has made me
; realise just what it is possible to achieve. Looking to the future, this
; assignment has made me realise that research is a fundamental learning aid and
; even essential when faced with a problem. It will certainly enable me to be
; more effective with any research that I will have to. This assignment has
; also taught me the useful and resourcefulness of using my peers as a means of
; learning and information. I have found that working in pairs or small groups
; often generates better ideas than I would have been able to on my own.


;*****************************************************************************
;* Originalsource is (C)opyright 1998,and from P&E Microcomputer Systems, Inc.
;* at http://www.pemicro.com.
```