PICAXE-28X1 (OCR / AQA ASSEMBLER)

Programming PICAXE-28X1 with OCR / AQA Assembler Code:

Many schools teaching OCR and AQA A-Level Electronics have asked if it is possible to program their existing PICAXE chips using the 'generic assembler code' specified on these courses. Revolution Education have therefore developed a special free compiler for this purpose. Assembler programs can also be simulated on screen before download to the real chip.

Instructions to enable the OCR/AQA compiler in the free PICAXE Programming Editor software (Windows):

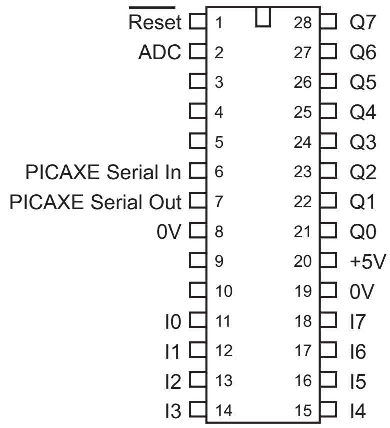
1) Ensure you are using version 5.2.10 or later (Help>About menu). Update is free from www.picaxe.co.uk
2) From the View>Options>Mode menu select the PICAXE-28X1 chip type
3) Click the 'Advanced' button that appears beside the chip type
4) Enable the OCR or AQA option as appropriate.

The special PICAXE-28X1 compiler simply 'adds' the assembler commands to the normal PICAXE compiler, so BASIC programming (and test features such as 'debug') can still be used on assembler enabled computers.

Please see the following sections for notes on both the OCR and AQA command sets.

OCR Assembler Notes:

PICAXE-28X1 (OCR)



Please see the PICAXE manual for the normal PICAXE download circuit, which is not changed for OCR use. Please note that, as with most commercial compilers, hex numbers should be preceded with 0x or $ (see above). Registers s0 to s7 are predefined for immediate use (PICAXE b0 to b7), as are the i/o ports Q (PICAXE outpins) and I (PICAXE pins).
Note that the PICAXE chip does not operate at the same speed as a PIC programmed in raw assembler code. However PICAXE is still a very convenient and low cost method to teach the OCR A2 requirements.

All the OCR defined assembler commands listed overleaf are included. In addition these 3 subroutines are predefined:

```
readtable    - copies the byte in the lookup table pointed at by
               S7 into S0. The lookup table is labelled table: when
               S7=0 the first byte from the table is returned in S0
wait1ms      - wait 1 ms before returning
readadc      - returns a byte in S0 proportional to the voltage at ADC (ADC0)
```

The following extra subroutines are also predefined for extension work (not in specification):

```
wait10ms     - wait 10 ms before returning
wait100ms    - wait 100 ms before returning
wait1000ms   - wait 1000 ms before returning
readadc0     - returns a byte in S0 proportional to the voltage at ADC0.
readadc1     - returns a byte in S1 proportional to the voltage at ADC1.
readadc2     - returns a byte in S2 proportional to the voltage at ADC2.
readadc3     - returns a byte in S3 proportional to the voltage at ADC3.
```

To preload the table for the 'rcall readtable' the normal PICAXE 'table' command is used e.g.

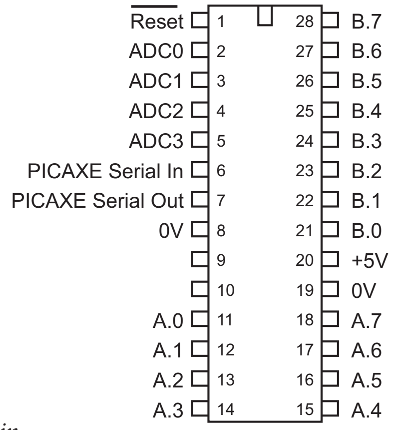
```
table                 (0x08,0x10,0x20,0x40,0x20,0x10)
start:     movi       s7,0x00
           movi       s5,0x06
back1:     rcall      readtable
           out        Q,s0
           movi       s3,0xFF
back0:     rcall      wait1ms
           dec        s3
           jnz        back0
           inc        s7
           mov        s6,s7
           sub        s6,s5
           jnz        back1
           jp         start
```

OCR Instruction Set:

Assembler Command	Function
MOVI Sd,n	Copy the byte n into register Sd
MOV Sd,Ss	Copy the byte from As to Sd
ADD Sd,Ss	Add the byte in Ss to the byte in Sd and store the result in Sd
SUB Sd,Ss	Subtract the byte in Ss from the byte in Sd and store the resultin Sd
AND Sd,Ss	Logical AND the byte in Ss with the byte in Sd and store the result in Sd
EOR Sd,Ss	Logical EOR the byte in Ss with the byte in Sd and store the result in Sd
INC Sd	Add 1 to Sd
DEC Sd	Subtract 1 from Sd
IN Sd,I	Copy the byte at the input port into Sd
OUT Q,Ss	Copy the byte in Ss to the output port
JP e	Jump to label e
JZ e	Jump to label e if the result of the last command was zero
JNZ e	Jump to label e if the result of the last command was not zero
RCALL s	Push the program counter onto the stack to store the return address and then jump to label s
RET	Pop the program counter from the stack to return to the place the subroutine was called from
SHL Sd	Shift the byte in Sd one bit left putting a 0 into the lsb
SHR Sd	Shift the byte in Sd one bit right putting a 0 into the msb

AQA Assembler Notes:

PICAXE-28X1 (AQA)



Please see the PICAXE manual for the normal PICAXE download circuit, which is not changed for AQA use. Please note that, as with most commercial compilers, hex numbers should be preceded with 0x or $
The registers listed below are predefined.

```
Hardware ports:              PORTA, PORTB, TRISA, TRISB
General purpose registers:   B0, B1, B2 etc. to B27
Special function registers:  SR, PRE, TMR

SR Bits are:       0     carry flag (C)     (use mask 'ANDW 0x01')
                   1     TMR overflow       (use mask 'ANDW 0x02')
                   2     zero flag (Z)      (use mask 'ANDW 0x04')
```

Note that the PICAXE chip does not operate at the same speed as a PIC programmed in raw assembler code. However PICAXE is still a very convenient and low cost method to teach the AQA A2 requirements.

All the AQA defined assembler commands listed below are included.
The following extra subroutines are also predefined for practical extension work (note these are not part of the specification and so must not be used in exams):

```
wait1ms      - wait 1 ms before returning
wait10ms     - wait 10 ms before returning
wait100ms    - wait 100 ms before returning
wait1000ms   - wait 1000 ms before returning
readadc0     - returns a byte in b0 proportional to the voltage at ADC0
readadc1     - returns a byte in b1 proportional to the voltage at ADC1
readadc2     - returns a byte in b2 proportional to the voltage at ADC2
readadc3     - returns a byte in b3 proportional to the voltage at ADC3
```

AQA Instruction Set:

Command		Description		Flags altered
NOP	none	No operation		
CALL	K	Call Subroutine		
RET	none	Return from Subroutine		
INC	R	Increments the contents of R	(R) <= (R) + 1	Z
DEC	R	Decrements the contents of R	(R) <= (R) - 1	Z
ADDW	K	Add K to W	W <= W + K	Z, C
ANDW	K	AND K with W	W <= W & K	Z
SUBW	K	Subtract K from W	W <= W - K	Z, C
ORW	K	OR K and W	W <= W | K	Z
XORW	K	XOR K and W	W <= W ^ K	Z
JMP	K	Jump to K (GOTO)	PC <= K	
JPZ	K	Jump to K on zero	PC <= K if Z=1	
JPC	K	Jump to K on carry	PC <= K if C=1	
MOVWR	R	Move W to R	(R) <= W	Z
MOVW	K	Move K to W	W <= K	Z
MOVRW	R	Move R to W	W <= (R)	Z

Also included, not part of official specification:

JPNZ	K	Jump to K on not zero	PC <= K if Z=0
JPNC	K	Jump to K on not carry	PC <= K if C=0

AQA Additional notes:

1) NOP timing loops can be simulated on screen but won't time correctly on the PICAXE system, but this is not a huge issue as students often use a standard predefined subroutine like 'wait1ms' to cause a longer delay when experimenting (note this is not allowed in the exam unless stated in the question). More complex timer delays using TMR and PRE do work (see examples overleaf). It has been assumed that loading a value into TMR resets (clears) the timer overflow bit in SR.

2) TRIS does not work on portB on the PICAXE-28X1 as these pins have a fixed output layout in the PICAXE-28X1 system. However TRISA is fully functional and gives 8 full bidirectional pins, and so is recommended as the default port to use. PORTB/TRISB are still available, but can only be used as all outputs (ie the value passed to TRISB is effectively ignored and left at 0x00). Therefore a simple workaround is to simply state to students that you must always use TRISB with value of 0x00 on the PICAXE system! The analogue inputs (ADC0-3) can be used with the predefined 'call readadcX' subroutines if desired for extension exercises.

3) The AQA example notes gives multiple examples of direct writes to register address numbers and jumps to direct memory locations e.g.

```
MOVWR  0xA0          ; Move the contents of W into 0xA0
INC 0xA0             ; Increment the register
```

This does demonstrate to students how register addresses work but is generally not used by commercial programmers - it is extremely difficult to remember / understand multiple register address numbers and is just asking for bugs in the generated code! In 'commercial assembler' the register would therefore normally be 'named' to an easily understandable name, so the code becomes more readable.

```
#DEFINE my_counter 0xA0
MOVWR  my_counter          ; Move the contents of W into 0xA0
INC  my_counter            ; Increment the register
```

This is exactly the same, but much easier to understand and program.

In 'PICAXE AQA assembler' direct register numbers are not available and so the same system of register re-naming is achieved using the symbol command and the pre-defined general purpose registers b0-b27

```
symbol my_counter = b1     ; define a user register
MOVWR  my_counter          ; move the contents of w into my_counter
INC  my_counter            ; Increment the register
```

4) JPNZ and JPNC commands are supported as well as JPC and JPZ in the PICAXE version. However it should be remembered that these are not part of the official AQA specification and so must not be used in exams.

Example 1 - Time Delay using predefined routines to flash all outputs on PORTB

```
; Flash the output pins at a 0.5Hz rate. The outputs
; will be on for one second and off for one second.

Init: movw   0
      movwr PORTB        ; All PORTB off. Set PORTB before TRISB so pins are in
                         ; known condition before they are converted to outputs
      movwr TRISB        ; All PORTB as outputs

Main: call   wait1000ms  ; wait 1 second (1 second = 1000 ms)
      movrw  PORTB       ; Read outputs
      xorw   0xFF        ; xor'ing each bit inverts it
      movwr  PORTB       ; Set outputs
      jmp    Main        ; back to start
```

Example 2 - Time Delay using TMR/PRE to flash all outputs on PORTB

```
; Flash the output pins at a 0.5Hz rate. The outputs
; will be on for one second and off for one second.
; One second delay is too long, so it is actually made up of 20 lots of 50ms.

symbol mycounter = b0    ; define a register with an easy to use name

Init: movw   00
      movwr PORTB        ; All PORTB off
      movwr TRISB        ; All PORTB as outputs

Strt: movw   20          ; Loop 20 x 50ms = 1000ms
      movwr mycounter    ; save 20 in mycounter

Main: movw   200         ; 200 x 250us = 50ms delay
      movwr PRE
      movw   250
      movwr TMR
; (loading TMR automatically clears the timer overflow bit in status SR)

Lp:   movrw SR           ; Check the Status Register
                         ; Status will be  %xxxxx0x if timer still active
                         ;              or %xxxxx1x if timer expired
      andw   2           ; now mask off the timer expired bit (2 = %00000010 in binary)
                         ; W result will now be 0 if timer still active,   W = 0 so Z bit = 1
                         ;                      or 2 if timer expired,      W = 2 so Z bit = 0
      jpz    Lp          ; If Z set (Z=1) then timer is still active so loop back up

      dec    mycounter    ; Decrement the value of mycounter
                         ; Z bit now = 1 if the decrement result is 0
      jpz    Tog         ; Looped 20 times? if Z bit is 1 we have finished and so goto Tog
      jmp    Main        ; Z bit is 0 so mycounter is not 0, so not done 20 yet so loop back up

Tog:  movrw PORTB        ; Read outputs
      xorw   0xFF        ; xor'ing each bit inverts it
      movwr  PORTB       ; Set outputs

      jmp    Strt        ; Start another 1 second loop
```

Example 3 - Demonstrate use of C carry bit

```
; Find the largest number that when 123 is added to
; it does not exceed 255. The result is displayed on
; the Port B output pins as a binary pattern.

symbol mynumber = b1     ; define a register with an easy to use name

Init: movw   0
      movwr PORTB        ; All PORTB off
      movwr TRISB        ; All PORTB as outputs
      movwr mynumber     ; also reset mynumber to 0

Test: movrw mynumber     ; Get the number and ...
      addw   123         ; ... add 123 to it
      jpc   Got          ; Overflowed if C=1, so result is greater than 255

      inc    mynumber    ; Not overflowed so try the next number
      jmp    Test

Got:  dec    mynumber    ; This number caused overflow so  ...
                         ; ... result of experiment is actually one less
      movrw mynumber     ; Get correct result
      movwr PORTB        ; Put the result to the output pins

; **** optionally use the PICAXE debug command to show mycounter (b1) value
;     debug
; ****

Done: jmp    Done        ; Finished - ever lasting loop
```