Application Note

ZiLOG eZ8 CPU Performance Benchmarking

AN012501-1202

ZiLOG Worldwide Headquarters • 532 Race Street • San Jose, CA 95126-3432
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.zilog.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
532 Race Street
San Jose, CA 95126-3432
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Table of Contents

List of Tables

Introduction

The eZ8 CPU is the central processor unit of ZiLOG's new Z8 Encore family of microcontrollers. It is designed to address continuing demand for faster and more code-efficient microcontrollers. The eZ8 CPU supports a superset of the original Z8® instruction set and provides an upgrade path that includes compatibility and performance for Z8 based designs.

This document evaluates the eZ8 CPU instruction set performance against the Motorola CPU08 using assembly routines that are commonly found in micro-controller applications. On-chip peripheral features and their performance are not considered here. The focus is to evaluate the CPU's instruction set, programming model efficiency, and execution speed. This document is a follow-up of the Application Note "ZiLOG's eZ8 CPU versus Motorola's CPU08 – A Comparison Study".

References

1. ZiLOG eZ8 CPU User Manual, UM0128
2. ZiLOG Z8 Encore! Microcontrollers with Flash and 10-bit A/D Product Specifications, PS0176
3. ZiLOG eZ8 CPU versus Motorola's CPU08 – A Comparison Study Application Note AN0123
4. Motorola CPU08 Reference Manual, CPU08RM/AD, Rev. 3, 2/2001
5. Motorola MC68HC908AB32 Microcontroller Technical Data, Rev. 1.0
6. COP8 Instruction Set Performance Evaluation, National Semiconductor Application Note 1042
7. A Comparison of 8-bit Microcontrollers, Microchip Application Note AN520

Performance Benchmarks Overview

We have used general-purpose commonly used operations and routines to compare the performance of the CPUs. These are not made up to highlight any one instruction set. The benchmarks are representative of typical micro-controller applications and used in industry literature [6,7] to compare core performances. A total of 11 benchmarks are used. These benchmarks exercise data movement, arithmetic operations, I/O manipulation and time keeping capabilities of the CPU.

Table 1 below provides an overview of the benchmarks. The benchmarks are coded in assembly and are optimized for speed. This is so especially in the case of the eZ8 CPU as it allows efficient unrolling of loops. Loop unrolling reduces instruction cycle count at the price of a slightly bigger code size.

Table 1. Benchmarks Overview

Routine	Description
Packing BCD	This benchmark takes two bytes in RAM or registers, each containing a BCD digit in the lower nibble and create a packed BCD data byte, which is stored back in the register or RAM location holding the low BCD digit.
Loop Control	This benchmark is a simple loop control where a register containing a loop count is decremented, tested for zero, and if not zero, then branched back to the beginning of the loop.
Bit Test & Branch	This benchmark tests a single bit in a register or a RAM location and makes a conditional branch. We assume that the most significant bit is tested and a branch is to be taken if the bit is set.
Shifting out 8-bit data & clock	This benchmark generates data and clock under program control by toggling two output pins.
Five Byte Block Move	This benchmark moves only a block of five data bytes from a specific location to a specific destination location.
Software Timer	This benchmark implements a 10ms time delay loop subroutine.
Four Byte Binary Addition	This benchmark adds two four byte binary numbers and replaces the first operand with the result. This emulates an adding machine addition, where A + B replaces A. The benchmark is programmed as a subroutine, with the carry flag indicating an overflow.
Four Byte Packed BCD Subtraction	This benchmark adds two eight digit (four bytes each) packed BCD numbers and replaces the first operand with the result. This emulates an adding machine addition, where A - B replaces A. The benchmark is programmed as a subroutine, with the carry flag being used to indicate a positive or negative result. The BCD decimal-adjust (da) instruction is used following the subtraction to achieve the correct BCD result.
Three Byte Table Search	This benchmark searches a 200-byte table resident in program memory for a three-byte character string, which may be resident anywhere in the lookup table (not necessarily on three byte boundaries). The status of the carry bit indicates the success or failure of the search. The benchmark is programmed as a subroutine.
Input / Output Manipulation	This benchmark compares two 8-bit I/O ports, P1 and P2. If they are equal, a nine is output as the least significant digit (lower nibble) of a third port P3. If port P1 is greater than port P2, then port P2 is output on port P1. If port P1 is less than port P2, then the most significant digit of Port P1 is copied to the least significant digit of Port P3.

Table 1. Benchmarks Overview (Continued)

Routine	Description
Switch Activated Two-Second Delay	This benchmark samples a switch input to activate a two-second output to turn on an LED. This switch is debounced with a 10ms delay on both opening and closing. Once activated, the switch turns on an LED output for two seconds and turns it off, regardless of whether or not the switch is still activated. Once the switch is turned off, the procedure is repeated. Both the switch input and the LED output are low true.

Results Summary

Table 2 below summarizes the performance benchmarking results. For each benchmark, the total instruction cycles, execution time and code size are presented. The execution time is the product of the total instruction cycles and instruction cycle time. The CPU08 executes instructions at 8MHz bus frequency, while the eZ8 CPU executes instructions at 20MHz.

Table 2. Benchmark Results

	Benchmarks (Optimized for speed)	ZiLOG eZ8 CPU (Instruction Cycle Time = 0.05µs)	Motorola CPU08 (Instruction Cycle Time = 0.125µs)	eZ8 CPU Execution Speed Ratio[1]	eZ8 CPU Code Efficiency Ratio[2]
		Execution cycles/ Time and Code Size	Execution cycles/ Time and Code Size		
Benchmarks (optimized for speed)	Packing Binary Coded Decimal (BCD)	5 cycles/0.25µs	12 cycles/1.5µs	6.0	0.57
		4 bytes	7 bytes		
	Loop Control	3 cycles/0.15µs	3 cycles/0.375µs	2.5	1.0
		2 bytes	2 bytes		
	Bit Test & Branch	3 cycles/0.15µs	5 cycles/0.625µs	4.17	1.0
		3 bytes	3 bytes		
	Shifting out 8-bit Data & Clock	169 cycles/8.45µs	205 cycles/25.625µs	3.03	1.12
		28 bytes	25 bytes		
	10ms Software Timer	-	-	-	1.0
		9 bytes	9 bytes		
	Five Byte Block move	14 cycles/0.7µs	25 cycles/3.125µs	4.46	1.0
		15 bytes	15 bytes		
	Four Byte Binary Addition	16cycles/0.8µs	55 cycles/6.875µs	8.59	1.08
		13 bytes	12 bytes		
	Four Byte Packed BCD Subtraction	101 cycles/5.05µs	151 cycles/18.875µs	3.73	1.17
		41 bytes	35 bytes		
	Three Byte Table Search	68 cycles/3.4µs	63 cycles/7.875µs	2.32	1.08
		43 bytes	40 bytes		
	Input / Output Manipulation	30/17/20 cycles 1.5/0.85/1.0µs	37/27/33 cycles 4.625/3.375/4.125µs	3.08/3.97/4.13	1.03
		38 bytes	37 bytes		
	Switch Activated Two Second LED	59 cycles/2.95µs	64 cycles/8µs	2.71	1.18
		46 bytes	39 bytes		

Table 2. Benchmark Results (Continued)

	Benchmarks (Optimized for speed)	ZiLOG eZ8 CPU (Instruction Cycle Time = 0.05μs)	Motorola CPU08 (Instruction Cycle Time = 0.125μs)	eZ8 CPU Execution Speed Ratio[1]	eZ8 CPU Code Efficiency Ratio[2]
	TOTAL	505 cycles 25.25μs	680 cycles 85μs	3.36	1.08
		242 bytes	224 bytes		

1. The speed ratio is calculated as follows: (Time CPU08) / (Time eZ8 CPU). A number higher than 1 means eZ8 CPU is faster.
2. The code efficiency ratio is calculated as follows: (Code Size eZ8 CPU / Code Size CPU08). A number less than 1 means eZ8 CPU is more efficient.

In terms of execution time, the eZ8 CPU executes ***3.36 times faster*** compared to the CPU08 using 26% less CPU cycles overall. In terms of code efficiency, the eZ8 CPU uses 8% more program space compared to the CPU08.

Benchmark Details

Presented in the following pages are the benchmark implementation with detailed comments. The comments include for each instruction, the byte and instruction cycle counts. For each benchmark, a score comprised of total bytes, total instruction cycles and execution speed is presented. The execution speed is the product of the instruction cycle count and instruction cycle time.

Benchmark#1 - Packing Binary Coded Data*

This benchmark takes two bytes in RAM or registers, each containing a BCD digit in the lower nibble and create a packed BCD data byte, which is stored back in the register or RAM location holding the low BCD digit.

eZ8 CPU

```
; REGHI and REGLO assumed to be in current register page

REGHI  EQU %10
REGLOW EQU %11

PACK_BCD:
    swap  REGHI         ; 2/2 Swap BCD digit of REGHI to high nibble
    or    REGHI, REGLO  ; 2/3 Pack BCD digits in REGHI and REGLO
```

Score: 5 cycles / 4 bytes / 0.25µs

CPU08

```
; REGHI and REGLO assumed to be in page0

REGHI  EQU $50
REGLOW EQU $51

PACK_BCD:
    lda REGHI        ; 2/3 Load A with REGHI
    nsa              ; 1/3 Swap BCD digit to high nibble
    ora REGLOW       ; 2/3 Pack BCD digits
    sta REGHI        ; 2/3 Store A in REGHI
```

Score: 12 cycles / 7 bytes / 1.5µs

*From Reference [7] on Page 1

Benchmark #2 – Loop Control*

This benchmark is a simple loop control where a register containing a loop count is decremented, tested for zero, and if not zero, then branched back to the beginning of the loop.

eZ8 CPU

```
; Working Register R15 with loop count

LOOP_START:
    □...
    djnz    R15, LOOP_START     ; 2/3 Decrement R15 and branch to
                                ;     LOOP_START if R15 not zero
```

Score: 3 cycles / 2 bytes / 0.15µs

CPU08

```
; Index Register X with loop count

LOOP_START:
    ...
    dbnzx   LOOP_START   ; 2/3 Decrement X and branch to
                         ;     LOOP_START if X not zero
```

Score: 3 cycles / 2 bytes / 0.375µs

*From Reference [7] on Page 1

Benchmark #3 – Bit Test & Branch*

This benchmark tests a single bit in a register or a RAM location and makes a conditional branch. We assume that the most significant bit is tested and a branch is to be taken if the bit is set.

eZ8 CPU

```
; Working Register R15 with value to be tested

    …
    btjnz    7,R15,NEW_ADDRS     ; 3/3 Test bit 7 of R15 and branch to
                                 ;     NEW_ADDRS if set
    …
NEW_ADDRS:
    …
```

Score: 3 cycles / 3 bytes / 0.15µs

CPU08

```
; TREG assumed to be in page0

TREG  EQU $50
    …
    brset    7,TREG,NEW_ADDRS     ; 3/5 Test bit 7 of TREG and branch to
                                  ;     NEW_ADDRS if set
    …
NEW_ADDRS:
    …
```

Score: 5 cycles / 3 bytes / 0.625µs

*From Reference [7] on Page 1

Benchmark #4 – Shifting Out 8-Bit Data & Clock*

This benchmark generates data and clock under program control by toggling two output pins.

eZ8 CPU

```
; code optimized for speed
; Data is output on Port A, pin 0
; Clock is output on Port A, pin 1


XFER_DATA  EQU %10      ; Temporary register, holds data to shift out

SHIFT_OUT:
        ld  R15,#%08    ; 2/2 Load R15 with count
        ldx FD1H, #%FC  ; 4/2+2 Configure Port A pins 0 & 1 as output

XMIT1:
        andx FD3H,#%FC  ; 4/3 Toggle Port A Data & Clock output pins
        rrc XFER_DATA   ; 2/2 Rotate next data bit into carry
        jr  nc, XMIT2   ; 2/2+2 Jump to XMIT2 if carry not set
        orx FD3H,#%01   ; 4/3+1 If carry, set data bit (Port A, pin 0)
XMIT2:
        orx FD3H,#%00   ; 4/3 Set Clock bit (Port A, pin 1)
        djnz R15,XMIT1  ; 2/3+1 Decrement R15 and jump to XMIT1 if not
                        ;         zero
        andx FD3H,#%FC  ; 4/3 Toggle Port A Data & Clock output pins
```

Score: 169 cycles / 28 bytes / 8.45µs

CPU08

```
; code optimized for speed
; Data is output on Port A, pin 0
; Clock is output on Port A, pin 1


XFER_DATA  EQU $50         ; Temporary Register, holds data to shift out
PORTA      EQU $00         ; Port A
DDRA       EQU $04         ; Port A Configuration Register

SHIFT_OUT:
        lda XFER_DATA      ; 2/3 Load A with transfer data
        ldx #08            ; 2/2 Load X with count
        bset 0,DDRA        ; 2/4 Configure Port A pin 0 as output
        bset 1,DDRA        ; 2/4 Configure Port A pin 1 as output
XMIT1:
        bclr 0,PORTA       ; 2/4 Clear clock bit
        bclr 1,PORTA       ; 2/4 Clear data bit
        rola               ; 1/1 Rotate A left through carry
        bcc  XMIT2         ; 2/3 Branch to XMIT2 if carry clear
        bset 1,PORTA       ; 2/4 Set data bit
XMIT2:
        bset 0,PORTA       ; 2/4 Set clock bit
        dbnzx  XMIT1       ; 2/3 Decrement X and Branch to XMIT1 if zero
        bclr 0,PORTA       ; 2/4 Clear clock bit
        bclr 1,PORTA       ; 2/4 Clear data bit
```

Score: 205 cycles / 25 bytes / 25.625µs

*From Reference [7] on Page 1

Benchmark #5 – Software Timer*

This benchmark implements a 10ms time delay loop subroutine

eZ8 CPU

```
; code optimized for speed


TMRCNT_LOW  EQU %9A           ; 10ms decrement count low byte
TMRCNT_HIGH EQU %6F           ; 10ms decrement count high byte

DELAY:
        ld  R15, #TMRCNT_LOW  ; 2/2 Initialize R15 with low byte
        ld  R14, #TMRCNT_HIGH ; 2/2 Initialize R14 with high byte

LOOP:
        decw rr14             ; 2/5 Decrement Register Pair R14,R15
        jr   nz, LOOP         ; 2/2 Jump to Loop if not zero
        ret                   ; 1/4 Return from subroutine
```

```
Score:   - / 9bytes / 9.9999ms
```

CPU08

```
; code optimized for speed


TMRCNT1  EQU $FF           ; 10ms decrement count1 byte
TMRCNT2  EQU $68           ; 10ms decrement count2 byte

DELAY:
        lda  #TMRCNT1      ; 2/2 Initialize A with count1
        ldx  #TMRCNT2      ; 2/2 Initialize X with count2

LOOP:
        dbnza LOOP         ; 2/3 Decrement A and branch if not zero
        dbnzx LOOP         ; 2/3 Decrement X and branch if not zero
        rts                ; 1/4 Return from subroutine
```

```
Score:  - / 9bytes / 9.985ms
```

*From Reference [7] on Page 1

Benchmark #6 – Five Byte Block Move*

This benchmark moves only a block of five data bytes from a specific source location to a specific destination location.

eZ8 CPU

```
; code optimized for speed
; Source (src) and Destination (dst) Locations in current register page

dst EQU %10
src EQU %20
5B_MOVE:
    ld dst,src     ; 3/2+1 Move byte 0 from source to destination
    ld dst+1,src+1 ; 3/2+1 Move byte 1 from source to destination
    ld dst+2,src+2 ; 3/2+1 Move byte 2 from source to destination
    ld dst+3,src+3 ; 3/2+1 Move byte 3 from source to destination
    ld dst+4,src+4 ; 3/2 Move byte 4 from source to destination
```

```
Score: 14 cycles / 15 bytes / 0.7µs
```

CPU08

```
; code optimized for speed
; Source (src) and Destination (dst) Locations assumed to be in page0

dst EQU $50
src EQU $60
5B_MOVE:
    mov src,dst     ; 3/5 Move byte 0 from source to destination
    mov src+1,dst+1 ; 3/5 Move byte 1 from source to destination
    mov src+2,dst+2 ; 3/5 Move byte 2 from source to destination
    mov src+3,dst+3 ; 3/5 Move byte 3 from source to destination
    mov src+4,dst+4 ; 3/5 Move byte 4 from source to destination
```

```
Score: 25 cycles / 15 bytes / 3.125µs
```

*From Reference [6] on Page 1

Benchmark #7 – Four Byte Binary Addition*

This benchmark adds two four byte binary numbers and replaces the first operand with the result. This emulates an adding machine addition, where A + B replaces A. The benchmark is programmed as a subroutine, with the carry flag indicating an overflow.

eZ8 CPU

```
; code optimized for speed
; Source (src) and Destination (dst) Locations in current register page

dst EQU %10
src EQU %20
4B_BADD:
    add dst+3,src+3  ; 3/3 Add src and dst, least significant byte first
    adc dst+2,src+2  ; 3/3 Add with carry 2nd byte
    adc dst+1,src+1  ; 3/3 Add with carry 3rd byte
    adc dst,src      ; 3/3 Add with carry most significant byte
    ret              ; 1/4 return from subroutine
```

Score: 16 cycles / 13 bytes / 0.8µs

CPU08

```
; code optimized for speed
; Source (src) and Destination (dst) Locations assumed to be in page0

dst EQU $50
src EQU $60
4B_BADD:
    ldx   #04     ; 2/2 Load index register with count
    clc           ; 1/1 Clear carry
LP1:
    lda   dst-1,X ; 2/3 Load accumulator (A) with dst operand
    adc   src-1,X ; 2/3 Add src operand to A
    sta   dst-1,X ; 2/3 Store result in dst
    dbnzx LP1     ; 2/3 Decrement count and loop if not zero
    rts           ; 1/4 Return from subroutine
```

Score: 55 cycles / 12 bytes / 6.875µs

*From Reference [6] on Page 1

Benchmark #8 – Four Byte Packed BCD Subtraction*

This benchmark adds two eight digit (four bytes each) packed BCD numbers and replaces the first operand with the result. This emulates an adding machine addition, where A - B replaces A. The benchmark is programmed as a subroutine, with the carry flag being used to indicate a positive or negative result. The BCD decimal-adjust (da) instruction is used following the subtraction to achieve the correct BCD result.

eZ8 CPU

```
; code optimized for speed
; Source (src) and Destination (dst) Locations in current register page

dst EQU %10
src EQU %20
4B_BCDSUB:
    sub dst+3,src+3  ; 3/3 Subtract src from dst, low byte first
    da  dst+3        ; 2/2+1 Decimal adjust the result
    sbc dst+2,src+2  ; 3/3 Subtract the 2nd byte
    da  dst+2        ; 2/2+1 Decimal adjust the result
    sbc dst+1,src+1  ; 3/3 Subtract the 3rd  byte
    da  dst+1        ; 2/2+1 Decimal adjust the result
    sbc dst,src      ; 3/3 Subtract the most significant byte
    da  dst          ; 2/2 Decimal adjust the result
    jr  nc, DONE     ; 2/2 Jump to DONE if carry not set
NEG:
    ld  R12, #%04    ; 2/2 Load R12 with loop count
    ld  R11, #dst+3  ; 2/2 Load R11 with dst low byte address
    rcf              ; 1/2 Reset Carry Flag
LP1:
    clr R10          ; 2/2 Clear R10
    sbc R10,@R11     ; 2/4 Subtract next dst byte from 0
    da  R10          ; 2/2 Decimal adjust the result
    ld  @R11, R10    ; 2/3 Store the result in dst
    dec R11          ; 2/2 Decrement R11
    djnz R12, LP1    ; 2/3 Decrement R12 and jump to LP1 if not zero
    scf              ; 1/2 Set carry flag
DONE:
    ret              ; 1/4 Return from subroutine
```

```
Score: 101 cycles / 41 bytes / 5.05µs
```

CPU08

```
; code optimized for speed
; Source (src) and Destination (dst) Locations assumed to be in page0
; In CPU 08, the BCD decimal adjust command (DAA) only works following
; addition, not subtraction. Consequently, the BCD subtraction must be
; implemented as an addition by adding the complement of the subtrahend
; (2nd operand) to the 1st operand. This complement is achieved by
; subtracting the subtrahend from a packed BCD 99 and then adding one
; to the result.

dst EQU $50
src EQU $60
4B_BCDSUB:
    ldx   #04     ; 2/2 Load X with loop count
    clc           ; 1/1 Clear carry
LP1:
    lda   #$99    ; 2/2 Load A with BCD 99
    sbc   src-1,X ; 2/3 Subtract src-1+X from A
    add   #01     ; 2/2 Add 1 to result
    daa           ; 1/2 Decimal adjust A
    add   dst-1,X ; 2/3 Add dst-1+X to A
    daa           ; 1/2 Decimal adjust A
    sta   dst-1,X ; 2/3 Store result in dst-1+X
    dbnzx LP1     ; 2/3 Decrement X and branch to LP1 if not zero
    bpl   DONE    ; 2/3 Branch on plus to DONE
NEG:
    clc           ; 1/1 Clear carry flag
    ldx   #04     ; 2/2 Load X with loop count
LP2:
    lda   #$99    ; 2/2 Load A with BCD 99
    sbc   dst-1,X ; 2/3 Subtract dst-1+X from A
    add   #01     ; 2/2 Add 1 to result
    daa           ; 1/2 Decimal adjust A
    sta   dst-1,X ; 2/3 Store result in dst-1+X
    dbnzx LP2     ; 2/3 Decrement X and branch to LP2 if not zero
    sec           ; 1/1 Set Carry flag
DONE:
    rts           ; 1/4 Return from subroutine
```

Score: 151 cycles / 35 bytes / 18.875µs

*From Reference [6] on Page 1

Benchmark #9 – Three Byte Table Search*

This benchmark searches a 200-byte table (resident in program memory) for a three-byte character string, which may be resident anywhere in the lookup table (not necessarily on three byte boundaries). The status of the carry bit indicates the success or failure of the search. The benchmark is programmed as a subroutine.

eZ8 CPU

```
; code optimized for speed
; CHAR1, CHAR2, CHAR3 are the three characters to be searched
; TBASE indicates Table Base in Program Memory

CHAR1 EQU %FA
CHAR2 EQU %FB
CHAR3 EQU %FC

SIZE  EQU %C6        ; Table Size (2 less than 200)
TBASE EQU %01        ; Table Base – starts at 0100H in program memory

3B_TABSRCH:
    ld  R15, #SIZE   ; 2/2 Load R15 with table size
    clr R14          ; 2/2 Temp Register to hold Table Offset Pointer
    clr R11          ; 2/2 R10 & R11 function as register pair to
    ld  R10, #TBASE  ; 2/2    hold program memory Table Base Address
LP1:
    ldc R12, @RR10   ; 2/5 Load R12 with table byte from program memory
    cp  R12, #CHAR1  ; 3/3 Compare R12 with CHAR1
    jr  ne,  FAIL    ; 2/2 Jump to FAIL if not equal

    inc R11          ; 2/2 Increment R11
    ldc R12, @RR10   ; 2/5 Load R12 with table byte from program memory
    cp  R12, #CHAR2  ; 3/3 Compare R12 with CHAR2
    jr  ne,  FAIL    ; 2/2 Jump to FAIL if not equal

    inc R11          ; 2/2 Increment R11
    ldc R12, @RR10   ; 2/5 Load R12 with table byte from program memory
    cp  R12, #CHAR3  ; 3/3 Compare R12 with CHAR1
    jr  ne,  FAIL    ; 2/2 Jump to FAIL if not equal

    scf              ; 1/2 Set carry flag to indicate match
    ret              ; 1/4 Return from subroutine
FAIL:
    inc R14          ; 2/2 Increment R14
    ld  R11, R14     ; 2/2 Load R11 with R14
    djnz R15, LP1    ; 2/3 Decrement R15 and Jump if not zero to LP1
    rcf              ; 1/2 Reset carry flag
    ret              ; 1/4 Return from subroutine
```

Score: 68 cycles / 43 bytes / 3.4µs
Assumption: First search iteration fails with first byte mismatch, second search iteration successful.

CPU08

```
; code optimized for speed
; CHAR1, CHAR2, CHAR3 are the three characters to be searched
; TBASE indicates Table Base in Program Memory


CHAR1   EQU $FA
CHAR2   EQU $FB
CHAR3   EQU $FC

SIZE    EQU $10      ; Temp Register to hold Table Size
TPTR    EQU $11      ; Temp Register to hold Table Offset Pointer


TBASE   EQU $0100

3B_TABSRCH:
        mov     #$C6,SIZE  ;3/4 Initialize SIZE to 198 (Table Size – 2)
        clrx               ;1/1 Clear X
        stx     TPTR       ;2/3 Store X in TPTR
SRCH:
        lda     TBASE,X    ;3/4 Load A with first byte
        cmp     #CHAR1     ;2/2 Compare A with CHAR1
        bne     FAIL       ;2/3 Branch to FAIL if not equal

        incx               ;1/1 Increment X
        lda     TBASE,X    ;3/4 Load A with second byte
        cmp     #CHAR2     ;2/2 Compare A with CHAR2
        bne     FAIL       ;2/3 Branch to FAIL if not equal

        incx               ;1/1 Increment X
        lda     TBASE,X    ;3/4 Load A with third byte
        cmp     #CHAR3     ;2/2 Compare A with CHAR3
        bne     FAIL       ;2/3 Branch to FAIL if not equal

        sec                ;1/1 Set carry to indicate match
        rts                ;1/4 Return from subroutine
FAIL:
        inc     TPTR       ;2/4 Increment TPTR
        ldx     TPTR       ;2/3 Load X with TPTR
        dbnz    SIZE,SRCH  ;3/5 Decrement SIZE & Branch if not zero
        clc                ;1/1 Clear carry
        rts                ;1/4 Return from subroutine
```

Score: 63 cycles / 40 bytes / 7.875µs
Assumption: First search iteration fails with first byte mismatch, second search iteration successful.

*From Reference [6] on Page 1

Benchmark #10 – Input / Output Manipulation*

This benchmark compares two 8-bit I/O ports P1 and P2. If they are equal, a nine is output as the least significant digit (lower nibble) of a third port P3. If port P1 is greater than port P2, then port P2 is output on port P1. If port P1 is less than port P2, then the most significant digit of Port P1 is copied to the least significant digit of Port P3.

eZ8 CPU

```
; code optimized for speed

PORTCMP:
    srp #%DF          ; 2/2 Set Page as $F, Group as $D
    ld  R1, #%FF      ; 2/2 Configure Port A as input (P1)
    ld  R5, #%FF      ; 2/2 Configure Port B as input (P2)
    ld  R9, #%0       ; 2/2 Configure Port C as output (P3)

    cp  R6, R2        ; 2/3 Compare P2 and P1
    jr  ge, POSITIVE  ; 2/2 Jump to POSITIVE if P2 > P1
NEGATIVE:
    ld  R1, #%0       ; 2/2+1 Configure P1 as output
    ld  R3, R6        ; 3/2 Write P2 value to P1
    jr  FINISH        ; 2/2 Jump to FINISH
POSITIVE:
    jr  zero, EQUAL   ; 2/2 Jump to EQUAL if P2 = P1
    srp #%0           ; 2/2+1 Set RP as $0
    ldx R15, FD2H     ; 3/2 Load R15 with P1
    swap R15          ; 2/2+1 Swap Upper and Lower Nibbles of R15
    and R15, #%0F     ; 3/3 Retain only Lower Nibble of R15
    ldx FDBH, R15     ; 3/2 Output R15 to P3
    jr  FINISH        ; 2/2 Jump to Finish
EQUAL:
    ld  R11, #%09     ; 2/2 Output digit 9 to P3
FINISH:
     ...
```

```
Score: 38 bytes
       P1 < P2: 30 cycles / 1.5µs
       P1 = P2: 17 cycles / 0.85µs
       P1 > P2: 20 cycles / 1.0µs
```

CPU08

```
; code optimized for speed


PORTA   EQU $00      ; Port P1
PORTB   EQU $01      ; Port P2
PORTC   EQU $02      ; Port P3
DDRA    EQU $04      ; Port A configuration register
DDRB    EQU $05      ; Port B configuration register
DDRC    EQU $06      ; Port C configuration register

PORTCMP:
        cla                 ; 1/1 Clear A
        sta   DDRA          ; 2/3 Configure Port A as input (P1)
        sta   DDRB          ; 2/3 Configure Port B as input (P2)
        deca                ; 1/1 Decrement A to all ones
        sta   DDRC          ; 2/3 Configure Port C as output (P3)

        lda   PORTB         ; 2/3 Load A with Port P2 data
        cmp   PORTA         ; 2/3 Compare Port P1 data with Port P2 data
        bpl   POSITIVE      ; 2/3 Branch to POSITIVE if P2 > P1
NEGATIVE:
        mov   #$FF,DDRA     ; 3/4 Configure Port A as output(P1)
        lda   PORTB         ; 2/3 Load A with Port P2 data
        sta   PORTA         ; 2/3 Output Port P2 data to Port P1
        bra   FINISH        ; 2/3 Branch to FINISH
POSITIVE:
        beq   EQUAL         ; 2/3 Branch to EQUAL if P2 = P1
        lda   PORTA         ; 2/3 Load A with Port P1 data
        and   #$F0          ; 2/2 Extract high order nibble of Port P1
        nsa                 ; 1/3 Exchange upper and lower nibbles
        sta   PORTC         ; 2/3 Output result to Port P3
        bra   FINISH        ; 2/3 Branch to FINISH
EQUAL:
        mov   #$09,PORTC    ; 3/4 Output digit 9 to Port P3
FINISH:
        ...
```

```
Score: 37 bytes
       P1 < P2: 37 cycles / 4.625µs
       P1 = P2: 27 cycles / 3.375µs
       P1 > P2: 33 cycles / 4.125µs
```

*From Reference [6] on Page 1

Benchmark #11 – Switch Activated Two Second LED*

This benchmark samples a switch input to activate a two-second output for turning on an LED. The switch is debounced with a 10ms delay on both opening and closure. Once activated, the switch turns on an LED output for two seconds and turns it off, regardless of whether the switch is still activated. Once the switch is turned off, the procedure is repeated. Both the switch input and LED output are low true.

eZ8 CPU

```
; code optimized for speed
TLO     EQU $6F      ; 10ms Delay count low byte
THI     EQU $9A      ; 10ms Delay count high byte

LED2:
    srp #%DF          ; 2/2 Set Page as $F, Group as $D
    ld  R5, #%F0      ; 2/2+1 Configure Port B : Upper nibble as
                      ;     output and lower nibble as input

WAIT1:
    btjnz 0,R6,WAIT1  ; 3/3 Wait for input switch (Port B, Pin 0) ON
    call DLY10        ; 3/3+1 Call 10ms delay subroutine
    andx FD7H,#%7F    ; 4/3 Turn on LED (Port B, Pin 7)
    ld   R0, #200     ; 2/2+1 Initialize R0 to obtain 2 second delay
SEC5:
    call DLY10        ; 3/3 Call 10ms delay subroutine
    djnz R0, SEC5     ; 2/3 Decrement and loop if not zero
    srp #%DF          ; 2/2 Set Page as $F, Group as $D
    bset 7,R7         ; 2/2+1 Turn off LED
WAIT2:
    brclr 0,R6,WAIT2  ; 3/5 Wait for input switch OFF
    call  DLY10       ; 3/3 Debounce 10ms
    srp #%DF          ; 2/2 Set Page as $F, Group as $D
    bra   WSWON       ; 2/3 Repeat procedure

DLY10:
    srp #%0           ; 2/2 Set Page and WRG as 0
    ld  R15,#TLO      ; 2/2 Initialize R15 with low byte
    ld  R14,#THI      ; 2/2 Initialize R14 with high byte
LOOP:
    decw rr14         ; 2/5 Decrement Register Pair R14,R15
    jr   nz, LOOP     ; 2/2 Jump to Loop if not zero
    ret               ; 1/4 Return from subroutine
```

```
Score: 59 cycles / 46 bytes / 2.95µs
Note: Cycles summed up ignoring wait loops
```

CPU08

```
; code optimized for speed


PORTB   EQU $01      ; Port B memory address
DDRB    EQU $05      ; Port B configuration register
TLO     EQU $FF      ; 10ms Delay count1 byte
THI     EQU $68      ; 10ms Delay count2 byte

LED2:
        lda   #$F0           ; 2/2 Load A with Port B config and data
        sta   PORTB          ; 2/3 Output 0 on Port B upper nibble
        sta   DDRB           ; 2/3 Configure Port B : Upper nibble as
                             ;     output and lower nibble as input
WSWON:
        brset 0,PORTB,WSWON  ; 3/5 Wait for input switch ON(low true)
SWON:
        jsr   DLY10          ; 2/4 Debounce 10ms
        bclr  7,PORTB        ; 2/4 Turn on LED (low true)
        lda   #200           ; 2/2 Initialize A
SEC5:
        jsr   DLY10          ; 2/4 Call 10ms subroutine 200 times
                             ;     to get 2s LED on time
        dbnza                ; 2/3 Decrement A & loop if not zero
        bset  7,PORTB        ; 2/4 Turn off LED
WSWOFF:
        brclr 0,PORTB,WSWOFF ; 3/5 Wait for input switch OFF
        jsr   DLY10          ; 2/4 Debounce 10ms
        bra   WSWON          ; 2/3 Repeat procedure

DLY10:
        psha                 ; 1/2 Save A
        lda   #TLO           ; 2/2 Set up outer loop count
        ldx   #THI           ; 2/2 Set up inner loop count
LOOP:
        dbnzx                ; 2/3 Decrement X & loop if not zero
        dbnza                ; 2/3 Decrement A & loop if not zero
        popa                 ; 1/2 Restore A
        rts                  ; 1/4 Return from subroutine
```

```
Score: 64 cycles / 39bytes / 8µs
Note: Cycles summed up ignoring wait loops
```

*From Reference [6] on Page 1