# ASSEMBLY PROBLEMS

*Introduction to the Logic Countdown game – to be developed in assembly code for the AT Mega Micro-controller.*

By Kayleigh Lamb
(13th April 2015)

# *Contents*

# ASSEMBLY PROBLEMS

## *Introduction to the Logic Countdown Game*

Write a program such that two 8 bit random numbers are displayed on the seven segment display, the buzzer should sound briefly sound after each number is displayed, the numbers should then be logically combined by randomly selecting one the of the logic functions - AND, NAND, OR, NOR or XOR and displaying the result on the LEDs. e.g. if the two numbers were AC and B2, and the program had randomly chosen the OR function, the LEDs would display 10111110

| | Number (Hex) | Number (Binary) | Function |
|---|---|---|---|
| Number 1 | AC | 10101100 | OR |
| Number 2 | B2 | 10110010 | |
| Result | BE | 10111110 | |

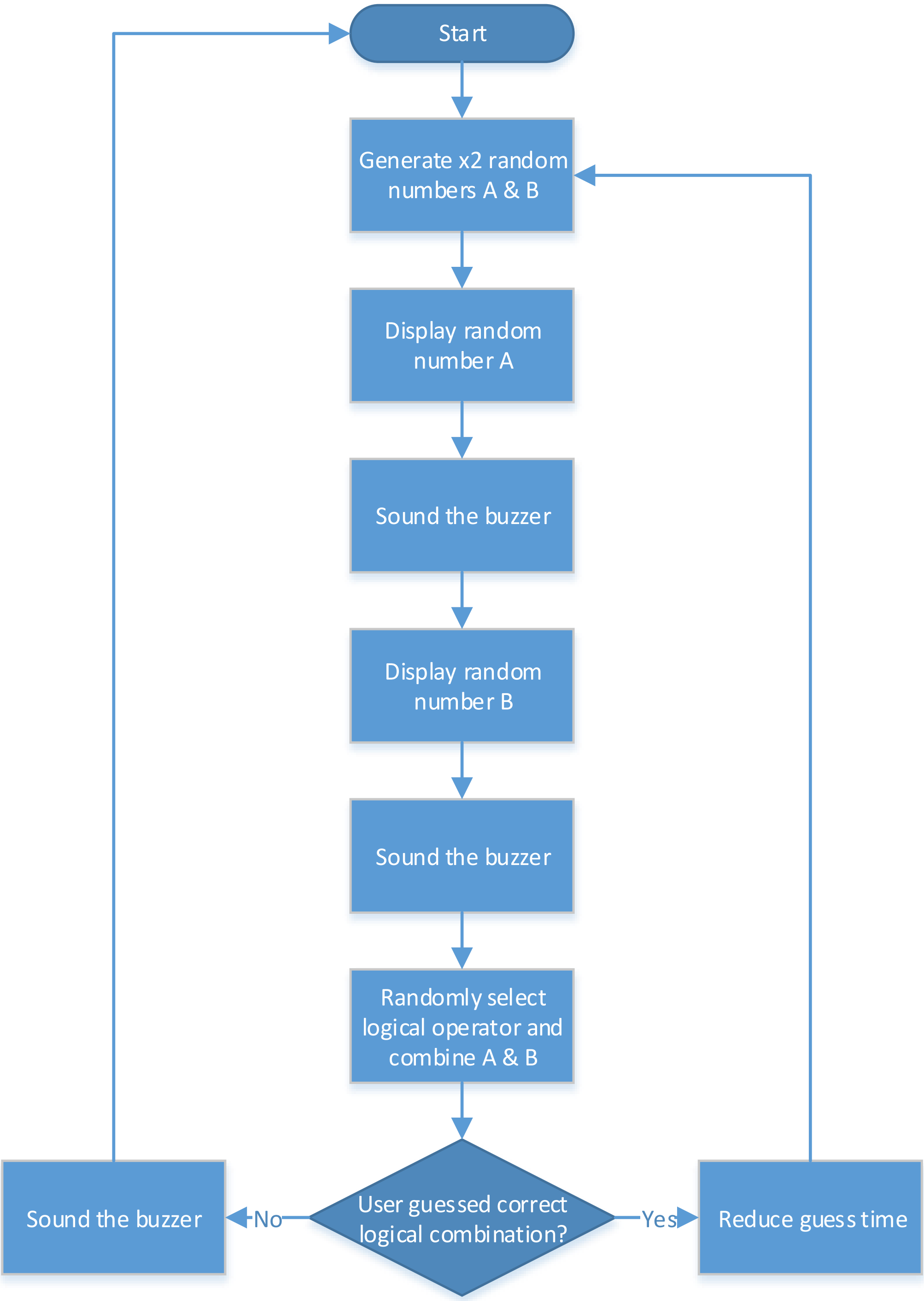
The user should then push one of the push buttons to indicate which logic function had been performed on the two numbers. The push button mapping is:-

| PB 4 | PB 3 | PB 2 | PB 1 | PB 0 |
|---|---|---|---|---|
| AND | NAND | OR | NOR | XOR |

After each correct entry the program should randomly logically combine two more random numbers and display the result on the LEDs and decrease the amount of time the user has to select the correct logic function. If the user is timed out before the correct function is entered, the program should alert the user of the error by sounding the buzzer and restarting the game.

# Design

## *Main Flowchart*


Start
Generate x2 random numbers A & B
Display random number A
Sound the buzzer
Display random number B
Sound the buzzer
Randomly select logical operator and combine A & B
User guessed correct logical combination?
No
Sound the buzzer
Yes
Reduce guess time

# Development

## *Stage 1*

I will begin the development of my program with stage 1, which will firstly involve getting the buzzer to sound for a certain amount of time.

I was able to achieve this by plugging the buzzer into port C of the SK200 development board and writing a piece of code that firstly sent 0 to port C, called a small delay (period) then sent 1 to port C and called the small delay again.
I then looped round this for as long as I wanted the buzzer sound - 199.662 ms.

I had no problems writing this section of the code, but during testing I noticed that altering the length of the small delay, determined the pitch of the buzzer.

```
;***************Buzzer subroutine 195.84 ms plus 2x 1.911 ms (199.662
ms)***************

.equ  PORTC =$15  ;Port C Output Address (Buzzer)
.equ  DDRC  =$14  ;Port C Data Direction Register Address

buzzer: ldi temp2,$FF       ;Initialise loop2 counter
two:    ldi temp1,$FF       ;Initialise loop1 counter
        ldi temp3,$FF       ;Load temp3 with 1's
        out PORTC,temp3     ;Load port C with 1 on bit 3
        rcall period        ;Period of sound wave
        ldi temp3,$00       ;Load temp3 with 0's
        out PORTC,temp3     ;Load port C with 0 on bit 3
        rcall period        ;period of sound wave
one:    dec temp1           ;Decrement loop one counter
        brne one            ;If loop one counter not yet 0, go round loop1
again
        dec temp2           ;Decrement loop two counter
        brne two            ;If loop two counter not yet 0, go round loop2
again
        rjmp buzzer         ;Loop forever

    ;Period delay section of code (1.911 ms @ 1MHz) - utilises r25 & r24

period:  ldi r24,$19        ;Initialise 2nd loop B counter
loop2:   ldi r25,$19        ;Initialise 1st loop A counter
loop1:   dec r25            ;Decrement the 1st loop counter
         brne loop1         ;and continue to decrement until 1st loop counter
= 0
         dec r24            ;Decrement the 2nd loop counter
         brne loop2         ;If the 2nd loop counter is not equal to zero
repeat the 1st loop, else continue
         ret
```

# Stage 2

For stage 2 of my program design, I will create a random number generator. It is not possible to create a true random number generator, but I should be able to achieve a sense of randomness if I follow the suggested algorithm in the coursework notes as follows:

“Pseudo Random Number Algorithm
If you start with a pre-seeded number, something other than zero, you can use the XOR and shift method to create a random-number generator. When done correctly, you will have a sequence that doesn't repeat for ($2^n$ - 1) times, where n is the number of bits you are shifting.
It works like this. Start with a number, for example the hex number 0x66. or 01100110. Take the bottom two bits and XOR them together, then take that result and shift it onto the most significant bit of the number.
bit 0 = 0
bit 1 = 1
bit 0 XOR bit 1 = 1
result = 10110011 or 0xB3
If you continue this method, you achieve a sense of randomness. First number and successive numbers: - 66, B3, 59, AC, 56, AB, 55, etc.”

I was able to implement this fairly easily and came up with the following code which produced the same numbers as the above algorithm:

```
    ;random number generator
        ldi ZL,low(table*2) ;Set Z pointer to start of table
        ldi ZH,high(table*2)
        ldi temp1,$1        ;load mask to extract bit 0
        ldi temp2,$2        ;load mask to extract bit 1
        and temp1,random    ;Extract bit 0
        and temp2,random    ;Extract bit 1
        lsr temp2           ;shift mask2 right by 1 bit
        eor temp1,temp2     ;XOR bit 1 & 0
        ror temp1           ;Rotate right through carry flag
        ror random          ;Rotate the number in the carry flag onto beg of original
number
```

# *Stage 3*

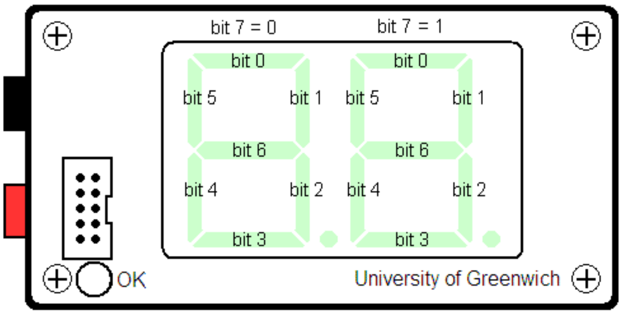
In order to send the correct hexadecimal values to port B for the 7 segment display, I will need to find a way to convert my random number to a min of $00 and a max of $0F to use for the Z register pointer. The Z register is two 8 bit registers paired to work as one 16 bit register.

As the maximum that a 16 bit register can hold is 256 decimal or $FF hex we can convert the random number by decrementing it by 16 decimal or $10 hex as we know that $16^2 = 256$. After each decrement, we can increment the Z register by 1 and then test if the random number is now negative. If the random number is not negative then we simply need to repeat the process until this is true.

```
;Take the random number and ensure it has min - max (0 - 15)
        mov copy,random         ;make a copy of the random number
loop:   inc temp1               ;increment temp1
        adiw ZL,1               ;increment z pointer
        subi copy,$10           ;subtract decimal 16 from copy
        brcs minus              ;branch if minus
        rjmp loop               ;if copy not minus repeat loop

minus:  dec temp1
        sbiw ZL,1
        mov address,temp1       ;make a copy of the address location for the 
lookup table
        clr temp1               ;clear temp register
```

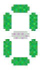
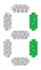
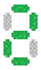
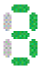
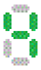
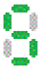
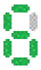
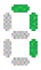
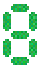
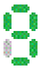
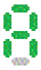
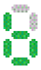
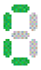
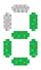
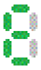
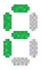
# *Stage 4*



This will involve generating just one random 8 bit hexadecimal number and displaying it on the 7 segment display. I will begin by storing the hexadecimal values for each display digit in a lookup table as follows:

```
table: .db
$3F,$06,$5B,$4F,$66,$6D,$7D,$07,$7F,$6F,$77,$7C,$39,
$5E,$79,$71
```

### *Data used for lookup table:*

| Display Number | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Hexadecimal | Display |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 3F |  |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 06 |  |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5B |  |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 4F |  |
| 4 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 66 |  |
| 5 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 6D |  |
| 6 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 7D |  |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 07 |  |
| 8 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7F |  |
| 9 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 6F |  |
| A | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 77 |  |
| B | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 7C |  |
| C | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 39 |  |
| D | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 5E |  |
| E | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 79 |  |
| F | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 71 |  |

## Stage 5 (Final Stage)

For stage 5 of the program development, I need to piece together the other subroutines and write code that will send the random lookup table hex values to port C where I will plug in the 7 segment display. Please see the final version of my program below:

```
;***************Logic Count Down Game - By Kayleigh Lamb - 000773715
***************

    ;Stack and Stack Pointer Addresses
.equ    SPH     =$3E    ;High Byte Stack Pointer Address
.equ    SPL     =$3D    ;Low Byte Stack Pointer Address
.equ    RAMEND  =$25F   ;Stack Address

    ;Port Addresses
.equ    PORTB   =$18    ;Port B Output Address (7 segment display)
.equ    DDRB    =$17    ;Port B Data direction register address

.equ    PORTC   =$15    ;Port C Output Address (Buzzer)
.equ    DDRC    =$14    ;Port C Data Direction Register Address

    ;Register Definitions
.def    sgmnt   =r0     ;Register to store data for seven segment display
.def    temp1   =r16    ;Register for temporary storage (load, use and clear)
.def    temp2   =r17    ;Register for temporary storage (load, use and clear)
.def    temp3   =r18
.def    address =r19    ;Register for table address location
.def    random  =r20    ;Register to store random number data
.def    copy    =r21    ;Register to store copies of data
.def    count   =r22

.def    YL      =r28    ;Define low byte of Y
.def    YH      =r29    ;Define high byte of Y
.def    ZL      =r30    ;Define low byte of Z
.def    ZH      =r31    ;Define high byte of Z

;Program Initialisation
    ;Set stack pointer to end of memory
        ldi temp1,high(RAMEND)
        out SPH,temp1       ;Load high byte of end of memory address
        ldi temp1,low(RAMEND)
        out SPL,temp1       ;Load low byte of end of memory address
        clr temp1           ;reset the temp register

    ;Initialise ports for output
        ldi temp1,$FF
        out DDRB,temp1      ;Set port B for output
        out DDRC,temp1      ;Set port C for output
        clr temp1           ;Reset the temp register


;*************** Begin Main Program ***************
        ldi random,$66      ;Load $66 into random number register
main:   rcall rand1         ;Call the random number1 subroutine
        rcall hex           ;Call the hex subroutine
        rcall buzzer        ;Call the buzzer subroutine
        rjmp main           ;restart the program
```

```
;*************** Random Table Address Number Generator ****************
    ;first random number
rand1:  ldi ZL,low(table*2) ;Set Z pointer to start of table
        ldi ZH,high(table*2)
        ldi temp1,$1        ;load mask to extract bit 0
        ldi temp2,$2        ;load mask to extract bit 1
        and temp1,random    ;Extract bit 0
        and temp2,random    ;Extract bit 1
        lsr temp2           ;shift mask2 right by 1 bit
        eor temp1,temp2     ;XOR bit 1 & 0
        ror temp1           ;Rotate right through carry flag
        ror random          ;Rotate the number in the carry flag onto beg of
original number
        clr temp1           ;Clear the temp storage registers
        clr temp2

    ;Take the random number and ensure it has min - max (0 - 15)
        mov copy,random     ;make a copy of the random number
loop:   inc temp1           ;increment temp1
        adiw ZL,1           ;increment z pointer
        subi copy,$10       ;subtract decimal 16 from copy
        brcs minus          ;branch if minus
        rjmp loop           ;if copy not minus repeat loop

minus:  dec temp1
        sbiw ZL,1
        mov address,temp1   ;make a copy of the address location for the
lookup table
        clr temp1           ;clear temp register
        ret                 ;Return to next line in main program

;**************** Lookup table for 7 segment display hex values
****************

table: .db $3F,$06,$5B,$4F,$66,$6D,$7D,$07,$7F,$6F,$77,$7C,$39,$5E,$79,$71

;**************** Send random number to 7 segement display ****************

hex:    rcall delay         ;Call delay subroutine
        lpm                 ;Load sgmnt with data pointed to by Z
        out PORTB,sgmnt     ;and display data on port B
        rcall delay
        ret

    ;Delay Subroutine (25.349 ms @ 1MHz)
delay:  ldi YH,high($ffff)  ;Load high byte of Y
        ldi YL,low($ffff)   ;Load low byte of Y
loops:  sbiw Y,1            ;Decrement Y
        brne loops          ;and continue to decrement until Y=0
        ret                 ;Return to next line 7 segment display routine

;***************Buzzer subroutine 195.84 ms plus 2x 1.911 ms (199.662
ms)***************

buzzer: ldi temp2,$FF       ;Initialise loop2 counter
two:    ldi temp1,$FF       ;Initialise loop1 counter
        ldi temp3,$FF       ;Load temp3 with 1's
        out PORTC,temp3     ;Load port C with 1 on bit 3
        rcall period        ;Period of sound wave
        ldi temp3,$00       ;Load temp3 with 0's
        out PORTC,temp3     ;Load port C with 0 on bit 3
        rcall period        ;period of sound wave
one:    dec temp1           ;Decrement loop one counter
        brne one            ;If loop one counter not yet 0, go round loop1
again
        dec temp2           ;Decrement loop two counter
        brne two            ;If loop two counter not yet 0, go round loop2
again
        ret                 ;Return to main program

    ;Period delay section of code (1.911 ms @ 1MHz) - utilises r25 & r24

period:  ldi r24,$19        ;Initialise 2nd loop B counter
loop2:   ldi r25,$19        ;Initialise 1st loop A counter
loop1:   dec r25            ;Decrement the 1st loop counter
         brne loop1         ;and continue to decrement until 1st loop counter
= 0
         dec r24            ;Decrement the 2nd loop counter
         brne loop2         ;If the 2nd loop counter is not equal to zero
repeat the 1st loop, else continue
         ret
```