CPE301 – SPRING 2019

Design Assignment 1A

Student Name: Clayton Higbee

Student #: 5001189864

Student Email: higbee@unlv.nevada.edu

Primary Github address:

Directory: ESD301/DA/DA\_1A

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

No components used in this lab

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

;

; 16\_8\_Bit\_MULT\_V1.asm

;

; Created: 2/7/2019 3:37:04 PM

; Author : clayt

;

; This ASM file multiplies together a 16bit number with an 8bit number

; The bits of the 8bit number will be used to determine when to shift and add

; in the process of determining the result. Since there are 8 bits, a maximum

; of up to 8 shifts can occur. A shift and carry will only result if a bit

; of the 8 bit number is logical HIGH, or 1.

.ORG 00

; Declarations and initializations

LDI R25, 0b00000001 ; InputH (High) 16bit byte Multiplicand

LDI R24, 0b10101010 ; InputL (Low) 16bit byte Multiplicand

LDI R22, 0b00001111 ; Input 8bit byte Multiplier

STS 0x0111, R22 ; Store a copy of the Multiplier

LDI R21, 7 ; Maximum number of times to shift the 8bit number

STS 0x0100, R21 ; Storage address for maximum number of shifts

LDI R23, 0 ; Initialize the count of current multiplier shifts that have occurred

STS 0x0101, R23 ; Storage address of current multiplier shifts

L1:

LDS R21, 0x0100 ; Final Count Register for convenience

LDS R23, 0x0101 ; Hold a copy of COUNT to compare

SUB R23, R21 ; Check to see if COUNT has reached 8

BRNE ADDER ; If COUNT has not reached 8, then continue to attempt to shift and add

; If COUNT has reached 8, then we finish the program

RJMP FINISH

ADDER:

LSR R22 ; Logical Shift Right the 8bit multiplicand

BRCS SHIFTCHECK; If carry flag is set to 1, copy the 16bit multiplicand and use for addition

; else increment counter by 1 and return

CLC ; clear carry flag

LDS R23, 0x0101 ; Increment and store the counter even if no addition occurs

INC R23

STS 0x0101, R23

RJMP L1 ; Begin the next count

SHIFTCHECK:

LDS R21, 0 ; Initialize a zero in the Shift Counter

STS 0x0110, R21

MOV R26, R24 ; copy the multiplicand

MOV R27, R25

LDI R28, 0

L2:

; Compare number of times shifted to the current count

; Count determines number of shifts needed

LDS R23, 0x0101

LDS R29, 0x0110

SUB R29, R23 ; Check to see if shifted enough

; If number of times shifted does not match the count, then shift again

BRNE SHIFT1

; If the number of times shifted does match the count, then add to the current total

RJMP TOTAL

SHIFT1:

CLR R16 ; Clear Carry Register

CLR R17 ; Clear Carry Register

; Increment the current shift counter

LDS R29, 0x0110

INC R29

STS 0x110, R29

; Create an empty register for convenience of carries

LDI R29, 0

LSL R26

; If a carry occurs, it will be added into carry register R16

ADC R16, R29

LSL R27

; If a carry occurs, it will be added into carry register R17

ADC R17,R29

LSL R28

ADD R27, R16 ; Add in any carry bit

ADD R28, R17 ; Add in any carry bit

RJMP L2 ; Return to check in shifted enough times

TOTAL:

CLR R29 ; Clear for use

CLR R30 ; Clear for use

ADD R18, R26 ; Final result location

BRCS CARRY1 ; If a carry occurs from totaling, keep it in R29

CARRY1D: ; 1st Carry was picked up if needed

ADD R19, R27 ; Final Result location

BRCS CARRY2

CARRY2D: ; 2nd Carry was picked up if needed

ADD R20, R28 ; Final Result location

ADD R19, R29 ; Add in any carry bit

ADD R20, R30 ; Add in any carry bit

LDS R23, 0x0101

INC R23

STS 0x0101, R23

RJMP L1

CARRY1:

LDI R29, 0b00000001

RJMP CARRY1D

CARRY2:

LDI R30, 0b00000001

RJMP CARRY2D

FINISH:

LDS R22, 0x0111 ; Load a copy of the Multiplier in

; Proof Check using the MUL function

; See Proof Check results in R31:R29

MUL R24, R22

MOV R29, R0

MOV R30, R1

MUL R25, R22

ADD R30, R0

ADC R31, R1

end:

jmp end

1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**

The following code was used to implement hardware memory for the storage of the 8bit multiplier and the 16bit multiplicand, as well as the 24bit result:

.ORG 00

; Declarations and initializations

LDI R25, 0b00000001 ; InputH (High) 16bit byte Multiplicand

LDI R24, 0b10101010 ; InputL (Low) 16bit byte Multiplicand

LDI R22, 0b00001111 ; Input 8bit byte Multiplier

STS 0x0111, R22 ; Store a copy of the Multiplier

.

.

.

TOTAL:

CLR R29 ; Clear for use

CLR R30 ; Clear for use

ADD R18, R26 ; Final result location

BRCS CARRY1 ; If a carry occurs from totaling, keep it in R29

CARRY1D: ; 1st Carry was picked up if needed

ADD R19, R27 ; Final Result location

BRCS CARRY2

CARRY2D: ; 2nd Carry was picked up if needed

ADD R20, R28 ; Final Result location

ADD R19, R29 ; Add in any carry bit

ADD R20, R30 ; Add in any carry bit

LDS R23, 0x0101

INC R23

STS 0x0101, R23

RJMP L1

1. **SCHEMATICS**

No schematics needed.

1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**

Task 1 – The method of multiplication I used was to mimic direct Binary multiplication using a shift and add method. The number of times to shift and add are dependent on the bits of the multiplier. A bit is checked to determine if the shift and add occurs. If the bit is a 1, then it calculates the number of times it needs to shift, then it adds it to the running sum. If the bit is 0, then it does nothing and checks the next bit. Following the code, you will begin to see the procedure of binary multiplication.

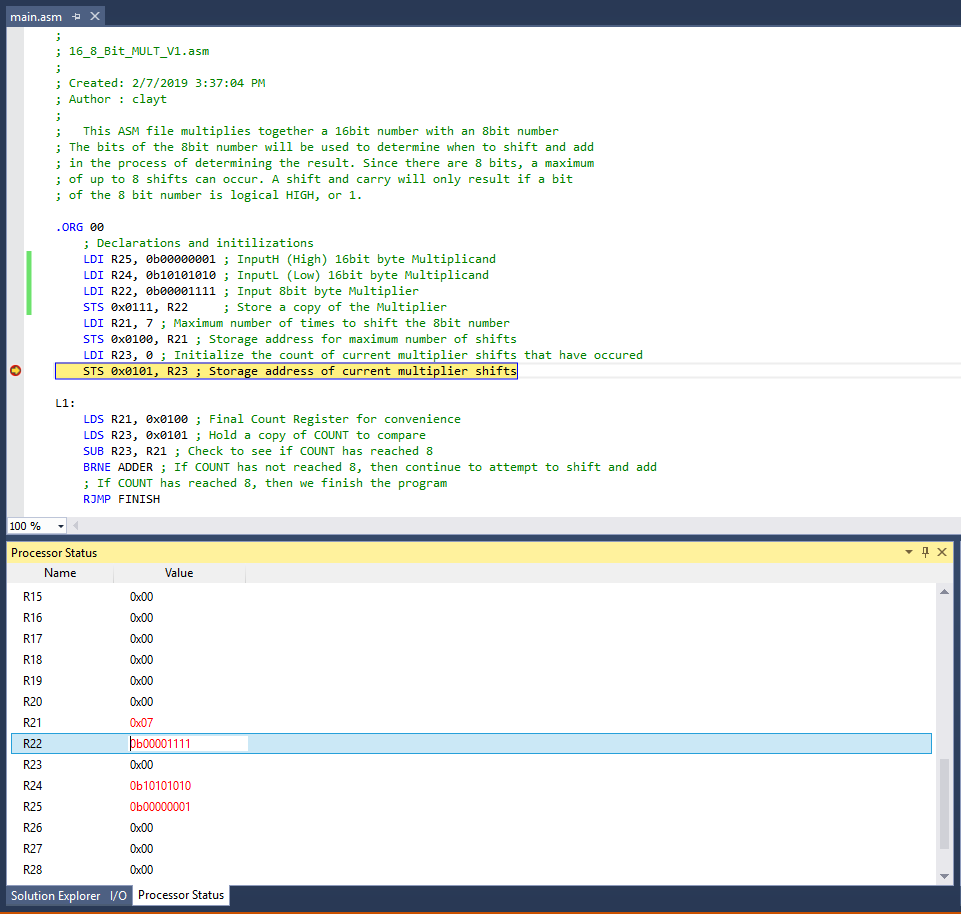
Task 2 – Depositing the answer

Provided numbers:

Multiplicand in R25:R24: 00000001 10101010

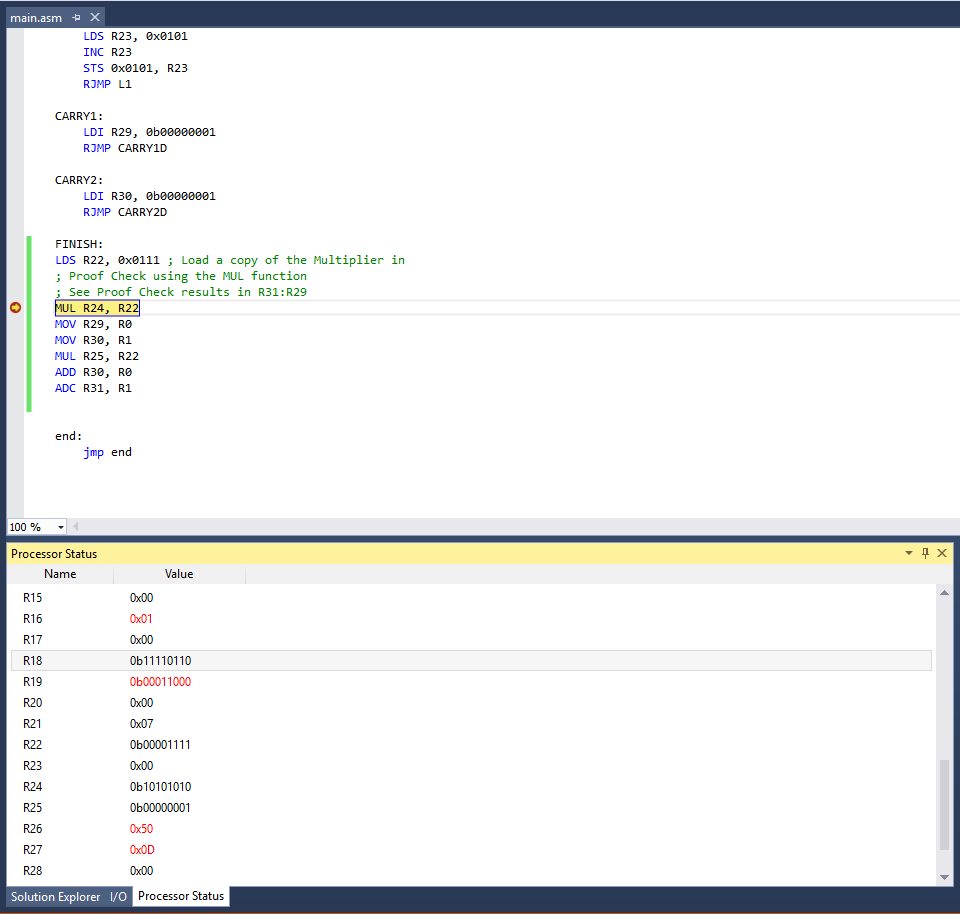
Multiplier in R22: 00001111

Screenshot of initial input numbers:



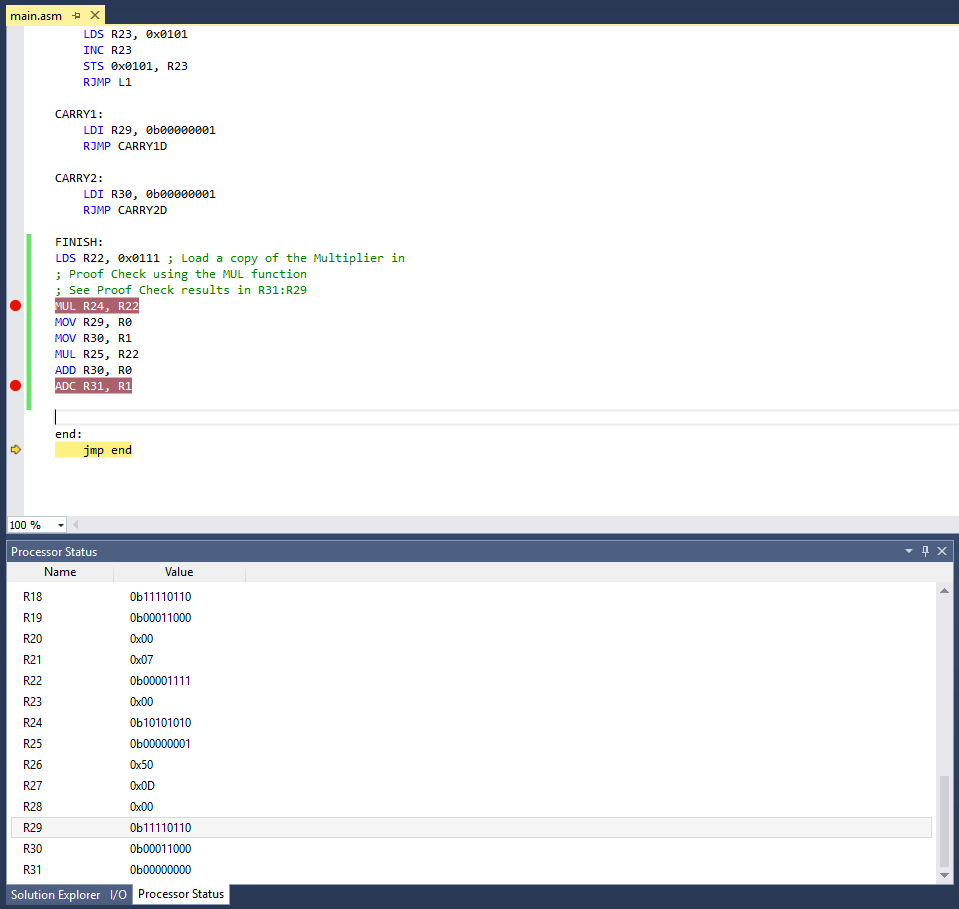
Output is located in R20:R18: 00000000 00011000 11110110

Screenshot of output:



Task 3

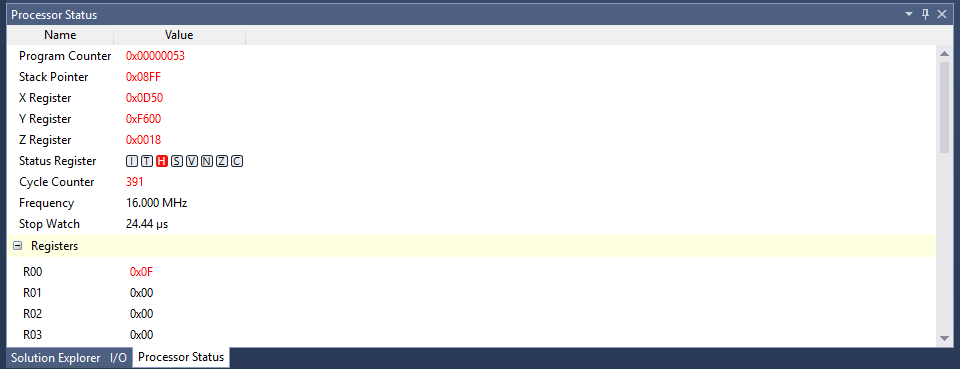
Using MUL and 3 result registers, we can check the result. The MUL Result has been stored in R31:R29



Task 4

Using ATMEL studio, we can determine the execution time. This program is very much dependent on the number of bits that are 1 in the multiplier. If all the bits are 1, it will be its longest execution, while if all are 0, then it would be its shortest. Current example has 4 out of 8 bits to be 1.

Looking at the processer status, it tells us that the full cycle counter came up to 391 at 16MHz, which comes to a total of 24.44μs



1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**

No screenshots needed as we did not setup the board for this assignment

1. **VIDEO LINKS OF EACH DEMO**

No video links required for this assignment

1. **GITHUB LINK OF THIS DA**

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Clayton John Higbee