CPE301 – SPRING 2019

Design Assignment 1A

Student Name: Francisco Mata Carlos

Student #: 1012593607

Student Email: matacarl@unlv.nevada.edu

Primary Github address: https://github.com/chicosisco/da\_sub.git

Directory: repository/cpe301/DesignAssignments/DA\_1A

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

List of Components used

Block diagram with pins used in the Atmega328P

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

Placed in number 3

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

; This code multiplies a 16-bit number with an 8-bit number using the shift and add topology.

; The multiplicand is the 16-bit number and the 8-bit number is the multiplier. The result can

; be up to 24 bits in length and will use three registers.

;

; Assignment\_1A.asm

;

; Created: 2/10/2019 3:09:12 AM

; Author : Francisco Mata

;

.set multiplicand\_hi = 0xFF ; value given to high byte of multiplicand

.set multiplicand\_lo= 0xFF ; value given to low byte of multiplicand

.set multiplier = 0xFF ; value givne to the multiplier

.org 0

start:

; r25 and 25 below are working together to represent a 16-bit multiplicand

; ========================================================================================

ldi r25, multiplicand\_hi ; register used to store high byte for multiplicand ;

ldi r24, multiplicand\_lo ; register used to store low byte for multiplicand ;

; =========================================================================================

ldi r22, multiplier ; register holding the 8-bit multiplier

; =========================================================================================

; the registers r18, r19, and r20 below hold the result

; =========================================================================================

ldi r18, 0 ; register used to store first 8-bit value of result

ldi r19, 0 ; register used to store second 8-bit value for result

ldi r20, 0 ; register used to store third 8-bit value of result

; ========================================================================================

ldi r26, 0 ; integer specifying number of times "shift and add" will loop

ldi r27, 0 ; register to temporarily hold the multiplier

ldi r28, 0 ; register use to place values of multiplicand as they are being shifted left

;ldi r16, 0 ;

ldi r17, 0 ; register used as a zero holder for math operations

mov r27, r22

add r22, r17 ; checking if multiplier is zero

breq end

; section below counts how many times the operation will shift and add

;========================================================================================

num\_shifts:

inc r26

lsr r27

brne num\_shifts

mov r27, r22

; =========================================================================================

; A) section below checks for the least sifnificant bit of multiplier as is

; being shifted right

;========================================================================================

loop:

lsr r27

brcs shift\_add

; section below only shift the multiplicand left if the LSB of the multiplier being checked

; section A) is zero

;========================================================================================

lsl r28

lsl r25

adc r28, r17

lsl r24

adc r25, r17

dec r26 ; decrementing r26 until is zero then all values from multiplier have been shifter right

; and multiplication process will be done

breq done ; if r26 is 0, process will end

jmp loop

; section below adds the multiplicand and shift it left if the LSB of the multiplier

; being checked in section A) is zero

;========================================================================================

shift\_add:

add r18, r24 ; adding r24 with r18 and storing in r18

adc r19, r25 ; adding r25 with r19 with carry and storing in r19

adc r20, r28 ; adding r28 with r20 with carry and storing in r20

lsl r28 ; shift r28 left by one number

lsl r25 ; shift r25 left by one value

adc r28, r17 ; moving the one value shifted left from r25 to r28

lsl r24 ; shifting r24 to the logically

adc r25, r17 ; moving the value shifted left from r24 to r25

dec r26 ; decrementing r26 until is zero then all values from multiplier have been shifter right

breq done ; if r26 is 0, process will end

jmp loop

done:

;jmp done

; the code for the multiplication using "shift and add" has ended here. Next is the code verifying answer using mul instruction

; =========================================================================================

; below is the code using mul to verify the answer

; =========================================================================================

; first the next set of register get cleared to load new values

clr r22

clr r24

clr r25

clr r28

clr r27

clr r26

; then values with multiplicand and multiplier are added again

ldi r24, multiplicand\_lo

ldi r25, multiplicand\_hi

ldi r22, multiplier

mul r24, r22 ; low byte of multiplicand being multiplied with multiplier, and result sent to r0 and r1

mov r26, r0 ; moving the value of r0 to r26

mov r27, r1 ; moving the value of r1 to r27

mul r25, r22 ; high byte of multiplicand being multiplied with multiplier, and result sent to r0 and r1 again

mov r28, r1 ; moving the value of r1 to r28

add r27, r0 ; adding the value of r0 with the value of r27

adc r28, r17 ; adding the carry if any from previous instruction to r28

end:

jmp end ; code and simulation ended by going into infinite loop

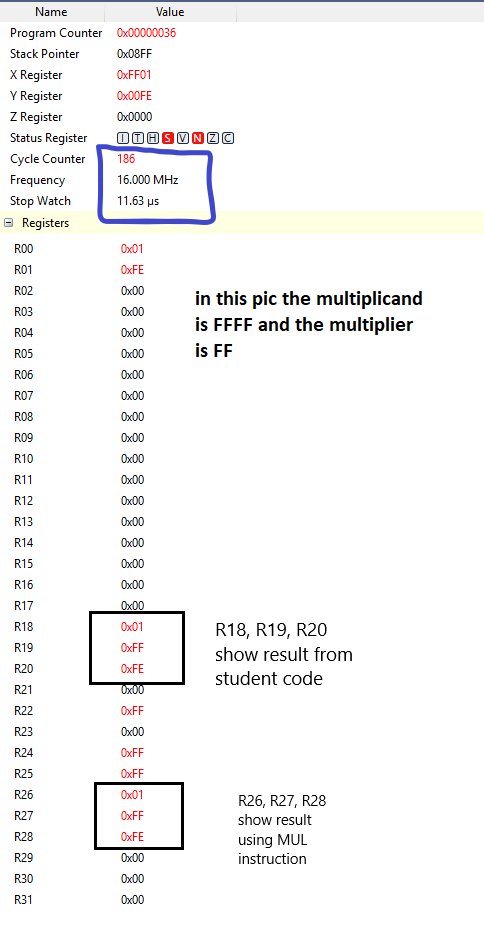
1. **SCHEMATICS**

NA

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

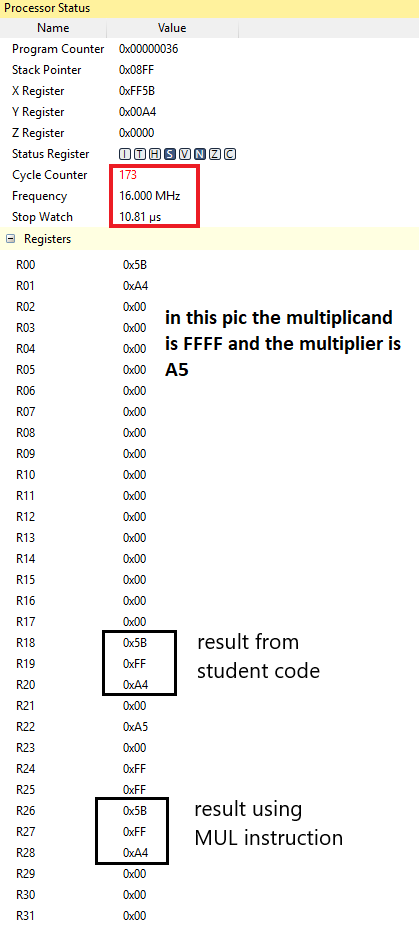
**Example 1 result**

Figure below shows the result in r18, r19, and r20 from the student’s code. The result in r26, r27, and r28 are the result of the multiplication using the MUL instruction. They are the same



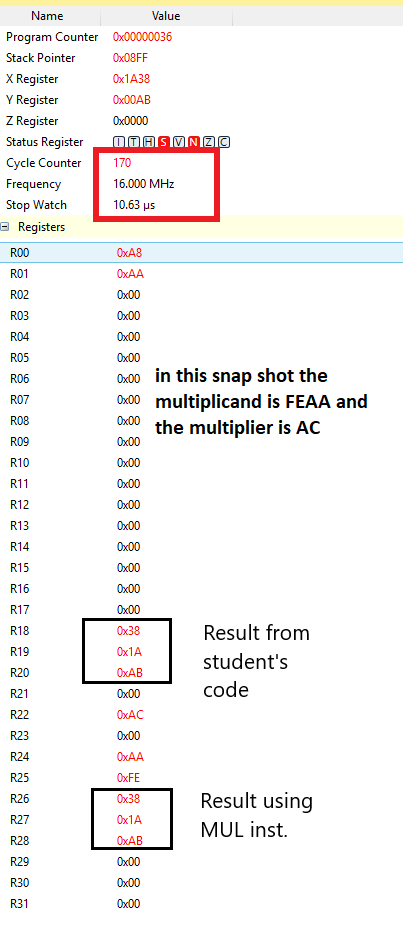
**Example 2 result**

Figure below shows the result in r18, r19, and r20 from the student’s code. The result in r26, r27, and r28 are the result of the multiplication using the MUL instruction. They are the same



**Example 3 result**

Figure below shows the result in r18, r19, and r20 from the student’s code. The result in r26, r27, and r28 are the result of the multiplication using the MUL instruction. They are the same



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

NA

1. **VIDEO LINKS OF EACH DEMO**

Only screenshots were taken

1. **GITHUB LINK OF THIS DA**

<https://github.com/chicosisco/da_sub.git>

**Student Academic Misconduct Policy**

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

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

Francisco Mata Carlos