CPE301 – SPRING 2019

Design Assignment 1, Part A

Student Name: James Skelly
Student #: 2000945485
Student Email: skellj1@unlv.nevada.edu
Primary Github address: https://github.com/skellj1/submission_da
Directory: skellj1/submission_da

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also, include the comments.
2. Use the previously create a Github repository with a random name (no CPE/301, Lastname, Firstname). Place all labs under the root folder ESD301/DA, sub-folder named LABXX, with one document and one video link file for each lab, place modified asm/c files named as LabXX-TYY.asm/c.
3. If multiple asm/c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) and other include files, c) text file with youtube video links (see template).

1. COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS

- Components used for this assignment include Atmel Studio 7 Simulator (for programming in assembly, viewing register and memory contents after execution, and analyzing processor status, status register at termination of program, and cycle counter) and the online hexadecimal calculator (https://www.miniwebtool.com/hex-calculator/) in order to verify correct calculation.

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

```
/*
 * Skelly_James_DA1.asm
 *
 *  Created: 2/3/2019 3:11:23 PM
 *  Author: James Skelly (CPE 301, Sp. 2019)
 */

 .ORG 0X00                      ; Sets the program to begin at memory location 0x00

 .EQU MULTIPLICAND = 0XFFFF     ; Initializes the multiplicand (16-bit value)
 .EQU MULTIPLIER = 0XFF         ; Initializes the multiplier (8-bit value)

 LDI R25, HIGH(MULTIPLICAND)    ; Places the higher 8-bits of the multiplicand into register 25
 LDI R24, LOW(MULTIPLICAND)     ; Places the lower 8-bits of the multiplicand into register 24
 LDI R22, MULTIPLIER            ; Loads the multiplier value into register 22
 LDI R21, MULTIPLIER            ; Keeps a copy of the multiplier in register 21 for review when
                                       ;program terminates
 LDI R16, 0x00                  ; Places the value zero into register 16

 LOOP:                          ; Loop label for repeated (iterative) addition
        ADD R20, R24            ; Begins repeated addition of the lower 8 bits of the multiplicand,
                                       ; places the value of the repeated addition in the first solution
                                       ; register, R18
        ADC R19, R25            ; Repeated addition of upper 8 bits, including addition of carry bit
                                       ; from SREG, places result in R19, the second solution register
        ADC R18, R16            ; Allocates bits 16-23 of the third solution register, initially adding
                                       ; the value 0 until further iterations where it will begin to sum
                                       ; up the carry values it receives from the SREG carry bit
        DEC R22                 ; Decrements R17, the multiplier, by 1 on each iteration
        BRNE LOOP               ; Branches back to the top of the LOOP subroutine until R17 is zero

        RJMP END                ; Jumps to the END label

END: RJMP END                   ; Program terminates
```

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

Not available for this assignment.

4. SCHEMATICS

Not available for this assignment.

5. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

TASKS

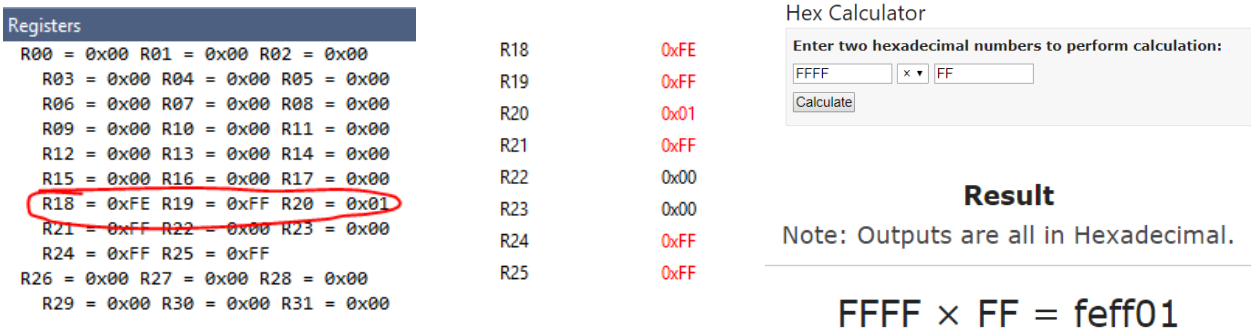
1. Perform a multiplication of a 16-bit multiplicand with an 8-bit multiplier **without using the MUL instruction.** Use iterative addition to perform this multiplication.
2. Registers R25:R24 should hold the multiplicand, R22 should hold the multiplier, and R20:R19:R18 should hold the result, which will be 24 bits.
3. Verify your algorithm using any valid form of verification.
4. Determine the execution time @ 16 MHZ, and the number of cycles, of your algorithm during simulation.

SAMPLE SOLUTION 1

Below is the code which initializes the multiplicand and the multiplier and places the necessary values for calculation into the registers specified in the instructions. The values 0xFFFF and 0xFF were selected so that the maximum execution time would be reached, as this multiplier will force the multiplicand to add to itself 255 times.

```
.EQU MULTIPLICAND = 0XFFFF      ; Initializes the multiplicand (16-bit value)
.EQU MULTIPLIER = 0XFF          ; Initializes the multiplier (8-bit value)

LDI R25, HIGH(MULTIPLICAND)     ; Places the higher 8-bits of the multiplicand into register 25
LDI R24, LOW(MULTIPLICAND)      ; Places the lower 8-bits of the multiplicand into register 24
LDI R22, MULTIPLIER             ; Loads the multiplier value into register 22
LDI R21, MULTIPLIER             ; Keeps a copy of the multiplier in register 21 for review when
                                    ;program terminates
```

At the completion of the execution of the code, we can look to the "Processor Status" and "Registers" windows to determine number of clock cycles, time elapsed (execution time), and to verify that the registers hold the values that we expect.



Above we can see the status of the data registers (the 24-bit result can be placed in R18:R19:R20 so that they are in order from MSB to LSB, or viceversa) at the end of the code execution. The result was verified with the online hex calculator shown above and to the right. Note also that the multiplicand is held in R24:R25 and the multiplier is held in R21.

Processor Status

Name	Value
Program Counter	0x0000000B
Stack Pointer	0x08FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	1535
Frequency	16.000 MHz
Stop Watch	95.94 µs

Shown to the left is the "Processor Status" window at the completion of execution. From this window, we can see that the entire execution took 95.94 us, or 1535 clock cycles at 16 MHz.

SAMPLE SOLUTION 2

Here is a second example in which the value of the multiplicand was selected to be 0xAF23, and the value of the multiplier was selected to be 0x23.

```
.EQU MULTIPLICAND = 0Xaf23      ; Initializes the multiplicand (16-bit value)
.EQU MULTIPLIER = 0X23          ; Initializes the multiplier (8-bit value)

LDI R25, HIGH(MULTIPLICAND)     ; Places the higher 8-bits of the multiplicand into register 25
LDI R24, LOW(MULTIPLICAND)      ; Places the lower 8-bits of the multiplicand into register 24
LDI R22, MULTIPLIER             ; Loads the multiplier value into register 22
LDI R21, MULTIPLIER             ; Keeps a copy of the multiplier in register 21 for review when
                                    ;program terminates
```

Below, we can see the solution stored in the solution registers R18:R19:R20, along with the multiplier in R21 and the multiplicand in R24 and R25.

Registers

```
R00 = 0x00 R01 = 0x00 R02 = 0x00
  R03 = 0x00 R04 = 0x00 R05 = 0x00
  R06 = 0x00 R07 = 0x00 R08 = 0x00
  R09 = 0x00 R10 = 0x00 R11 = 0x00
  R12 = 0x00 R13 = 0x00 R14 = 0x00
  R15 = 0x00 R16 = 0x00 R17 = 0x00
  R18 = 0x17 R19 = 0xF1 R20 = 0xC9
  R21 = 0x23 R22 = 0x00 R23 = 0x00
  R24 = 0x23 R25 = 0xAF
R26 = 0x00 R27 = 0x00 R28 = 0x00
  R29 = 0x00 R30 = 0x00 R31 = 0x00
```

R18	0x17
R19	0xF1
R20	0xC9
R21	0x23
R22	0x00
R23	0x00
R24	0x23
R25	0xAF

Hex Calculator

Enter two hexadecimal numbers to perform calculation:

0xaf23 × 0x23

Calculate

Result

Note: Outputs are all in Hexadecimal.

0xaf23 × 0x23 = 17f1c9

We can see on the right that the online hex calculator used for verification has a result which matches the result registers above to the left.

Processor Status

Name	Value
Program Counter	0x0000000B
Stack Pointer	0x08FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	215
Frequency	16.000 MHz
Stop Watch	13.44 µs

Shown to the left is the "Processor Status" window at the completion of execution. From this window, we can see that the entire execution took only 13.44 us, or 215 clock cycles at 16 MHz. Notice that this is a much smaller execution time than we had in solution 1. This is because the first solution used the maximum 8 bit and maximum 16 bit value for the largest possible computation using iterative addition to multiply.

SAMPLE SOLUTION 3

Here is one final sample solution to test the program, which uses 0x7254 as the multiplicand, and 0x10 as the multiplier.

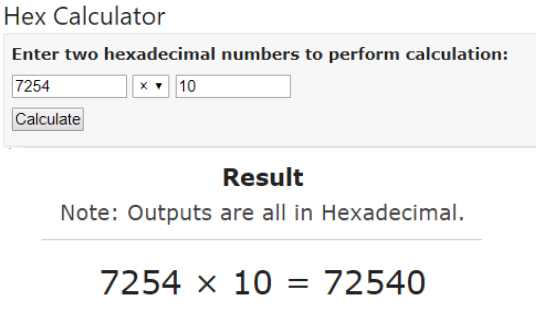
```
.EQU MULTIPLICAND = 0X7254      ; Initializes the multiplicand (16-bit value)
.EQU MULTIPLIER = 0X10          ; Initializes the multiplier (8-bit value)

LDI R25, HIGH(MULTIPLICAND)     ; Places the higher 8-bits of the multiplicand into register 25
LDI R24, LOW(MULTIPLICAND)      ; Places the lower 8-bits of the multiplicand into register 24
LDI R22, MULTIPLIER             ; Loads the multiplier value into register 22
LDI R21, MULTIPLIER             ; Keeps a copy of the multiplier in register 21 for review when
                                   ;program terminates
```

Below, we can see the solution stored in the solution registers R18:R19:R20, along with the multiplier in R21 and the multiplicand in R24 and R25.

Registers

```
R00 = 0x00 R01 = 0x00 R02 = 0x00
  R03 = 0x00 R04 = 0x00 R05 = 0x00
  R06 = 0x00 R07 = 0x00 R08 = 0x00
  R09 = 0x00 R10 = 0x00 R11 = 0x00
  R12 = 0x00 R13 = 0x00 R14 = 0x00
  R15 = 0x00 R16 = 0x00 R17 = 0x00
  R18 = 0x07 R19 = 0x25 R20 = 0x40
  R21 = 0x10 R22 = 0x00 R23 = 0x00
  R24 = 0x54 R25 = 0x72
R26 = 0x00 R27 = 0x00 R28 = 0x00
  R29 = 0x00 R30 = 0x00 R31 = 0x00
```

R18	0x07
R19	0x25
R20	0x40
R21	0x10
R22	0x00
R23	0x00
R24	0x54
R25	0x72



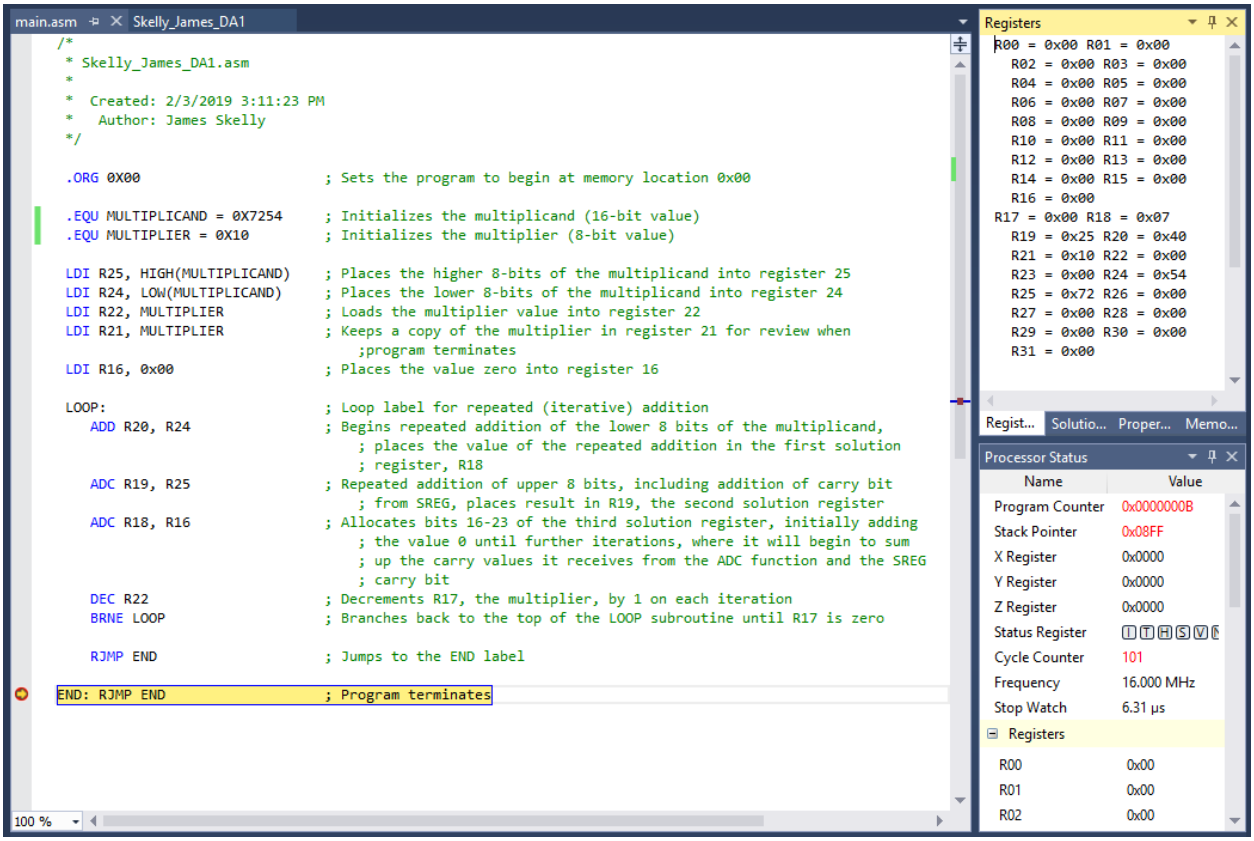
We can see on the right that the online hex calculator used for verification has a result which matches the result registers above to the left.

Processor Status

Name	Value
Program Counter	0x0000000B
Stack Pointer	0x08FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	101
Frequency	16.000 MHz
Stop Watch	6.31 µs

Note that as we decrease the value of the multiplier in each solution, the number of cycles/execution time decrease notably. This is because for repeated addition, the number of cycles is based entirely on the multiplier value, as repeated addition is simply the sum of the multiplicand added to itself n times, where n is the value of the multiplier.

Below is a final screenshot of the program written in Atmel Studio 7 with registers and processor status also viewable.



6. SCREENSHOT OF EACH DEMO (BOARD SETUP)

Not available for this assignment.

7. VIDEO LINKS OF EACH DEMO

Not available for this assignment.

8. GITHUB LINK OF THIS DA

https://github.com/skellj1/submission_da

Student Academic Misconduct Policy
http://studentconduct.unlv.edu/misconduct/policy.html

"*This assignment submission is my own, original work*".
James W. Skelly