

ECE2049 Homework #2

Submitted by: Jack C. Bergin

—

Date: 02/12/2021

Question	Grade
1-- 25	
2-- 10	
3-- 20	
4-- 20	
5-- 25	
Total: 100	

Jack Bergin
02/12/21
C21-ECE 2049

ECE2049 Homework #2 – The MSP430 Architecture & Basic Digital IO

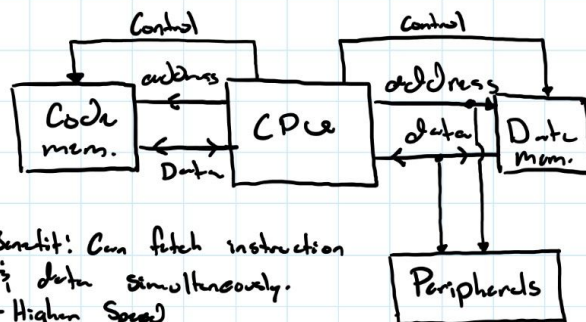
1) In the lab we are using the MSP430F5529. Using the MSP430x5xx Datasheet and MSP430F5529 User's Guide (under Useful Links on the class website) and class notes, fully answer the following questions about the architecture and memory map of the MSP430F5529.

(25 pts)

a) The MSP430 uses a von Neumann architecture. What is the main difference between the Harvard architecture and the VonNeumann architecture? Sketch both architectures showing CPU, memory, peripherals and all buses. Name one strength and one weakness for each architecture.

Question 1 Part A:

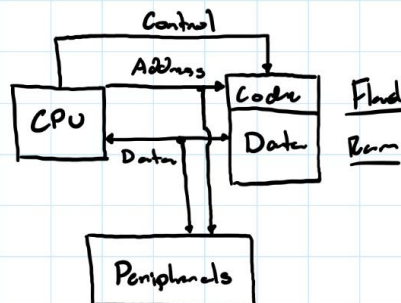
Harvard Architecture



Benefit: Can fetch instruction
& data simultaneously.
- Higher Speed
- Too complex on hardware and
software side.

Von Neumann Architecture - Single memory address space for all code & data.

- Intel
- MSP430 } Both use this
- Slower
- Complexity is
reduced



Flash
Mem

b) The MSP430F5529 memory space contains both RAM and FLASH. Why? What is the difference between RAM and FLASH? Why not use just RAM or just FLASH?

- ❖ RAM is known as volatile memory. This memory is not permanent so for example due to loss of power, the RAM memory is erased. Flash is known as non-volatile memory and is permanent unless direct action is taken to clear the memory. Even if the power is shut off flash will still save the information stored on it (hence the term flash used in “flash drive”). The purpose of volatile memory is to store temporary information and data that will be utilized for program testing. An example of this is with the serial monitor with arduino ide. The output is stored for a little but upon re-execution of the program code, the data is erased and new data is written. Non-volatile memory is the permanent storage of data and can be seen as the actual program being saved at hand. Computer systems utilize a combination of both volatile and nonvolatile memory because there are functions that require both in order to work.

c) Flash serves a number of purposes in the MSP430. How much Flash memory does the MSP430F5529 have? Into how many banks is code memory divided? How many bytes per bank? Which bank of code memory contains the address 0x0CFA2?

- ❖ The MSP430F5529 has 128KB of Flash in total with the code memory separated into four banks. Each bank has 32KB of memory and the bank with the address of 0x0FA2 is bank B.

d) How much RAM does the MSP430F5529 have and how is it organized (number of banks, etc). What is the beginning address of RAM? What is RAM used for?

- ❖ The MSP430F5529 has a total of 10KB of RAM It is divided into five 2KB sectors.

e) In lab, when CCS downloads a program to the MSP430F5529, where is the executable code stored? Beginning at what address?

- ❖ In lab, when CCS downloads a program to the MSP430 the data is stored on the flash of the device. More specifically it is stored in code memory beginning at the address 04400h.

f) When you declare a local variable inside your main() where in memory is that value stored? What if you declared a global variable, where in memory are globals stored?

- ❖ Both variables are stored in the RAM of the device with the address of the storage beginning at 02400h

g) Why does the MSP430F5529 have a 20-bit address bus when its word size is 16 bits? Also why are registers R0 (i.e. Program Counter) and R1 (the Stack Pointer) 20 bit registers instead of 16 bit?

- ❖ The MSP430 has a 20-bit memory address bus so that it has the ability to access a maximum of 1 MB of memory. The Program Counter and Stack Pointer are used to hold memory addresses and when trying to store a maximum of 1 MB, the 20-bit memory addresses are the ones utilized.

h) What is the address for the Port 6 direction register, the Port 3 input register, and the Port 7 select register. (See Datasheet Tables 9-26).

- ❖ The address for the Port 6 direction register is at 0245h, the Port 3 input register at 0220h, and the Port 7 select register at 026Ah.

i) What MSP430 peripheral device uses addresses 00700h-0073Fh? What package pins numbers are used for Port 8 digital IO?

- ❖ The peripheral device to use addresses 00700h-0073Fh is ADC12_A and the package pin numbers for Port 8 digital IO being 15-17.

j) Most of the MSP430F5529 package pins have multiple functions that are selected using a select register. Why are all these pins multiplexed? What functionality is multiplexed with the Port 4.4-4.7 digital I/O? With Port 2.2 and 3.3? What package pin(s) is analog Vcc and which package pin(s) is GND (Vss)?

- ❖ These pins are multiplexed to give the processing chip on the MSP430 a greater amount of capability without requiring added space needed by having a single function per single pin. Rather than specifying one function per pin, the function of the pin can be specified. AVCC is packaged to pin 1`4 while AVCC is packaged to pin 11. This allows for AVVSS to be 0V (Ground) and for the VCC on the MSP4030 to be 3.3V.

P4.4/PM_UCA1TXD/ PM_UCA1SIMO	51	45	H3	D7	I/O	General-purpose digital I/O with reconfigurable port mapping secondary function Default mapping: Transmit data – USCI_A1 UART mode Default mapping: Slave in, master out – USCI_A1 SPI mode
P4.5/PM_UCA1RXD/ PM_UCA1SOMI	52	46	G3	C9	I/O	General-purpose digital I/O with reconfigurable port mapping secondary function Default mapping: Receive data – USCI_A1 UART mode Default mapping: Slave out, master in – USCI_A1 SPI mode
P4.6/PM_NONE	53	47	F3	C8	I/O	General-purpose digital I/O with reconfigurable port mapping secondary function Default mapping: no secondary function.
P4.7/PM_NONE	54	48	E4	C7	I/O	General-purpose digital I/O with reconfigurable port mapping secondary function Default mapping: no secondary function.
P2.2/TA2CLK/SMCLK	31	28	E7	J6	I/O	General-purpose digital I/O with port interrupt TA2 clock signal TA2CLK input SMCLK output
P3.3/UCA0TXD/ UCA0SIMO	40	37	G6	G9	I/O	General-purpose digital I/O Transmit data – USCI_A0 UART mode Slave in, master out – USCI_A0 SPI mode

2) The MSP430 is a small, 16-bit microcontroller. The device's forte is that it is capable of remarkably low power operation. It is not a processor suited to computationally heavy applications. It is a fixed point (i.e. integer) processor. That means the CPU contains a digital circuit called the Arithmetic Logic Unit (ALU) which performs addition and subtraction and the logic operations AND, OR, NOT and XOR on integers but it does not contain a Floating Point Unit (FPU). That is, there is no circuit inside an MSP430 that performs arithmetic operations on floating point numbers. The Code Composer Studio compiler does provides full support for floating point operations, as all C compilers must, but all floating point math on an MSP430 is *emulated* in software which has significant implications for both code size and run time. (10 pts)

i. Add a comment to each line of code below saying what it does
Part A)

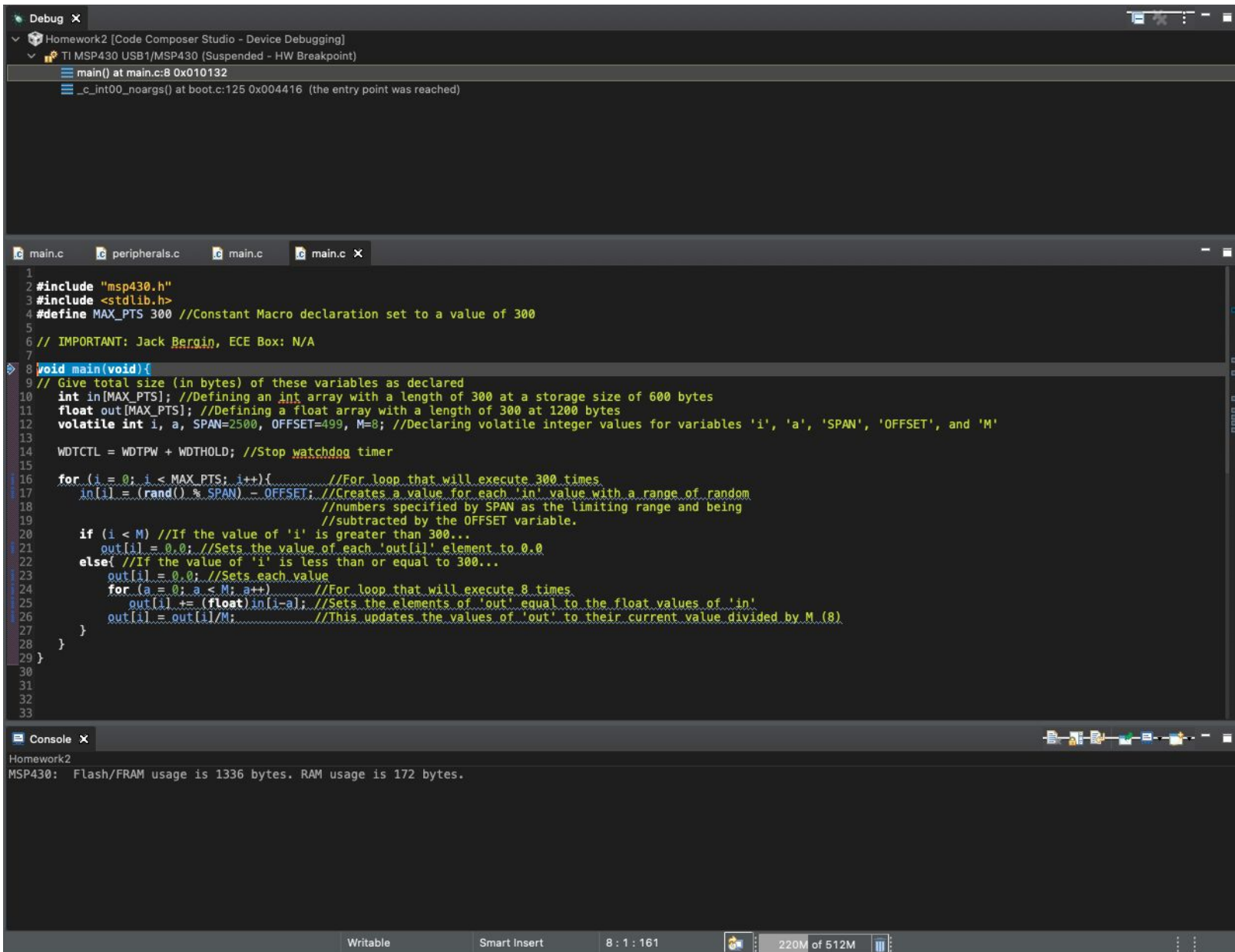
```
1 #include "msp430.h"
2 #include <stdlib.h>
3 #define MAX_PTS 300 //Constant Macro declaration set to a value of 300
4
5 // IMPORTANT: Jack Bergin, ECE Box: N/A
6
7 void main(void){
8 // Give total size (in bytes) of these variables as declared
9 int in[MAX_PTS]; //Defining an int array with a length of 300 at a storage size of 600 bytes
10 float out[MAX_PTS]; //Defining a float array with a length of 300 at 1200 bytes
11 volatile int i, a, SPAN=2500, OFFSET=499, M=8; //Declaring volatile integer values for variables i, a, SPAN, OFFSET, and M
12
13 WDTCTL = WDTPW + WDTHOLD; //Stop watchdog timer
14
15 for (i = 0; i < MAX_PTS; i++){ //For loop that will execute 300 times
16     in[i] = (rand() % SPAN) - OFFSET; //Creates a value for each 'in' value with a range of random
17                                     //numbers specified by SPAN as the limiting range and being
18                                     //subtracted by the OFFSET variable.
19
20     if (i < M) //If the value of 'i' is greater than 300...
21         out[i] = 0.0; //Sets the value of each 'out[i]' element to 0.0
22     else{ //If the value of 'i' is less than or equal to 300...
23         out[i] = 0.0; //Sets each value
24         for (a = 0; a < M; a++) //For loop that will execute 8 times
25             out[i] += (float)in[i-a]; //Sets the elements of 'out' equal to the float values of 'in'
26         out[i] = out[i]/M; //This updates the values of 'out' to their current value divided by M (8)
27     }
28 }
29 }
```

Part B)

```
1 #include "msp430.h"
2 #include <stdlib.h>
3
4 #define MAX_PTS 300    //Constant Macro declaration set to a value of 300
5 #define M 8           //Constant Macro declaration set to a value of 8
6 #define SHIFT 3       //Constant Macro declaration set to a value of 3
7 #define SPAN 2500     //Constant Macro declaration set to a value of 2500
8 #define OFFSET 499    //Constant Macro declaration set to a value of 499
9
10 // IMPORTANT: Jack Bergin, ECE Box: N/A
11
12 void main(void){
13 // Give total size (in bytes) of these variables as declared
14 int in[MAX_PTS];    //Defining an int array with a length of 300 at a storage size of 600 bytes
15 int out[MAX_PTS];   //Defining an int array with a length of 300 at a storage size of 600 bytes
16 volatile int i, a;  //Declaring volatile integer values for variables 'i', and 'a'
17 long int sum;       //Declaring long integer values for the variable 'sum'
18
19 WDTCTL = WDTPW + WDTHOLD; //Stop watchdog timer
20
21 for (i = 0; i < MAX_PTS; i++) {    //For loop that will execute 300 times
22     in[i] = (rand() % SPAN) - OFFSET; //Creates a value for each 'in' value with a range of random
23                                     //numbers specified by SPAN as the limiting range and being
24                                     //subtracted by the OFFSET variable.
25     if (i < M)    //If the value of 'i' is greater than 300...
26         out[i] = 0; //The element at position 'out[i]' is equal to zero
27     else
28         //If the value of 'i' is less than or equal to 300...
29         {
30             sum = 0; // 'sum' is set to 0
31             for (a = 0; a < M; a++) //For loop that will run 8 times
32                 sum += in[i-a]; //Increments array 'in' values all into sum
33             out[i] = sum >> SHIFT; //This will shift right 'sum' and sets it to the values of 'out'
34         }
35 }
36 }
```

ii. Run these 2 simple programs in Code Composer Studio. Take a screen capture of each being run in the debugger.

Part A)



The screenshot displays the Code Composer Studio interface. The top panel shows the debug console with the following messages:

```
main() at main.c:8 0x010132
_c_int00_noargs() at boot.c:125 0x004416 (the entry point was reached)
```

The main editor shows the source code for `main.c`:

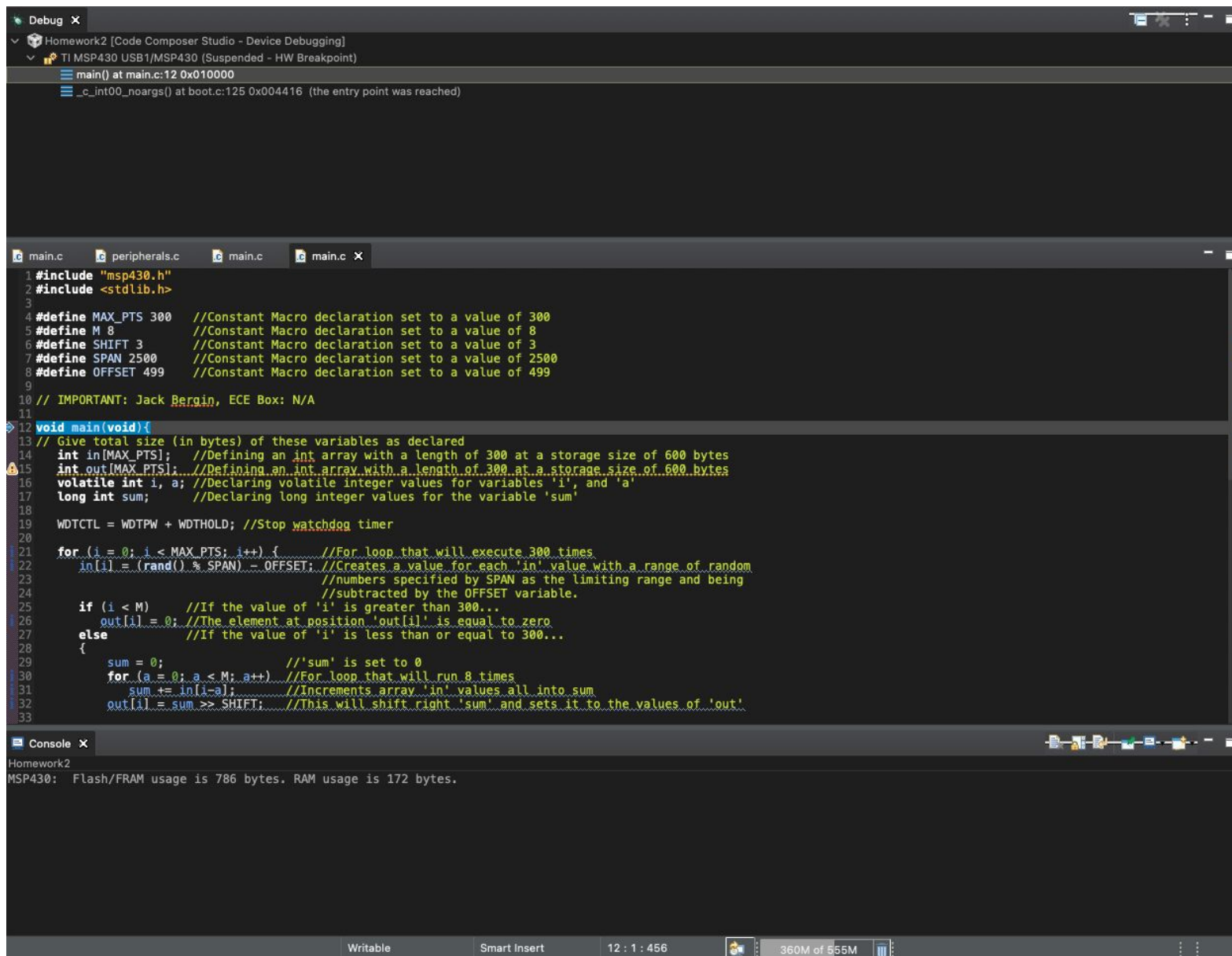
```
1
2 #include "msp430.h"
3 #include <stdlib.h>
4 #define MAX_PTS 300 //Constant Macro declaration set to a value of 300
5
6 // IMPORTANT: Jack Bergin, ECE Box: N/A
7
8 void main(void){
9     // Give total size (in bytes) of these variables as declared
10    int in[MAX_PTS]; //Defining an int array with a length of 300 at a storage size of 600 bytes
11    float out[MAX_PTS]; //Defining a float array with a length of 300 at 1200 bytes
12    volatile int i, a, SPAN=2500, OFFSET=499, M=8; //Declaring volatile integer values for variables 'i', 'a', 'SPAN', 'OFFSET', and 'M'
13
14    WDTCTL = WDTPW + WDTHOLD; //Stop watchdog timer
15
16    for (i = 0; i < MAX_PTS; i++){ //For loop that will execute 300 times
17        in[i] = (rand() % SPAN) - OFFSET; //Creates a value for each 'in' value with a range of random
18                                         //numbers specified by SPAN as the limiting range and being
19                                         //subtracted by the OFFSET variable.
20
21        if (i < M) //If the value of 'i' is greater than 300...
22            out[i] = 0.0; //Sets the value of each 'out[i]' element to 0.0
23        else //If the value of 'i' is less than or equal to 300...
24            out[i] = 0.0; //Sets each value
25            for (a = 0; a < M; a++) //For loop that will execute 8 times
26                out[i] += (float)in[i-a]; //Sets the elements of 'out' equal to the float values of 'in'
27            out[i] = out[i]/M; //This updates the values of 'out' to their current value divided by M (8)
28    }
29 }
30
31
32
33
```

The bottom panel shows the console output:

```
Homework2
MSP430: Flash/FRAM usage is 1336 bytes. RAM usage is 172 bytes.
```

The status bar at the bottom indicates: Writable, Smart Insert, 8 : 1 : 161, 220M of 512M.

Part B)



The screenshot displays the Code Composer Studio interface. The top panel shows the project 'Homework2' and the target 'TI MSP430 USB1/MSP430'. The main editor shows the file 'main.c' with the following code:

```
1 #include "msp430.h"
2 #include <stdlib.h>
3
4 #define MAX_PTS 300 //Constant Macro declaration set to a value of 300
5 #define M 8 //Constant Macro declaration set to a value of 8
6 #define SHIFT 3 //Constant Macro declaration set to a value of 3
7 #define SPAN 2500 //Constant Macro declaration set to a value of 2500
8 #define OFFSET 499 //Constant Macro declaration set to a value of 499
9
10 // IMPORTANT: Jack Bergin, ECE Box: N/A
11
12 void main(void){
13 // Give total size (in bytes) of these variables as declared
14 int in[MAX_PTS]; //Defining an int array with a length of 300 at a storage size of 600 bytes
15 int out[MAX_PTS]; //Defining an int array with a length of 300 at a storage size of 600 bytes
16 volatile int i, a; //Declaring volatile integer values for variables 'i', and 'a'
17 long int sum; //Declaring long integer values for the variable 'sum'
18
19 WDTCTL = WDTPW + WDTHOLD; //Stop watchdog timer
20
21 for (i = 0; i < MAX_PTS; i++) { //For loop that will execute 300 times
22     in[i] = (rand() % SPAN) - OFFSET; //Creates a value for each 'in' value with a range of random
23                                     //numbers specified by SPAN as the limiting range and being
24                                     //subtracted by the OFFSET variable.
25
26     if (i < M) //If the value of 'i' is greater than 300...
27         out[i] = 0; //The element at position 'out[i]' is equal to zero
28     else //If the value of 'i' is less than or equal to 300...
29     {
30         sum = 0; //sum is set to 0
31         for (a = 0; a < M; a++) //For loop that will run 8 times
32             sum += in[i-a]; //Increments array 'in' values all into sum
33         out[i] = sum >> SHIFT; //This will shift right 'sum' and sets it to the values of 'out'
```

The bottom panel shows the console output:

```
Homework2
MSP430: Flash/FRAM usage is 786 bytes. RAM usage is 172 bytes.
```

The status bar at the bottom indicates 'Writable', 'Smart Insert', '12 : 1 : 456', and '360M of 555M'.

iii. Comment on the efficiency (both memory space and run-time) of the integer math program vs the floating point program.

With the use of the float point program, the result is a larger file stored in FLASH when compared to the integer math program. Because this file is larger it will take more time to parse through and execute so the integer math program will be more efficient in both its memory space and its runtime. This difference in size and runtime can be attributed to the makeup of the MSP430 itself. While the device can provide support for float variables it is through extra software which takes more time to run as opposed to integer variables and their prospective operations.

3) Write the following functions using CCS C to complete the code framework below. You do not have to compile this code, but your code must be typed to be graded. (20 pts)

```
/*Libraries that must be included prior to each part*/
#include "msp430.h"
#include <stdlib.h>
/* Function declarations */
void setupP2_P3();
void in2_out3();
```

(a) A function setupP2_P3() that selects P3.7-4 for digital IO with those bits set as outputs and selects P2.7-4 for digital IO with those bits set as inputs. The inputs P2.7-4 are connected to external switches and require internal pull-down resistors. When the switch is open, a logic 0 is input and when the switch is closed, a logic 1 is input. None of the settings for P2 or P3 should be altered. Default drive strength is fine.

```
void setupP2_P3() {
    //Setup for P3
    P3SEL &= ~(BIT7 | BIT6 | BIT5 | BIT4); // Bit set to zero for digital I/O
    P3DIR |= (BIT7 | BIT6 | BIT5 | BIT4);    // Bit set to one for output

    //Setup for P2
    P2SEL &= ~(BIT7 | BIT6 | BIT5 | BIT4); // Bit set to zero for digital I/O
    P2DIR &= ~(BIT7 | BIT6 | BIT5 | BIT4); // Bit set to zero for inputs
    P2REN |= (BIT7 | BIT6 | BIT5 | BIT4); // This enables P2 Pull resistors
    P2OUT &= (BIT7 | BIT6 | BIT5 | BIT4); // Sets the pull resistors as pull downs
}
```

(b) A function in2_out3() that reads in from P2.7-4 to determine which switches are open and which are closed. The function then outputs a logic 1 on P3.7-4 if the switch on the correspond port 2 input is closed and a logic 0 if it is open (i.e. P2 pin 4 input corresponds to P3.4 output, P2.5 → P3.5, etc.).

```
void in2_out3() {
    char input;
    input = P2IN & (BIT7 | BIT6 | BIT5 | BIT4); // AND'd P2IN and
    P3OUT = P3OUT & ~(BIT7 | BIT6 | BIT5 | BIT4)
    P3OUT = P3OUT | input;
    return(P3OUT);
}
```

(c) Write a simple main() that sets up any variables and shows how your functions would be used. What value would result in P3OUT in your in2_out3() function if P2IN =0xB4? If P2IN =0x8F? Does it matter what the values of P3IN or P2OUT are? Explain.

```
/* Compiler directives (includes and defines) */
#include "msp430.h"
/* Function prototypes */
void setupP2_P3();
void in2_out3();
/** Implement your functions here ***/ ..
/* Write your main() here */
void main(){
    setupP2_P3();
    in2_out3();
    //Be sure to clear the timer
    WDTCTL = WDTPW | WDTHOLD;
}
```

```
/*
The values of P3IN and P2OUT matter because if P2IN was 0xB4, P3OUT would be 1011 xxxx
and if P2IN was 0x8F, P3OUT would be 1000 xxxx so more importantly it matters what P3IN is
because that directly impacts the value of P2OUT.
*/
```

4) Our MSP430F5529 Launchpad-based lab board has several digital IO devices on it including 2 LEDs, 2 push buttons, and a 3 x 4 keypad. (20 pts)

(a) Referring to resources like the MSP430F5529 Launchpad User's Guide and posted schematics for our lab board, fill out a table like the one below listing all the ports, pins, and MSP430F5529 package pins to which each of these devices is connected. This will be very useful to have in labs. (5 pts)

Device	I/O Port and Pins	MSP430F5529 Package Pins
2 Launchpad User Buttons	P2.1, P1.1	30, 22
2 Launchpad Used LEDs	P4.7, P1.0	54, 28
Keypad (shown below)	P2.5, P2.4, P1.5, P4.3, P1.2, P1.3, P1.4	34, 44, 26, 48, 23, 24, 25

(b) Are the 2 user LEDs on the Launchpad inputs or outputs? How do you know? What logic level (1 or 0) will cause the user LEDs to light? Explain your answer. (2.5 pts)

(c) Here is the keypad configuration function and part of the getKey function which from the demo project and how column 1 of the keypad is connected to the MSP430F5529.

```
void configKeypad(void) {  
    // Configure digital IO for keypad // smj -- 27 Dec 2015  
    // Col1 = P1.5 = // Col2 = P2.4 = // Col3 = P2.5 = // Row1 = P4.3 = // Row2 = P1.2 = // Row3 =  
    P1.3 = // Row4 = P1.4 =  
    // Select pins for digital IO  
    P1SEL &= ~(BIT5|BIT4|BIT3|BIT2); P2SEL &= ~(BIT5|BIT4);  
    P4SEL &= ~(BIT3);  
    // Columns are ?  
    P2DIR |= (BIT5|BIT4); P1DIR |= BIT5;  
    P2OUT |= (BIT5|BIT4); //  
    P1OUT |= BIT5; //  
    // Rows are ?  
    P1DIR &= ~(BIT2|BIT3|BIT4); P4DIR &= ~(BIT3);  
    P4REN |= (BIT3); //  
    P1REN |= (BIT2|BIT3|BIT4); P4OUT |= (BIT3); //
```

```

P1OUT |= (BIT2|BIT3|BIT4);
}
From getKey()
P1OUT &= ~BIT5; P2OUT |= BIT4; P2OUT |= BIT5;
if ((P4IN & BIT3)==0)
ret_val = '3'; if ((P1IN & BIT2)==0)
ret_val = '6'; if ((P1IN & BIT3)==0)
ret_val = '9'; if ((P1IN & BIT4)==0)
ret_val = '#'; P2OUT |= BIT5;

```

Fully describe the operation of the keypad. Indicate whether the various pins P1.5, P2.4, P2.5, P4.3, P1.2, P1.3 and P1.4 are inputs or outputs. On which ports and pins are the pull resistors being used. Are they pull up or pull down. Why are they necessary? Fully describe how the column read (from getKey) works. To what logic value (0 or 1) must P1.5 be set to be able to detect a key press in one of the rows in column 1. Explain. (10 pts)

The columns of P1.5, P2.4, and P2.5 are digital outputs while the rows of P4.3, P1.2, P1.3, and P1.4 are digital inputs. The digital inputs use the internal pull-up resistors to keep the input pins at 1 (logic 1) when a button is not pressed. But, when a button is pressed, the input reads the logic of the output on the column line for output values.

The getKey() method assigns P2.5 and P2.4 to 1 while setting P1.5 to logic 0. P1.5 is also set to logic 0. P1.5 is in column 1 and due to the row inputs defaulting to logic 1, logic 0 is needed because it signifies that a button is pressed. When P1.5 is set to logic 0 it allows the receiving keypad input from column 1 and is able to disable the other columns. After this is done, each row value is read and if any of the inputs are equal to 0 for logic, the key that corresponds with that row of column 1 is pressed. After this process, the P1.5 logic is reset to 1 in order for the next column to read through. This process is repeated for each column.

5) Assume that 4 slide switches like the one shown below are to be connected to the pins of Port 2 pins 6-3 such that when the switch is slid to the right it connects that pin to GND (0V) and when slid to the left it connects to Vcc (3.3V). (20 pts)

(a) Would these switches be digital inputs or outputs? How would they work (i.e. what switch operation would correspond to digital 1 or digital 0)?

The switch is an input because it provides information to execute an action. When the switch is thrown on it gives a digital 1 and when it is thrown off it gives a digital 0. From there outputs can be implemented based on whether the digital output is a 1 or a 0.

(b) Describe an example application for these switches. How might they be used? How is their functionality different from the push buttons (i.e. Why would you choose to use slide switches instead of buttons)?

An application for these switches would be to replace a toggle button if the button needed to be pressed for a long period of time. If, for example, one is trying to turn on a pair of headphones to use they want to switch the switch in the on position and then have the device continue to stay on. If a button were used instead, it could still complete the task but just not well. It is solely a time requirement that would determine the use of a button or a switch.

(c) Write 2 functions in Code Composer Studio C to configure and use the switches and a simple main which uses your functions. The function switchConfig() should configure P2.6-3 for digital IO. You should leave Drive Strength in its default configuration and disable pull up/ pull down resistors. The settings for all other Port 2 pins should not be altered. The function switchIO() should return a value between 0-F hex which corresponds to the switch settings assuming P2.3 is the LSB and P2.6 is the MSB. You do not have to compile this code but it does need to be typed.

```
/* Function prototypes */
```

```
void switchConfig();
```

```
char switchIO();
```

```
P2REN = 0;
```

```
void main(void) {
```

```
    switchConfig();
```

```
    switchIO();
```

```
}
```

```
void switchConfig(){
```

```
    P2SEL = P2SEL & ~( BIT6 | BIT5 | BIT4 | BIT3 ); // Set as digital I/O
```

```
    P2DIR = PDIR & ~( BIT6 | BIT5 | BIT4 | BIT3 ); //Set as input
```

```
    P2REN = 0; //Turns off the pull up/down resistors
```

```
}
```

```
char switchIO() { //This method will return a value between 0-F hex
    char bitsIn, bitsOut = 0;
    bitsIn = P2IN & ( BIT6 | BIT5 | BIT4 | BIT3 ); //Configures P2.6-3 for digital I/O
    bitsOut = (bitsIn >> 3); //Shift Bits to the right into low nibble
    return bitsOut;
}
```

(d) Assume your program has properly configured the 4 slide switches for digital IO. How would your functions interpret the following settings of the Port 5 input and output registers P2IN = 0xB5 and P2OUT = 0x10, and P2IN = 0x6C and P2OUT = 0x08? What would the function switchIO() return in each case?

The functions would recognize P2IN = 0xB5 and P2OUT = 0x10 as input pins because it describes the correct directory while the statement P2IN = 0x6C and P2OUT = 0x08 would through an error because 0x6C is not the proper address. The switchIO() function would return the B5 hex value for the first case and an error for the second case.