ME333 – Homework 4 – Yael Ben Shalom

Chapter 5, Exercise 3:

The code for this exercise attached in '/code/Ch5Q3' folder.

a) The combinations of data types and arithmetic functions that result in a jump to a subroutine are long long int, and division:

- Function for long long int with division:

  The C statement:

  `j3 = j1 / j2;`

  The assembly commands it expands to:

```
  j3 = j1 / j2;
9d001fd0: 8fc40088  lw  a0,136(s8)
9d001fd4: 8fc5008c  lw  a1,140(s8)
9d001fd8: 8fc60090  lw  a2,144(s8)
9d001fdc: 8fc70094  lw  a3,148(s8)
9d001fe0: 0f4009de  jal 9d002778 <__divdi3>
9d001fe4: 00000000  nop
9d001fe8: 00400013  mtlo  v0
9d001fec: 00600011  mthi  v1
9d001ff0: 00001012  mflo  v0
9d001ff4: 00001810  mfhi  v1
9d001ff8: afc20010  sw  v0,16(s8)
9d001ffc: afc30014  sw  v1,20(s8)
}
9d002000: 03c0e821  move  sp,s8
9d002004: 8fbf0084  lw  ra,132(sp)
9d002008: 8fbe0080  lw  s8,128(sp)
9d00200c: 27bd0088  addiu sp,sp,136
9d002010: 03e00008  jr  ra
9d002014: 00000000  nop
```

- Function for float with all operations.
- Function for long double with all operations.

b) For the statements that do not result in a jump to a subroutine, the combination(s) of data types and arithmetic functions that result in the fewest assembly commands are int with addition and subtraction:

```
  i3 = i1 + i2;
9d001ea8: 8fc30070  lw  v1,112(s8)
9d001eac: 8fc20074  lw  v0,116(s8)
9d001eb0: 00621021  addu  v0,v1,v0
9d001eb4: afc20000  sw  v0,0(s8)
  i3 = i1 - i2;
9d001eb8: 8fc30070  lw  v1,112(s8)
9d001ebc: 8fc20074  lw  v0,116(s8)
9d001ec0: 00621023  subu  v0,v1,v0
9d001ec4: afc20000  sw  v0,0(s8)
```

The smallest data type, char, does not involve in it, because it has an additional line "andi" for AND operation between v0 and 0xFF:

```
  c3 = c1 + c2;
9d001e20: 93c30070  lbu v1,112(s8)
9d001e24: 93c20074  lbu v0,116(s8)
9d001e28: 00621021  addu  v0,v1,v0
9d001e2c: 304200ff  andi  v0,v0,0xff
9d001e30: a3c20000  sb  v0,0(s8)
```

c) The filled table:

	Char	Int	long long	Float	long double
+	1.25 (5)	1.0 (4)	2.75 (11)	1.25 (5) J	2.0 (8) J
-	1.25 (5)	1.0 (4)	2.75 (11)	1.25 (5) J	2.0 (8) J
*	1.5 (6)	1.25 (5)	4.75(19)	1.25 (5) J	2.0 (8) J
/	1.75 (7)	1.75 (7) J	3.0 (12) J	1.25 (5) J	2.0 (8) J

d) The math subroutines installed in virtual memory at:

```
kseg0 Program-Memory Usage
section                   address  length [bytes]      (dec)  Description
-------                   ----------  ------------------------  -----------
.text                   0x9d001e00          0x4c0        1216  App's exec code
.text.dp32mul           0x9d0022c0          0x4b8        1208
.text                   0x9d002778          0x4b4        1204  App's exec code
.text.dp32subadd        0x9d002c2c          0x430        1072
.text.dp32mul           0x9d00305c          0x32c         812
.text                   0x9d003388          0x2b4         692  App's exec code
.text.fpsubadd          0x9d00363c          0x278         632
.text.fp32div           0x9d0038b4          0x230         560
.text.fp32mul           0x9d003ae4          0x1bc         444
```

Chapter 5, Exercise 4:

The code for this exercise attached in '/code/Ch5Q4' folder.

Each of the statements uses those command lines:

& - 4 commands

| - 4 commands

<< - 4 commands

>> - 4 commands

```
  u3 = u1 & u2;   // bitwise AND
9d002230: 8fc30000  lw  v1,0(s8)
9d002234: 8fc20004  lw  v0,4(s8)
9d002238: 00621024  and v0,v1,v0
9d00223c: afc20008  sw  v0,8(s8)
  u3 = u1 | u2;   // bitwise OR
9d002240: 8fc30000  lw  v1,0(s8)
9d002244: 8fc20004  lw  v0,4(s8)
9d002248: 00621025  or  v0,v1,v0
9d00224c: afc20008  sw  v0,8(s8)
  u3 = u2 << 4;   // shift left 4 spaces, or multiply by 2^4 = 16
9d002250: 8fc20004  lw  v0,4(s8)
9d002254: 00021100  sll v0,v0,0x4
9d002258: afc20008  sw  v0,8(s8)
  u3 = u1 >> 3;   // shift right 3 spaces, or divide by 2^3 = 8
9d00225c: 8fc20000  lw  v0,0(s8)
9d002260: 000210c2  srl v0,v0,0x3
9d002264: afc20008  sw  v0,8(s8)
```

Chapter 6, Exercise 1:

Pros and cons of using interrupts vs. polling:

Pros	Cons
Interrupt can take place at any time, whereas CPU steadily ballots the device at regular or proper interval	Interrupts are limited
In interrupts, processor is simply disturbed once any device interrupts it. On the opposite hand, in polling, processor waste countless processor cycles by repeatedly checking the command-ready little bit of each device	Requires special non-cacheable memory allocation for SFRs
Interrupts are faster	

Chapter 6, Exercise 4:

a) If an IRQ is generated for an ISR at priority level 4, subpriority level 2 while the CPU is in normal execution, the CPU will attend the ISR.

b) If that IRQ is generated while the CPU is executing a priority level 2, subpriority level 3 ISR, the CPU will attend the level 4 ISR and then jump back to the level 2 ISR.

c) If that IRQ is generated while the CPU is executing a priority level 4, subpriority level 0 ISR, the CPU will continue attending the current level 4 ISR before moving to the new level 4 ISR.

d) If that IRQ is generated while the CPU is executing a priority level 6, subpriority level 0 ISR, the CPU will continue attending the level 6 ISR before moving to the level 4 ISR.

Chapter 6, Exercise 5:

a) Assuming no shadow register set, the first thing the CPU must do before executing the ISR is to save the contents of the internal CPU registers, called the "context," to the stack (data RAM).

The last thing it must do upon completing the ISR is to restore the contexts back from the RAM into the registers and the CPU reverts to its initial state before the ISR was executed.

b) Using the shadow register set changes the situation by allowing the CPU to use these extra set of registers without needing to save and restore them.

Chapter 6, Exercise 8:

a)
```
IEC0SET = 0x100;        // Enabling the Timer2 interrupt
IFS0CLR = 0x100;        // Setting flag status to 0
IPC2CLR = 0x1F;         // Clearing IPC2 Register
IPC2SET = 0x16;         // Setting priority level to 5, sub priority level to 2
```

b)
```
IEC1SET = 0x8000;          // Enabling the Real-Time Clock and Calendar interrupt
IFS1CLR = 0x8000;          // Setting flag status to 0
IPC8CLR = 0x1F000000;      // Clearing IPC8 Register
IPC8SET = 0x19000000;      // Setting priority level to 6, sub priority level to 1
```

c)
```
IEC2SET = 0x10;            // Enabling the UART4 Receiver interrupt
ISF2CLR = 0x10;            // Setting flag status to 0
IPC12CLR = 0x1F00;         // Clearing IPC2 Register
IPC12SET = 0x1F00;         // Setting priority level to 7, sub priority level to 3
```

d)
```
IEC0SET = 0x800;           // Enabling the INT2 external input interrupt
ISF0CLR = 0x800;           // Setting flag status to 0
IPC2CLR = 0x1F00000;       // Clearing IPC2 Register
IPC2SET = 0xE000000;       // Setting priority level to 3, sub priority level to 2
INTCON = 0x4;              // Configuring to trigger on rising edge
```

Chapter 6, Exercise 9:

The code for this exercise attached in '/code/Ch6Q9' folder.

Chapter 6, Exercise 16:

The code for this exercise attached in '/code/Ch6Q16' folder.

Chapter 6, Exercise 17:

The code for this exercise attached in '/code/Ch6Q17' folder.