5.3)
a.
long long int: division
float: addition, subtraction, multiplication, division
long double: addition, subtraction, multiplication, division

an example using long long int division

j3 = j1/j2;

```
9d002698:    8fc60018    lw      a2,24(s8)
9d00269c:    8fc7001c    lw      a3,28(s8)
9d0026a0:    8fc40010    lw      a0,16(s8)
9d0026a4:    8fc50014    lw      a1,20(s8)
9d0026a8:    0f400780    jal     9d001e00 <__divdi3>
9d0026ac:    00000000    nop
9d0026b0:    afc20020    sw      v0,32(s8)
9d0026b4:    afc30024    sw      v1,36(s8)
```

b. ints with addition or subtraction is the shortest. It is not chars because they must check to see if there is overflow

i3= i1+i2;

```
9d002370:    8fc30000    lw      v1,0(s8)
9d002374:    8fc20004    lw      v0,4(s8)
9d002378:    00621021    addu    v0,v1,v0
9d00237c:    afc20008    sw      v0,8(s8)
```

	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)	3.0 (12)J	1.25(5)J	2.0(8) J

c. addu 9d002378 – 9d001fe8 = 390bits

5.4) AND and OR both use for assembly commands and the shifts only take 3

6.1) Interrupts are better for things that happen infrequently because it can save the amount of memory used by the cpu itself and cpu cycles. Polling is better when the data is continuous, so you don't have to reconfigure the interrupt each time an even happens.

6.4) a) does the ISR

b) continues original ISR then after it is finished does the second one

c) continues original ISR then after it is finished does the second one

d) CPU does the priority 4 then goes back to the 6

6.5)

a) First the data will be copied to ram and after the interrupt it will be copied back to the memory in the cpu

b) The SRS is an identical set of cpu storage so the CPU can swith to that memory to do the iSR then move back so it doesn't have to move the data to the ram and back.

6.8)

a) Enable the Timer2 interrupt, set its flag status to 0, and set its vector's priority and subpriority to 5 and 2, respectively.

```
IEC0SET = 0x8;
IFS0CLR = 0x8;
IPC2SET = 0x16;
```

b) Enable the Real-Time Clock and Calendar interrupt, set its flag status to 0, and set its vector's priority and subpriority to 6 and 1, respectively.

```
IEC1SET = 0xF;
IFS1CLR = 0xF;
IPC2SET = 0x1D000000;
```

c) Enable the UART4 receiver interrupt, set its flag status to 0, and set its vector's priority and subpriority to 7 and 3, respectively.

```
IEC2SET = 0x4;
IFS2CLR = 0x4;
IPC2SET = 0x1F00;
```

d) Enable the INT2 external input interrupt, set its flag status to 0, set its vector's priority and subpriority to 3 and 2, and configure it to trigger on a rising edge.

```
IEC0SET = 0xB;
IFS0CLR = 0xB;
IPC2SET = 0xE000000;
INTCONSET = 0x4;
```

6.9)

```
INTCONSET = 0x3;                  // step 3: INT0 and INT1 trigger on rising edge
  IPC0CLR = 31 << 24;               // step 4: clear 5 priority and subp bits for INT0
```

```
  IPC0 |= 24 << 24;                // step 4: set INT0 to priority 6 subpriority 0
  IPC1CLR = 0x1F << 24;            // step 4: clear 5 priority and subp bits for INT1
  IPC1 |= 0x18 << 24;              // step 4: set INT1 to priority 6 subpriority 0
  IFS0bits.INT0IF = 0;             // step 5: clear INT0 flag status
  IFS0bits.INT1IF = 0;             // step 5: clear INT1 flag status
  IEC0SET = 0x88;                  // step 6: enable INT0 and INT1 interrupts
```

6.16)

```
#include "NU32.h"          // constants, funcs for startup and UART

void __ISR(_EXTERNAL_0_VECTOR, IPL2SOFT) Ext0ISR(void) { // step 1: the ISR
  int count = 0;

  while (count <= 400000 && INTCONbits.INT0EP == 0) {
    count++;
  }

  if (count > 400000) {
    NU32_LED1 = 0;
    NU32_LED2 = 0;
    _CP0_SET_COUNT(0);

    while(_CP0_GET_COUNT() < 10000000) { ; } // delay for 10 M core ticks, 0.25 s

    NU32_LED1 = 1;
    NU32_LED2 = 1;
  }

  IFS0bits.INT0IF = 0;             // clear interrupt flag IFS0<3>
}
Changed above
Unchanged below

int main(void) {
  NU32_Startup(); // cache on, min flash wait, interrupts on, LED/button init, UART
init
  __builtin_disable_interrupts(); // step 2: disable interrupts
  INTCONbits.INT0EP = 0;          // step 3: INT0 triggers on falling edge
  IPC0bits.INT0IP = 2;            // step 4: interrupt priority 2
  IPC0bits.INT0IS = 1;            // step 4: interrupt priority 1
  IFS0bits.INT0IF = 0;            // step 5: clear the int flag
  IEC0bits.INT0IE = 1;            // step 6: enable INT0 by setting IEC0<3>
  __builtin_enable_interrupts();  // step 7: enable interrupts
                                  // Connect RD7 (USER button) to INT0 (RD0)
  while(1) {
      ; // do nothing, loop forever
  }
```

```
  return 0;
}
```

6.17) code and demo turned in