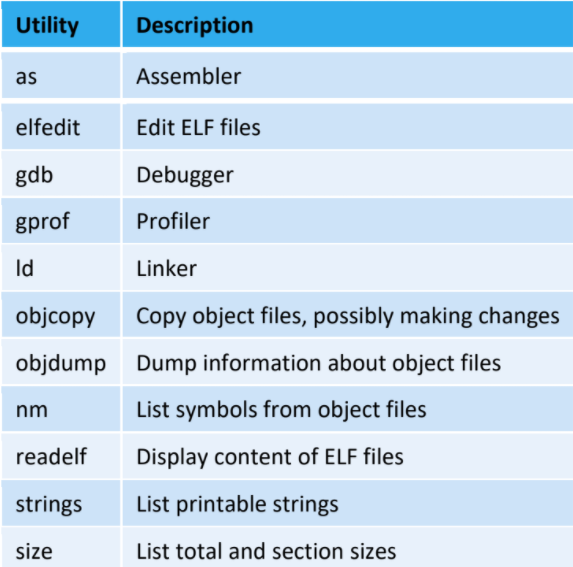
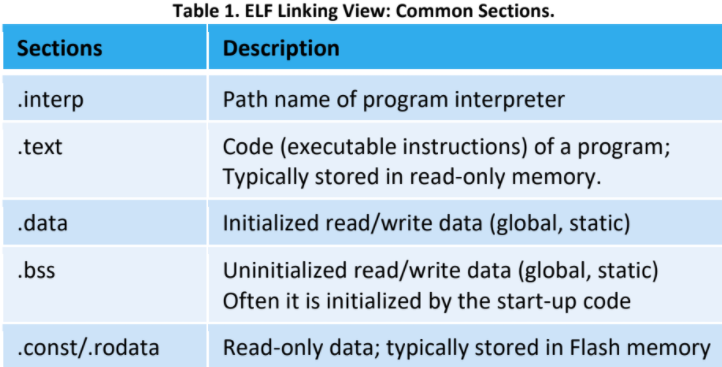
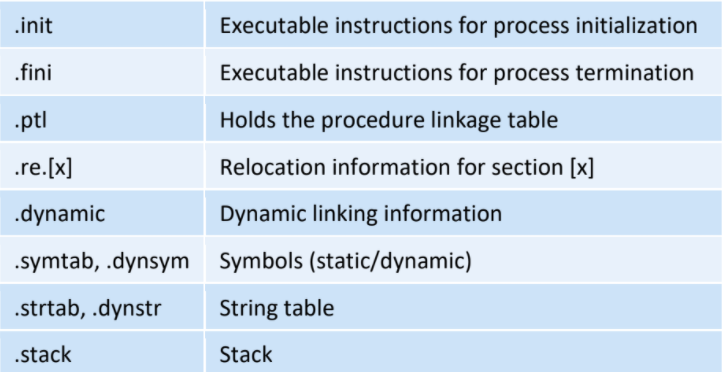
1. **Know what the utilities mentioned in the tutorial do, or what kind of information they provide. Given a scenario, you must be able to choose the correct tool.**
   1. msp430-elf-objdump.exe
      1. Disassembler, display information from object files
   2. msp430-elf-readelf.exe
      1. Display information about the contents of ELF format files
   3. MSP430Flasher.exe
      1. Loads binary files into memory
   4. naken\_util.exe
      1. Disassembler
   5. GNU Utilities graph from tutorial: strip Remove symbols from an object file.



* 1. ELF sections and what they mean:

1. **Know how to figure out the instruction size by looking at the disassembly code.**
   1. ? not sure
2. **What do different memory segments (.data, .text, .bss, .stack, etc.) signify?**
   1. .text -- Used for program code.
   2. .bss -- Used for uninitialized objects (global variables).
   3. .data -- Used for initialized non-const objects (global variables).
   4. .const -- Used for initialized const objects (string constants, variables declared const).
   5. .cinit -- Used to initialize C global variables at startup.
   6. .stack -- Used for the function call stack.
   7. .sysmem - Used for the dynamic memory allocation pool.
3. **Know how to work with (set direction, turn on/off, etc.) with the LEDs and switches in both assembly AND C.**
   1. Assembly:

|  |
| --- |
| bis.b #0x01, &P1DIR ; Set P1.0 as output, 0'b0000 0001  bis.b #0x80, &P4DIR ; Set P4.7 as output, 0'b1000 0000  bic.b #0x01, &P1OUT ; Turn P1.0 off.  bic.b #0x80, &P4OUT ; Turn P4.7 off.  **; Setting Switch 2's data (i/o).**  bic.b #0x02, &P1DIR ; Set P1.1 as input for SW2  bis.b #0x02, &P1REN ; Enable Pull-Up resister at P1.1  bis.b #0x02, &P1OUT ; required for proper IO set up  **; Setting Switch 1's data (i/o).**  bic.b #0x02, &P2DIR ; Set P2.1 as input for SW1  bis.b #0x02, &P2REN ; Enable Pull-up resistor at P2.1  bis.b #0x02, &P2OUT ; Required for proper IO setup.  xor.b #0x80, &P4OUT ; Toggle P4.7  bic.b #0x01, &P1OUT ; Turn P1.0 off.  bis.b #001h, &P1OUT ; Turn on LED1 |

* 1. C:

|  |
| --- |
| #define RED 0x01        // Red LED Pin  #define GREEN 0x80      // Green LED Pin  P1DIR |= RED;           // P1.0 is output direction for REDLED  P1REN |= BIT1;          // Enable the pull-up resistor at P1.1  P1OUT &= ~RED;          // LED is off at start  P4DIR |= GREEN;         // P4.7 is output direction for GREENLED  P2REN |= BIT1;          // Enable the pull-up resistor at P2.1  P4OUT &= ~GREEN;        // LED is off at start  P4OUT ^= GREEN;         // Toggle green LED  P1OUT ^= RED;           // Toggle the red LED  P1OUT |= RED; // Turn on RED LED.  P4OUT |= GREEN; // Turn on green LED  #define SW1 ((P2IN&BIT1)== 0)  #define SW2 ((P1IN&BIT1)== 0)  P2DIR &= ~BIT1;                 // Configuring Switch 1  P2REN |= BIT1;  P2OUT |= BIT1;  P1DIR &= ~BIT1;                 // Configuring Switch 2  P1REN |= BIT1;  P1OUT |= BIT1;  if (SW1)                     // If switch 1 is pressed |

1. **Make sure you know how to manipulate different ports in BOTH C and Assembly, and what are the functions of different registers (PxDIR, PxOUT, PxIN, etc.)?**
   1. Assembly:

|  |
| --- |
| bis.b #0x01, &P1DIR ; Set P1.0 as output, 0'b0000 0001  bis.b #0x80, &P4DIR ; Set P4.7 as output, 0'b1000 0000  bic.b #0x01, &P1OUT ; Turn P1.0 off.  bic.b #0x80, &P4OUT ; Turn P4.7 off. |

* 1. C:

|  |
| --- |
| #define RED 0x01  #define GREEN 0x80 // LED 1 = RED, LED 2 = GREEN  P1DIR |= RED; // P1.0 is output direction for 0x01, or red.  P1REN |= BIT1; // Enable the pull-up resistor at P1.1  P1OUT |= BIT1; // LED is on.  P4DIR |= GREEN; // P4.7 is output for 0x80, or green.  P2REN |= BIT1; // Enable pull-up resistor at P2.1  P2OUT |= BIT1; // LED is on |

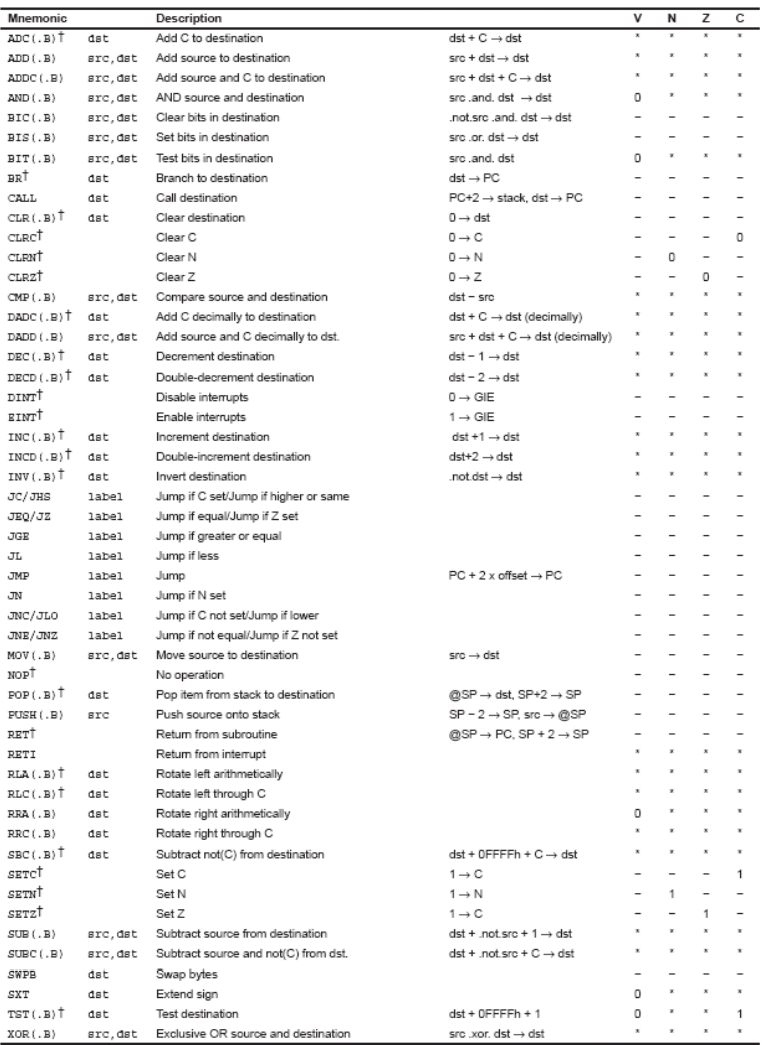
* 1. Functions of different registers
     1. **PxIN** - input register, reading it returns the logical values on the pins (determined by the external signals). These registers are read-only and bit value of 0 indicates that the corresponding input is low and bit value of 1 indicates that the input is high.
     2. **PxOUT** - output register, writing it sends the value to the corresponding port pin when the pin is configured for the I/O function with output direction. Bit value of 0 will produce low output voltage and bit value of 1 will produce high output voltage.
     3. **PxDIR** - direction register, configures the direction of the corresponding I/O pins (e.g., P2DIR=0xFC = 111111100b configures bits 1 and 0 of port P2 as input pins and all other pins are outputs).
     4. **PxSEL** - selection register, setting bits in this register allows the user to change the port pin function from the standard digital I/O to its corresponding special function. MSP430 interfaces external world predominantly through parallel ports and their default operation is a standard digital input/output (PxSEL=0x00). However, some of the pins have an alternative special function – e.g., they can act as an analog input channel (A0) to the analog-to-digital converter or serial data output of a serial communication interface (TDO). The reference manual specifies special functions for each port pin. They are highly device-specific – developers have to consult the reference manual for the microcontroller they are using.
     5. **PxREN** – enables the pull-up or pull-down resistor configuration (e.g, P2REN = 0x02 enables the pull-up resistor on P2.1 that is connected in a series with switch SW1 on the MSP-EXP430F5529LP board.)
     6. Ports P1 and P2 also have ability to serve as sources of interrupts and several registers are associated with this function. These are:
        1. **PxIE** – Port x Interrupt Enable register for enabling/disabling interrupts (x=1, 2),
        2. **PxIFG** – Port x Interrupt Flag register for tracking pending requests,
     7. **PxIES** – Port x Interrupt Edge Select register for selecting type of event that triggers an interrupt – rising edge at the port input (0 -> 1) or falling edge (1 -> 0);
     8. **PxIV** – Port x Interrupt Vector Word. All interrupts associated with a single port share a single interrupt service routine. The highest priority enabled pending interrupt request generates a number in the PxIV register. This number can be used by the code in the corresponding interrupt service routine to speed up interrupt processing.

1. **How to set a bit to 0 or 1 without affecting other bits (bit masking) in BOTH C and assembly**
   1. Set a specific bit: Use the or operator: **|=**
   2. Clear a specific bit: Use the and operator: **&=**
   3. Toggling a bit: use the XOR operator: **^=**

|  |
| --- |
| P4OUT ^= GREEN; // toggle the green LED  P4OUT |= GREEN; // Green LED is on,  P1OUT &= ~RED; // Red LED is off |

* 1. Assembly:
  2. Set bits in destination: BIC (.B) src, dst ; scr .or .dst->dst
  3. Clear bits in destination: BIC (.B) src, dst ; not.src and dst -> dst
  4. Toggle bits in destination: XOR (.B) src, dst ; src xor dst -> dst

|  |
| --- |
| xor.b #0x80, &P4OUT ; Toggle P4.7  bic.b #0x01, &P1OUT ; Turn P1.0 off.  bis.b #001h, &P1OUT ; Turn on LED1 |



1. **How to work with the switches and LEDs? What values would they return if they are pressed? What about when they are not pressed?**
   1. From the schematic we see that if we want LED1 on, **we should provide a logical '1' at the output port of the microcontroller (port P1.0), and a logical '0' if we want LED1 to be off.** We could take several approaches to solving this problem. Figure 2 illustrates one such approach - after initializing the port P1.0 as output (P1DIR=00000001), setting P1.0 to logic '1', the program will spend all its time in an infinite loop (Figure 2).
   2. If it has indeed been pressed (bit 1 of P2IN is 0) – i.e. The button pressed returns 0 and the button not pressed returns 1,=.
2. **How to calculate delays and how they are affected by different components (clock frequency, loop instruction, loop upper and lower limit, etc.)**
   1. delay of 16cc so the total delay is 65535\*16cc/2^20 ~= 1s
   2. Worst-case settling time for the DCO when the DCO range bits have been changed is n x 32 x 32 x f\_MCLK / f\_FLL\_reference. See UCS chapter in 5xx UG for optimization. 32 x 32 x 2.45 MHz / 32,768 Hz = 76600 = MCLK cycles for DCO to settle
3. **How would you clear an interrupt flag?**
   1. Switch 1 Press example (C):

|  |
| --- |
| 1. // Switch 1 Press 2. #pragma vector = PORT2\_VECTOR 3. \_\_interrupt void PORT2\_ISR(void) 4. { 5. WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer 6. P1OUT &= ~RED;              // Turn Red LED off 7. P4OUT &= ~GREEN;            // Turn Green LED off 8. P2IFG &= ~BIT1;             // Clear interrupt flag. 9. } |

* 1. SW2\_ISR (assembly):

|  |
| --- |
|  |