SYSC3310 Lab 2 – Debugging and Bit Manipulations

Fall 2018

Objectives:

- Continued practice in writing C programs for the embedded target MSP432P401R Launchpad, to re-inforce the practices of creating a project, configuring the debugger, building a program and downloading the program to the target.
- Exercises in C programming that will underpin later work: memory-specific data lengths and bit manipulations
- Use of the debugger: breakpoints and stepping

Submission: lab2.c

Equipment:

- MSP432 P401R LaunchPad Development Kit

References and Reading Material

- Lab 1 – For reference on how to create a project and configure the debugger
- CMSIS-DAP Debugger Manual: http://www.keil.com/support/man/docs/dapdebug/dapdebug_ctx_trace.htm

Part A: Preparing the Project

Using Lab 1 to remind you of the details, create a new project called **Lab2**. Prepare the minimal C program and test that your program builds correctly.

Part B: Bit Manipulations

Write a C program that does the following sequence of bit manipulations on two variables: `aByte` is an 8-bit unsigned integer and `aWord` is a 16-bit unsigned integer.

- The manipulations are in sequence: the second follows the first, the third follows the second. Or in other words, the lines of code depend on the lines above them.
- Each manipulation must be written in one line of code
- Beside each line of code, write the current hexademical contents of the affected variable
  - Take the time to figure out what you expect happens after each instruction. Don't worry – it's not being marked and you'll get a chance to fix any errors before you submit. It is however important for your learning to try these out in your head first.

1. Initialize `aByte` to 0x33
2. Set Bits 0 and 7 on `aByte`
3. Clear Bits 0 and 7 on `aByte`, using & only
4. Clear Bit 7 on `aByte`, using both & and ~ operators in the same statement.
5. Shift the bits of `aByte` left by 5 places
6. Shift the bits of `aByte` right by 5 places
7. Initialize `aWord` to zero
8. Set the high byte of `aWord` to all ones
9. OR the contents of `aWord` with the contents of `aByte`
10. You've given the code for this next step. In the comment (in addition to the current contents of `aWord`) you must describe what the code does (in a similar fashion to Steps 1 to 9)

    ```
    aWord &= ~(aByte << 8);
    ```

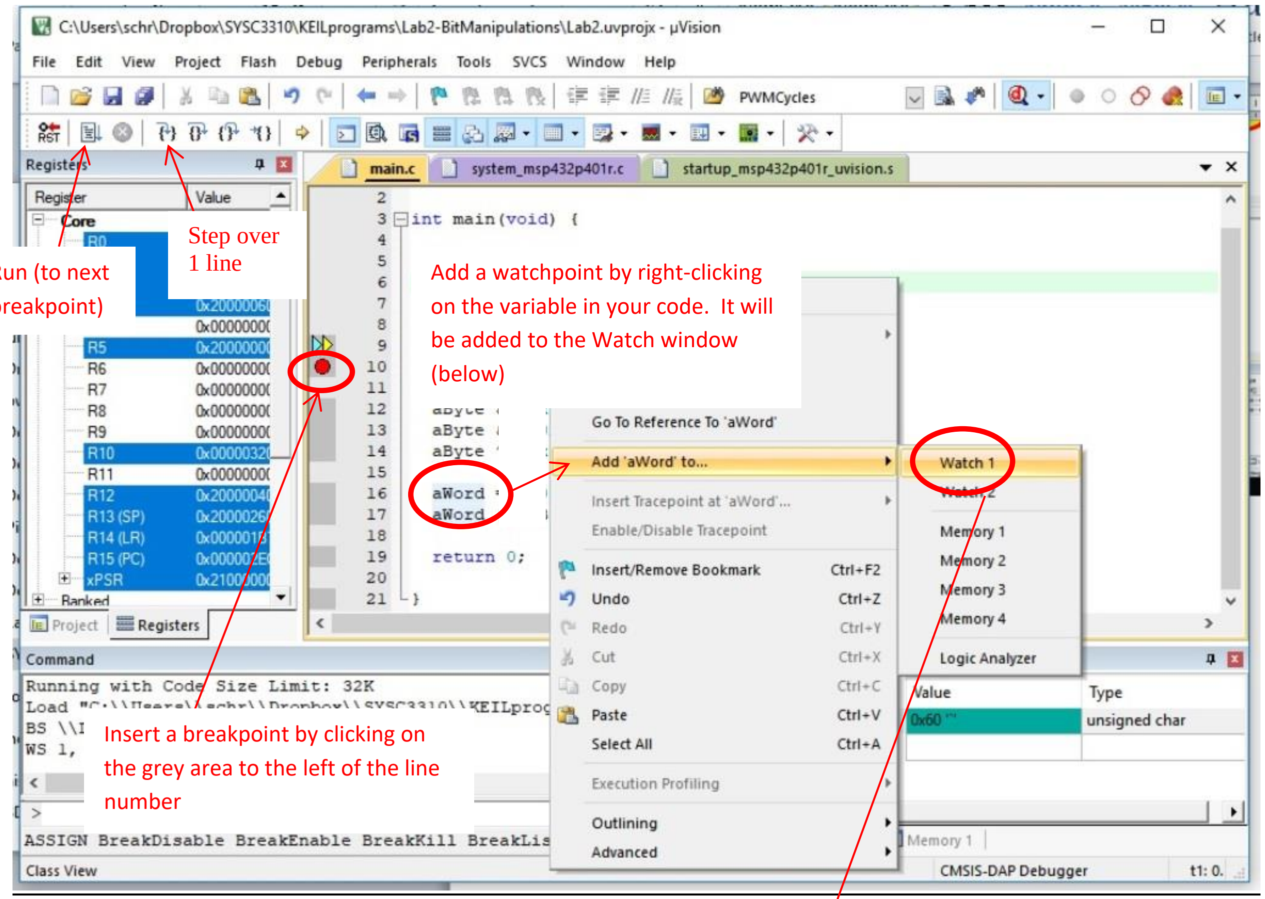
11. Shift the contents of `aWord` 8 bits to the right.

Build your program and download it for running.

Part C: Debugging

You will see if your understanding of bit manipulations is correct by running the program and comparing the current value of the variable to what you've written in your comments. Of course, you can change the comments if you discover you were wrong!

1. Build your program and download it for running (**Debug->Start/Stop Debug Session**)
2. Prepare your debugging plans (See Figure below, on the next page)
    a. Add in a breakpoint on your first line of code (after the variable declarations)
    b. Add in 2 watchpoints on the two variables `aByte` and `aWord`. The debugger will "watch" these variables and will display their current value whenever they change.
3. Run the program to the first breakpoint. Do not step, because there is a lot of hidden startup code that you want to skip over.
4. Now step instruction by instruction, checking the two watches on the variables to confirm that their contents matches your expectation after each instruction.


C:\Users\schr\Dropbox\SYSC3310\KEILprograms\Lab2-BitManipulations\Lab2.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
PWMCycles
Registers
main.c
system_msp432p401r.c
startup_msp432p401r_uvision.s
Go To Reference To 'aWord'
Add 'aWord' to...
Watch 1
Watch 2
Memory 1
Memory 2
Memory 3
Memory 4
Logic Analyzer
Insert Tracepoint at 'aWord'...
Enable/Disable Tracepoint
Insert/Remove Bookmark Ctrl+F2
Undo Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Select All Ctrl+A
Execution Profiling
Outlining
Advanced
Command
Class View
CMSIS-DAP Debugger

Run (to next breakpoint)

Step over 1 line

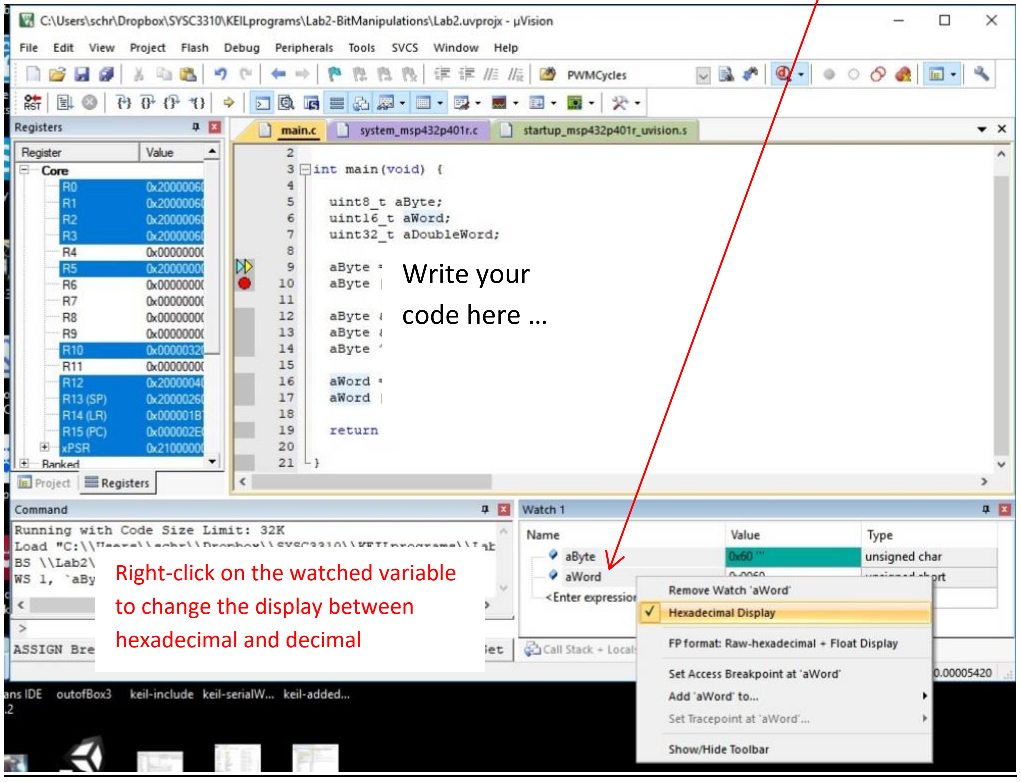
Add a watchpoint by right-clicking on the variable in your code. It will be added to the Watch window (below)

Insert a breakpoint by clicking on the grey area to the left of the line number

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

```
Running with Code Size Limit: 32K
ASSIGN BreakDisable BreakEnable BreakKill BreakLis
```

Value	Type
0x60	unsigned char


C:\Users\schr\Dropbox\SYSC3310\KEILprograms\Lab2-BitManipulations\Lab2.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Watch 1
Remove Watch 'aWord'
Hexadecimal Display
FP format: Raw-hexadecimal + Float Display
Set Access Breakpoint at 'aWord'
Add 'aWord' to...
Set Tracepoint at 'aWord'...
Show/Hide Toolbar

```
int main(void) {

    uint8_t aByte;
    uint16_t aWord;
    uint32_t aDoubleWord;
```

Write your code here …

Name	Value	Type
aByte	0x60	unsigned char
aWord		
<Enter expression		

Right-click on the watched variable to change the display between hexadecimal and decimal

<u>Marking Scheme</u>:

<u>Part A</u>: No submission or demonstration required.
<u>Part B</u> and/or Part C:

<u>Demonstration</u>:

- Demonstration of single stepping through the code
- Able to explain why/how each operation works.

<u>Inspection</u>: 1 mark each

1. All steps are implemented
2. All steps are commented with expected results
3. Overall Style

<u>Overall marks for style (To be used for all labs in this course):</u>

- Comments, indentation, and well-named functions and variables
- Removal of all extra code (no commented out sections, no unused code leftover from some other example)