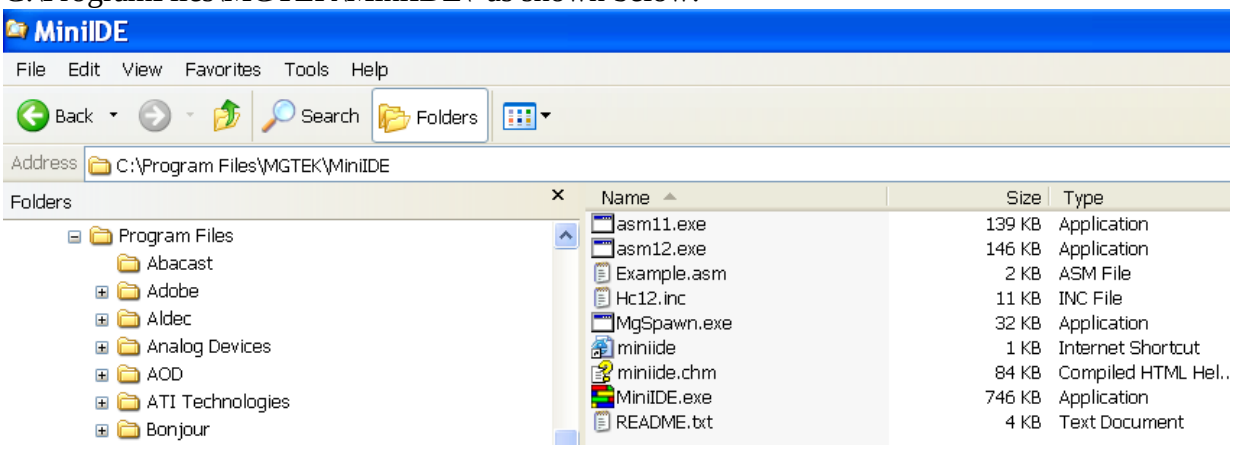
EGR 270
Fundamentals of Computer Engineering
File: Wookie .doc

Example: Mini IDE Assembler and Wookie Simulator

MiniIDE

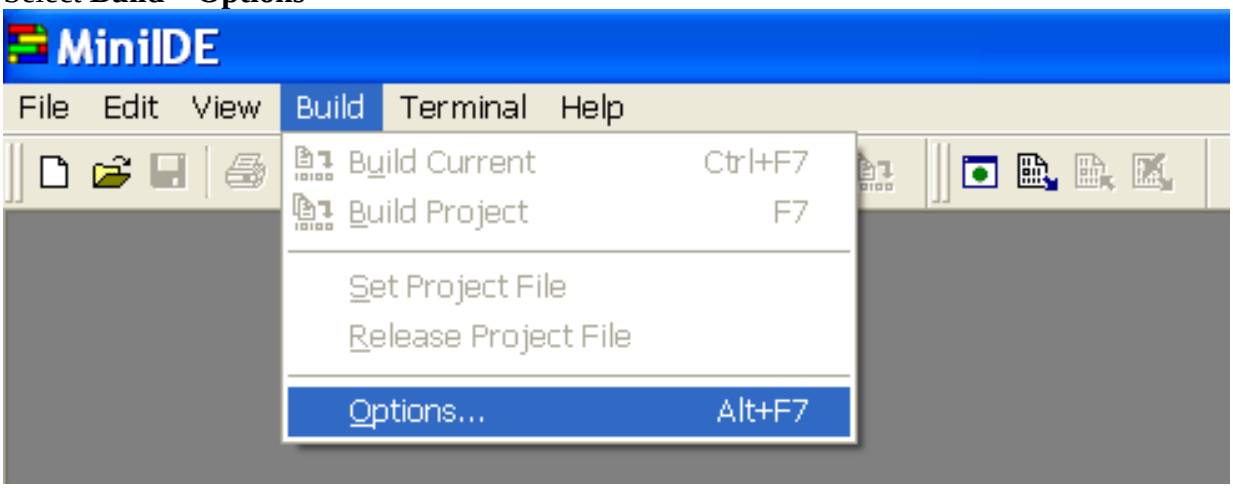
MiniIDE is a freeware assembler for the 68HC11 and 68HC12 that can be used to create an executable machine language program (.S19 file) and a listing (.LST file) from an assembly language program (.ASM file).

1) **Install MiniIDE**
   - MiniIDE can be downloaded from the instructor's web page or from the following URL: www.mgtek.com/**miniide/**
   - The installation program should install MiniIDE in a new folder named C:\ProgramFiles\MGTEK\MiniIDE\ as shown below:

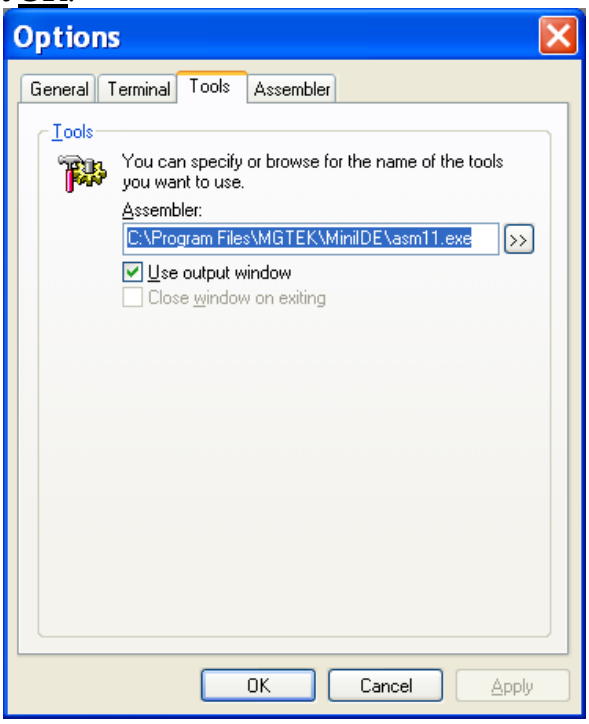


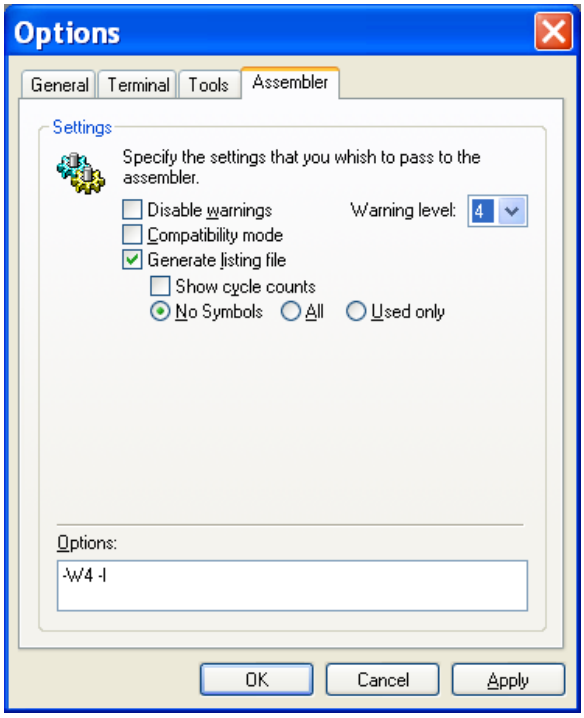
2) **Launch MiniIDE**
   - Launch MiniIDE using Start – **All Programs – MGTEK MiniIDE**

3) **Set Options** in MiniIDE
   - Select **Build – Options**

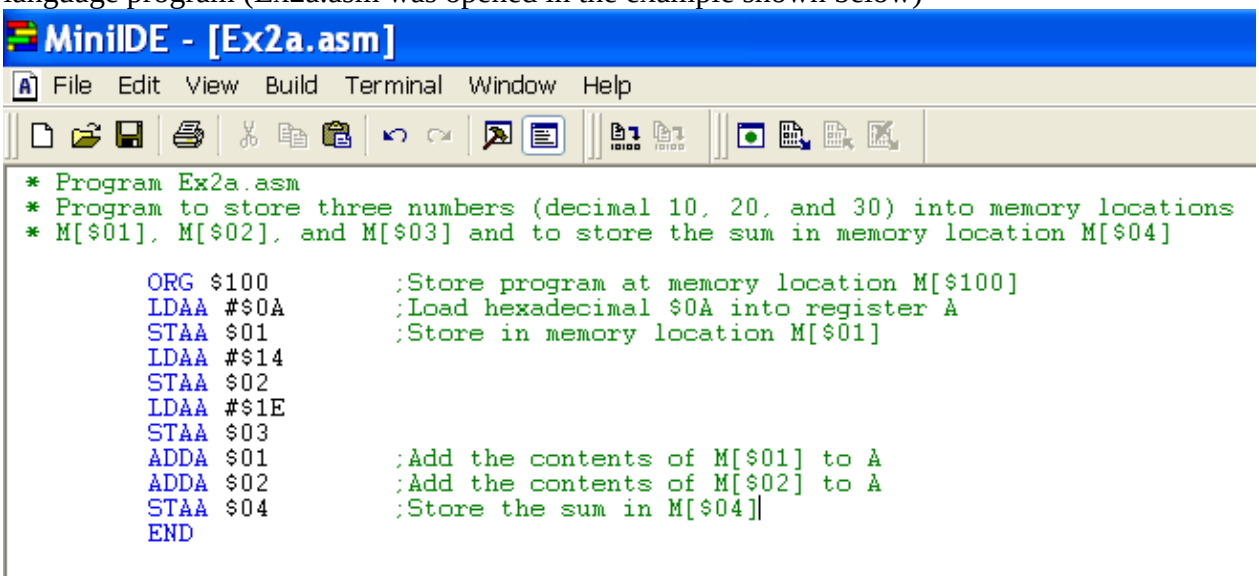


- Select the **Tools** tab in the **Options** window (see screen on left below)
  Select the assembler **asm11.exe** that is located in the MiniIDE folder and select OK.
- Select the **Assembler** tab in the **Options** window (see screen on right below)
  Select **Generate listing file** and **Warning level 4** (the highest level so all warnings are displayed).
- Select **OK**.



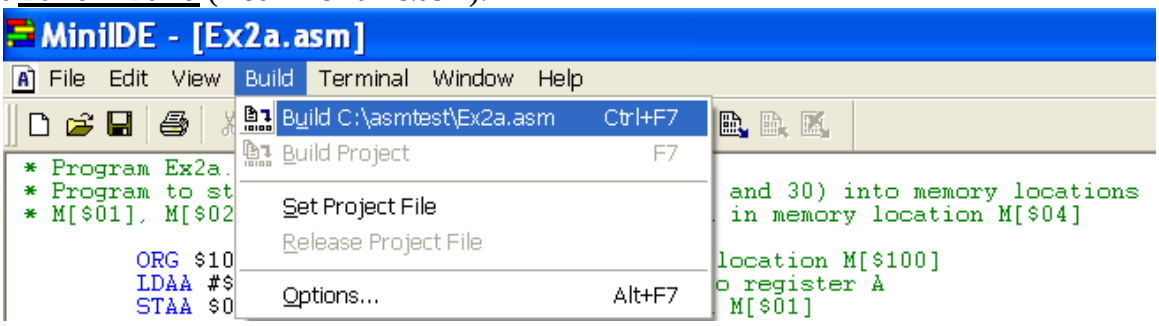


4) **Open or create an assembly language program**
   - To create a new assembly language program, select **File – New** and enter your program. Then select **File – Save** and save your program using an **asm** extension.
   - To open an existing assembly language program, select **File – Open** and select your assembly language program (Ex2a.asm was opened in the example shown below)

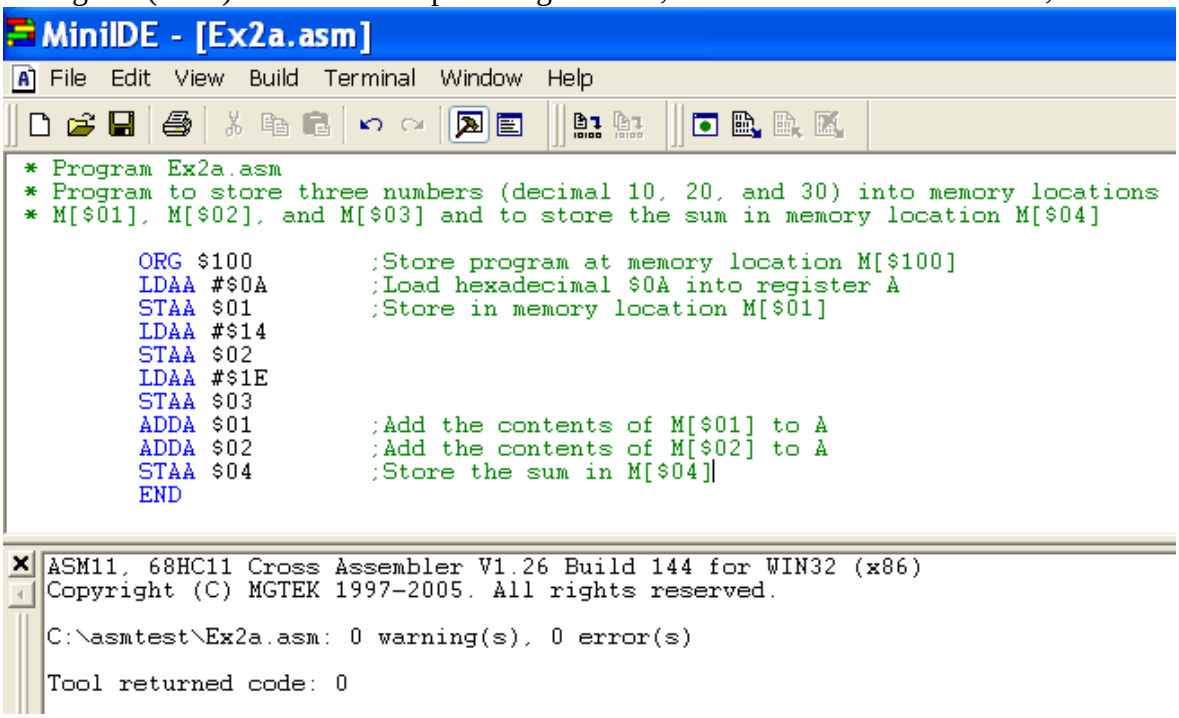


5) **Assemble the program**

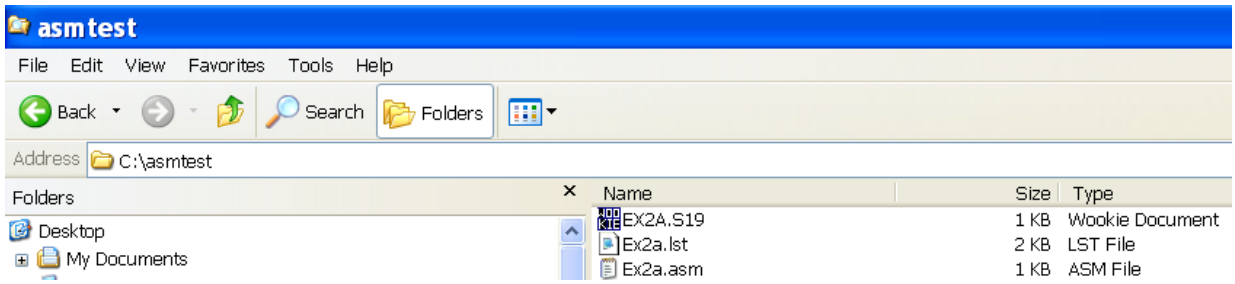
- Select **Build – Build** (YourFileName.asm).



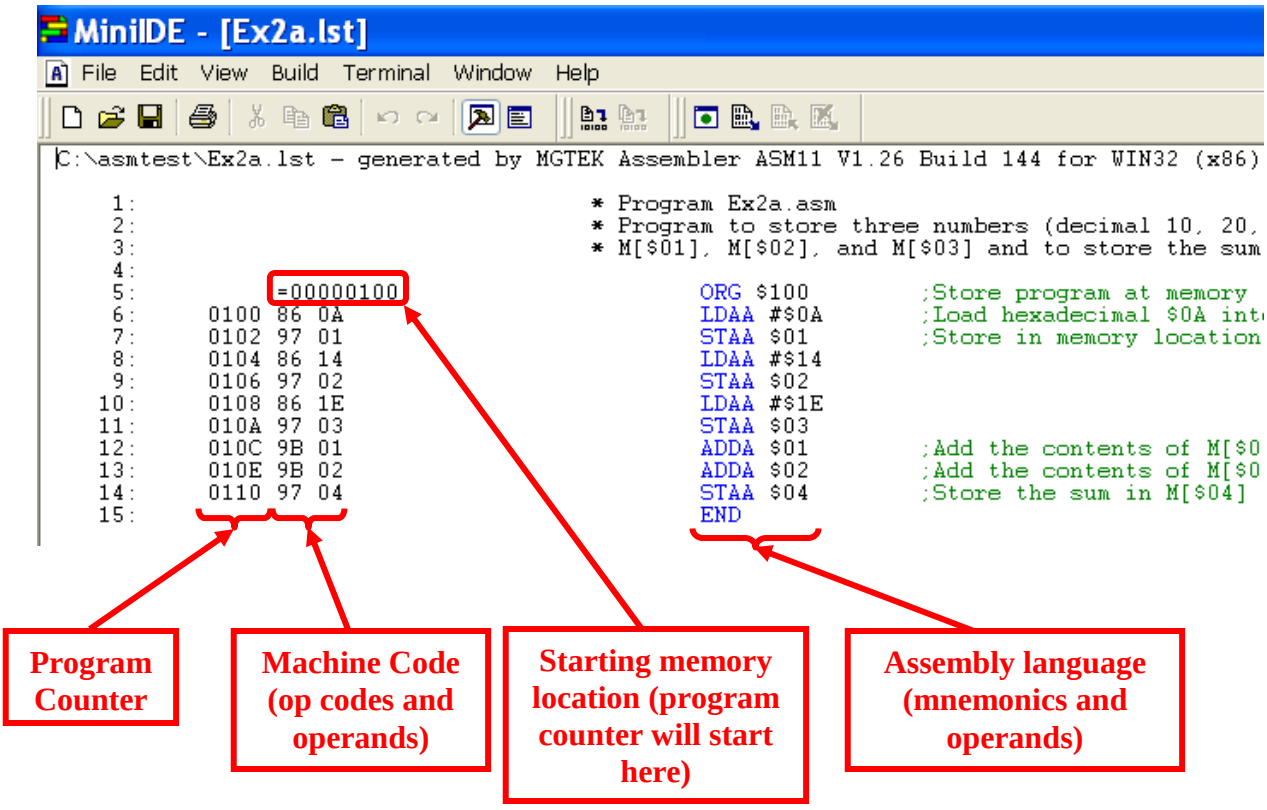
- Assembling the program results in the creation of:
  1. Error messages if the assembler detects any errors (note 0 errors in the example below)
  2. Object code or machine code (.S19 file)
  3. Listing file (.LST) which shows op codes generated, offsets for branch statements, etc.



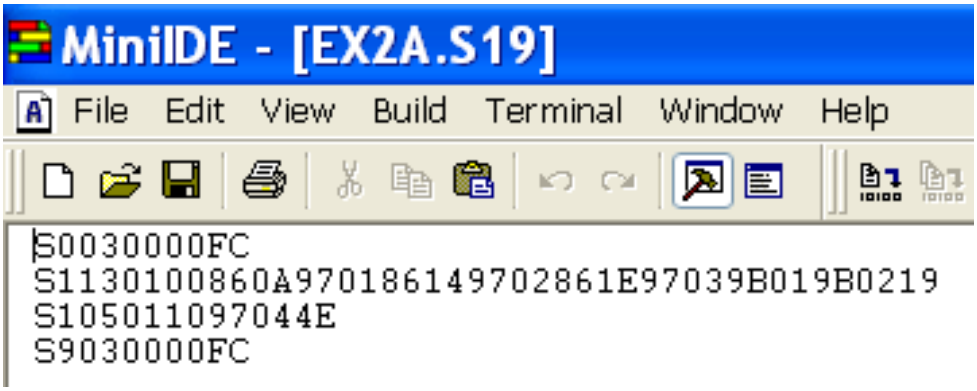
- Note that the .S19 and .LST files are created in the same folder where the .ASM file is stored as shown below.



- **Listing File**. You can open the listing file using MiniIDE or with any word processor, such as Notepad. The listing file includes:
    - original assembly language instructions (mnemonics and operands)
    - machine code (op codes and operands)
    - the starting memory location
    - program counter



- **S19 File** The S19 file isn't a file that we normally open, but let's look at it here just to see that it mainly contains the starting memory address and machine code. See if you can spot the starting address and the machine code by comparing it to the LST file above. The S19 file is downloaded into ROM in the 68HC11.

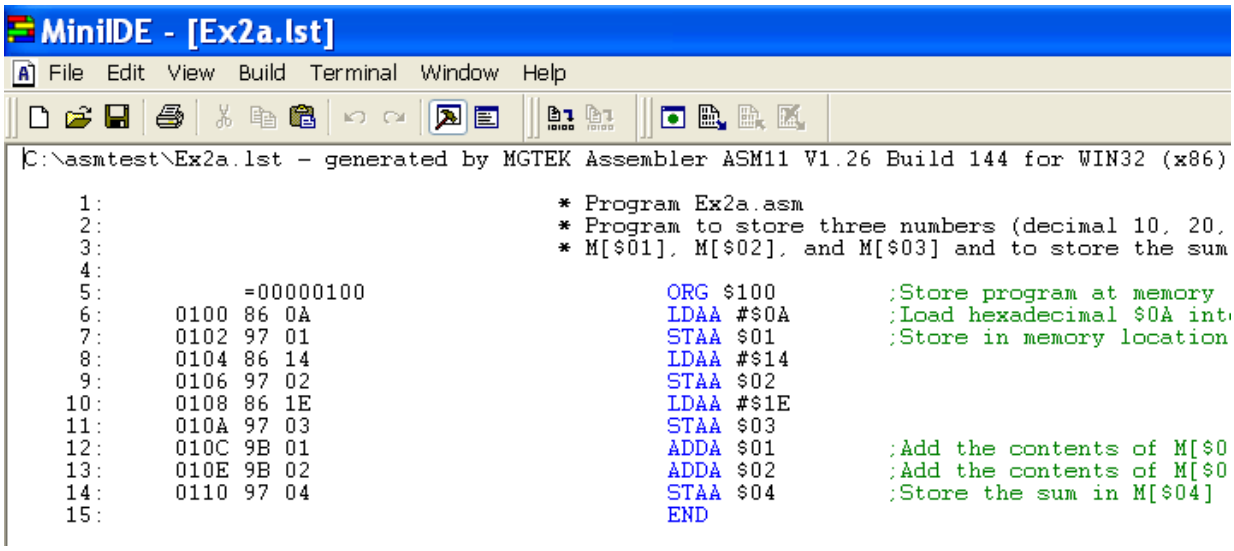


Wookie 68HC11 Simulator

The Wookie simulator is an excellent freeware program that can be used to simulate 68HC11 programs before actually downloading them into a microprocessor in lab. Wookie, short for "Wireless Object-Oriented Kindly Interfaced Emulator," is a Win32 emulator for 68HC11-based software development that was developed by a senior design team at the Milwaukee School of Engineering (MSOE). The software can be downloaded from the instructor's web page or from various websites, including: http://www.msoe.edu/eecs/ce/ceb/resources/

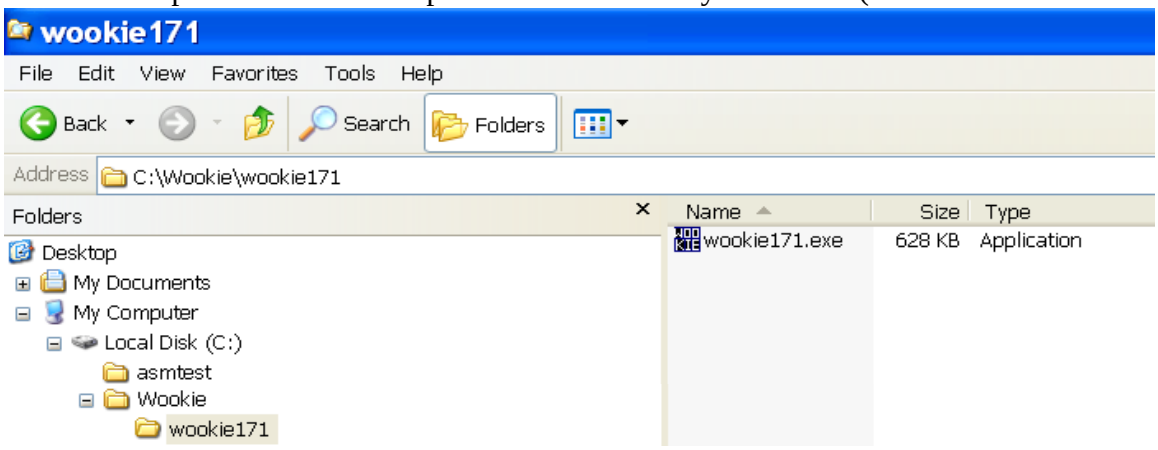
Before running Wookie, let's take a closer look at the program Ex2a.asm shown above and see what it does.

- It stores the program at memory location $100, so the program counter will start here.
- It loads $0A (decimal 10) into accumulator A and then stores it at memory location [$01].
- It loads $14 (decimal 20) into accumulator A and then stores it at memory location [$02].
- It loads $1E (decimal 30) into accumulator A and then stores it at memory location [$03].
- It adds the contents of M[$01] to accumulator A (so A will now contain decimal 30+10=40 or $28)
- It adds the contents of M[$02] to accumulator A (so A will now contain decimal 40+20=60 or $3C)
- It stores the value in accumulator A ($3C or decimal 60) at memory location [$04].



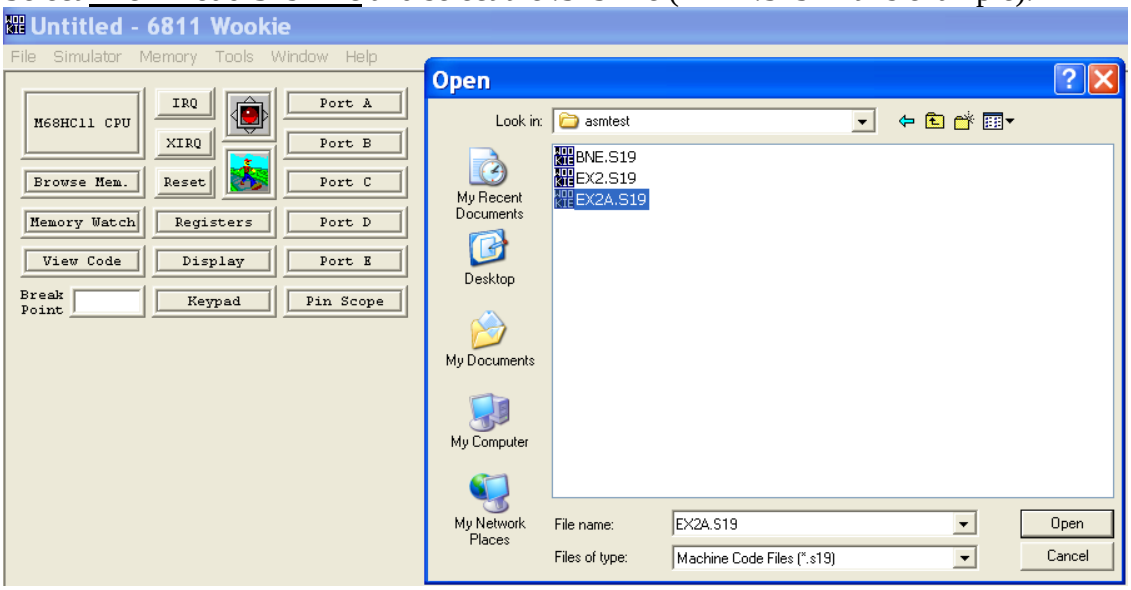
Now we will load the S19 file into Wookie and watch it change the accumulator contents and memory address contents as we step through the program.

1) **Install Wookie**
   - Wookie can be downloaded from the instructor's web page or from various websites, including: http://www.msoe.edu/eecs/ce/ceb/resources/
   - Extract the zip file Wookie171.zip into the location of your choice (C:\Wookie is used below).

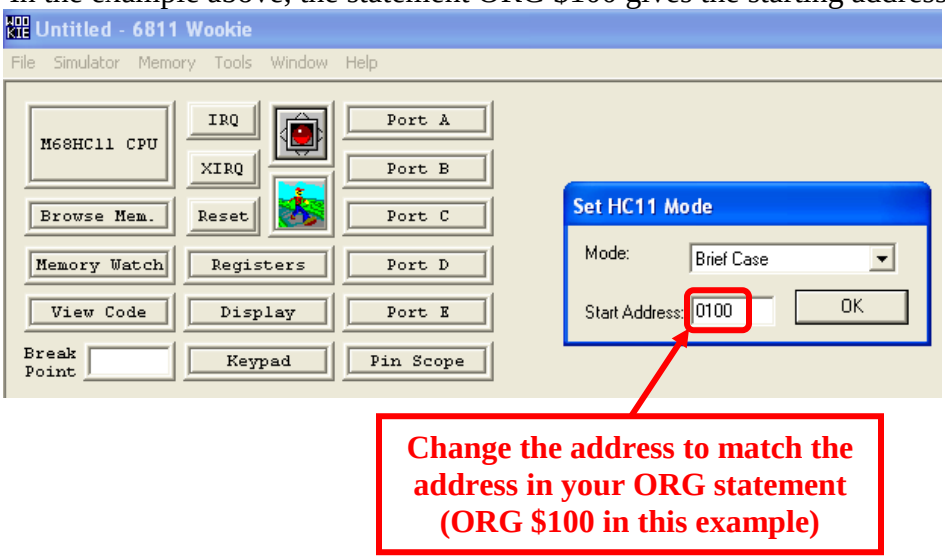


2) **Launch the Wookie simulator**
   - Locate Wookie171.exe using Windows Explorer or MyComputer and launch the program.

3) **Loading the S19 file**
   - Select **File – Load S19 File** and select the .S19 file (Ex2A.S19 in this example).

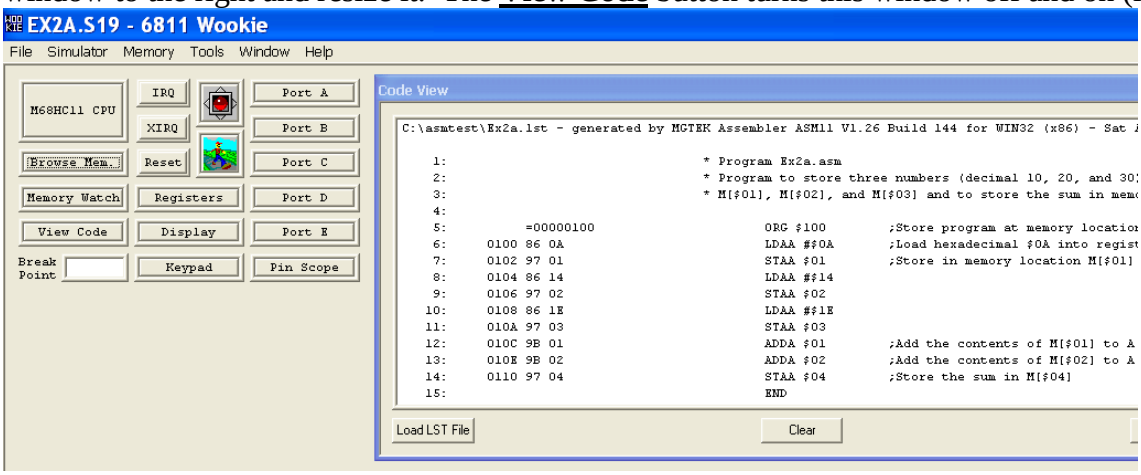


- After loading the S19 file above, the **Set HC11 Mode** window will automatically appear. Leave the mode as **Brief Case**, but change the starting address to the address in the ORG statement in your .asm file). In the example above, the statement ORG $100 gives the starting address.

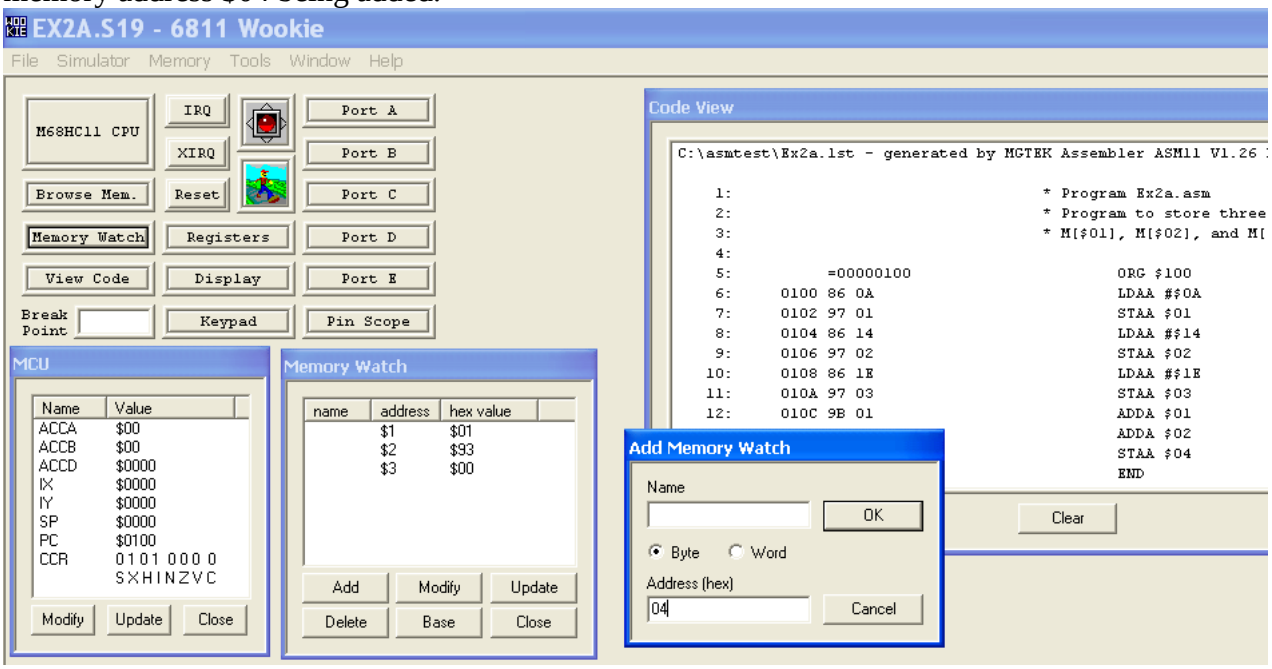


- The **Choose LST file format** window will now automatically appear. Change the **File Type** to **AS6811/ASM11 (addr code)** and leave the **Address Offset** at **0** (default).

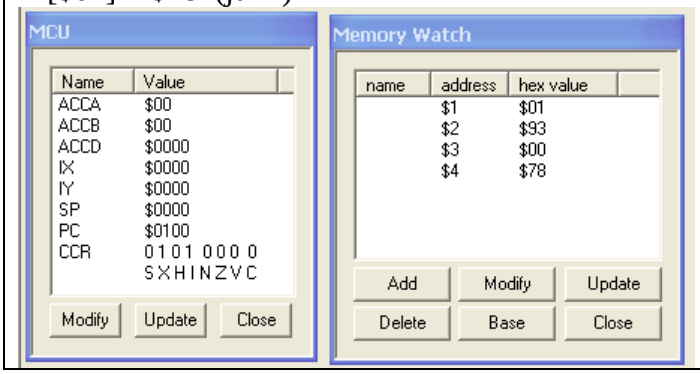
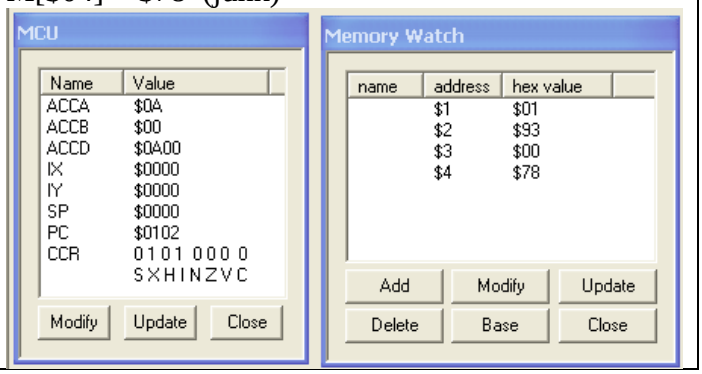
- Wookie should now display your code in the **Code View** window. You might need to drag the window to the right and resize it. The **View Code** button turns this window off and on (leave it on).



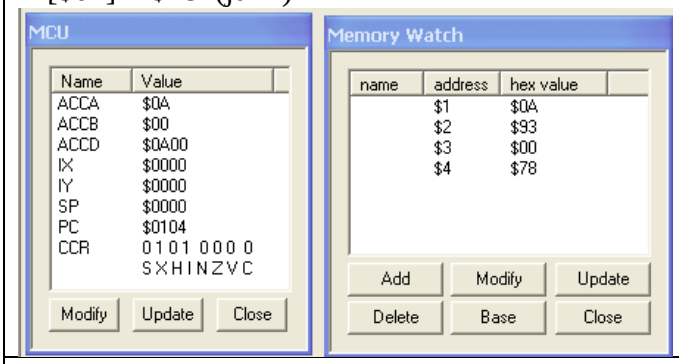
4) **Opening Useful Windows in Wookie**
   - After loading the .S19 file, the **Code View** window should be open.
   - Also click on the **M68HC11 CPU** button to open the MCU window which will show the contents of some key registers.
   - Also click on the **Memory Watch** button. Select **Add** and enter the memory addresses $01, $02, $03, and $04 since they are used in this example. The **Add Memory Watch** window below shows memory address $04 being added.



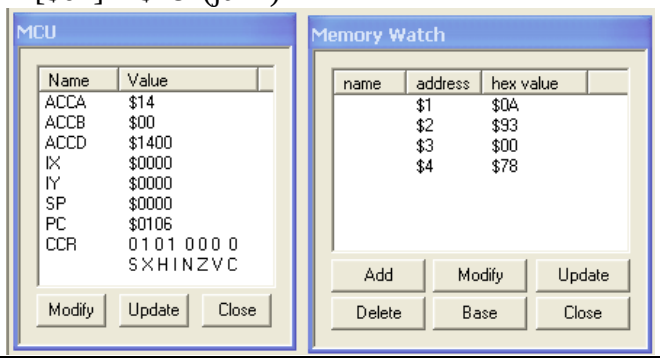
5) **Single Step Through the Simulation.** Each time you click on the icon of the person walking (or select **Simulator – Simulator Step** or press the space bar), the Program Counter (PC) should advance to the address of the next instruction. You can check the contents of memory addresses and registers as you advance.

Before starting:	**After 1 step**: (after executing `LDAA #$0A`)
PC = $0100 (starting address) ACCA = $00 M[$01] = $01 (junk) M[$02] = $93 (junk) M[$03] = $00 (junk) M[$04] = $78 (junk)	PC = $0102 ACCA = $0A M[$01] = $01 (junk) M[$02] = $93 (junk) M[$03] = $00 (junk) M[$04] = $78 (junk)
	

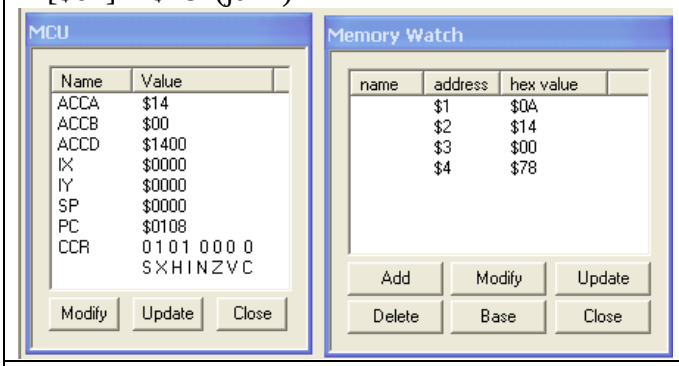
After 2 steps: (after executing `STAA $01`)
PC = $0104
ACCA = $0A
M[$01] = $0A
M[$02] = $93 (junk)
M[$03] = $00 (junk)
M[$04] = $78 (junk)



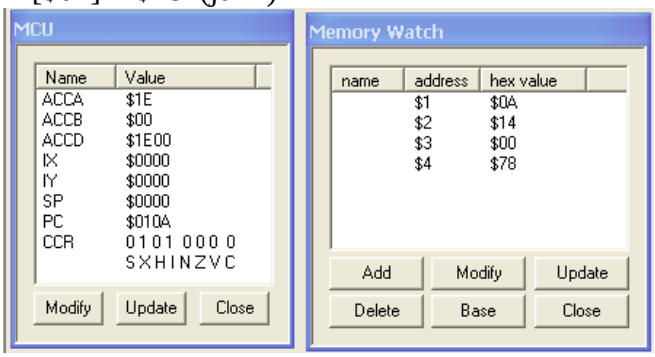
After 3 steps: (after executing `LDAA #$14`)
PC = $0106
ACCA = $14
M[$01] = $0A
M[$02] = $93 (junk)
M[$03] = $00 (junk)
M[$04] = $78 (junk)



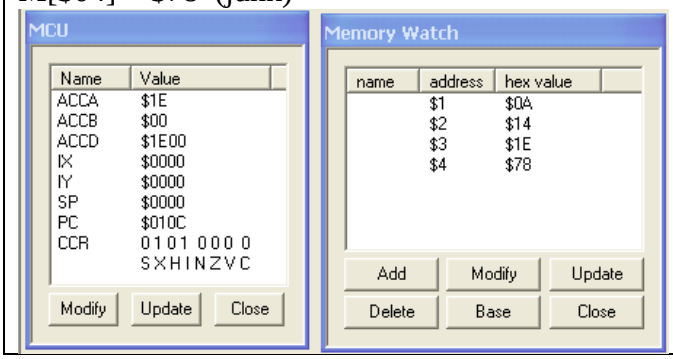
After 4 steps: (after executing `STAA $02`)
PC = $0108
ACCA = $14
M[$01] = $0A
M[$02] = $14
M[$03] = $00 (junk)
M[$04] = $78 (junk)



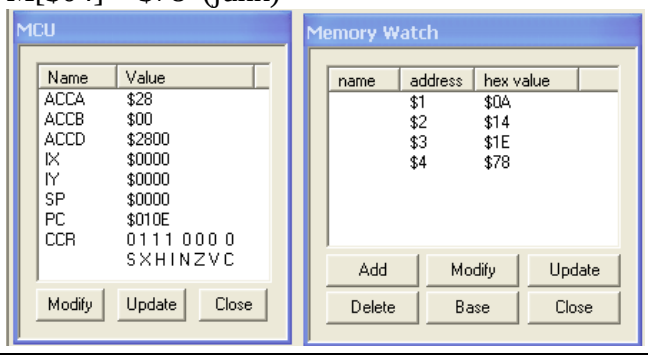
After 5 steps: (after executing `LDAA #$1E`)
PC = $010A
ACCA = $1E
M[$01] = $0A
M[$02] = $14
M[$03] = $00 (junk)
M[$04] = $78 (junk)



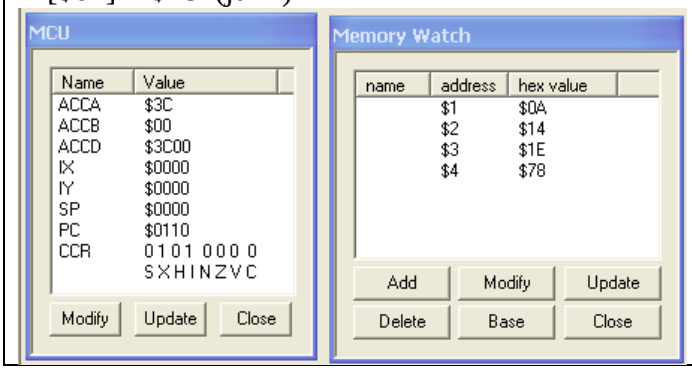
After 6 steps: (after executing `STAA $03`)
PC = $010C
ACCA = $1E
M[$01] = $0A
M[$02] = $14
M[$03] = $1E
M[$04] = $78 (junk)



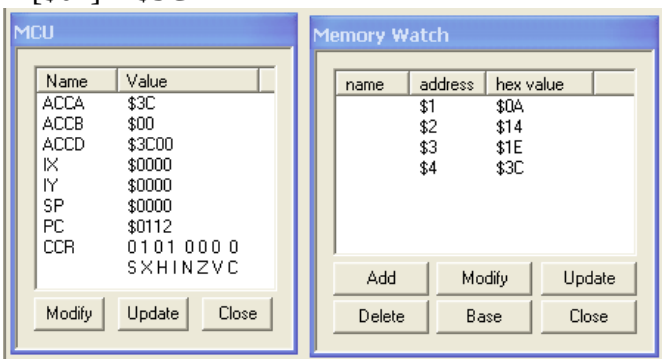
After 7 steps: (after executing `ADDA $01`)
PC = $010E
ACCA = $28
M[$01] = $0A
M[$02] = $14
M[$03] = $1E
M[$04] = $78 (junk)



After 8 steps: (after executing `ADDA $02`)
PC = $0110
ACCA = $3C
M[$01] = $0A
M[$02] = $14
M[$03] = $1E
M[$04] = $78 (junk)

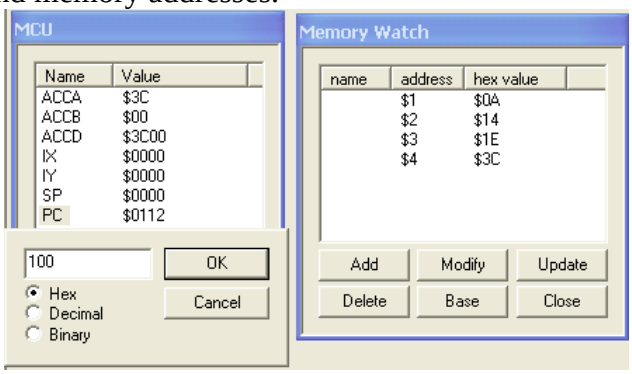


After 9 steps: (after executing `STAA $04`)
PC = $0102
ACCA = $3C
M[$01] = $0A
M[$02] = $14
M[$03] = $1E
M[$04] = $3C



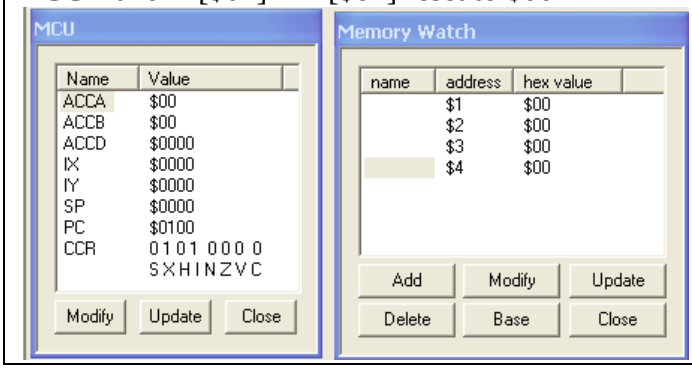
- Note that the simulation was correct.

6) **Run the Entire Simulation** You can also run the entire simulation with one command as described below.
   - **Reset the PC**. Before rerunning a program, set the PC back to the starting address ($0100 in this case) using the **Modify** button in the **Memory Watch** window (see below). Note that you can also reset other registers and memory addresses.

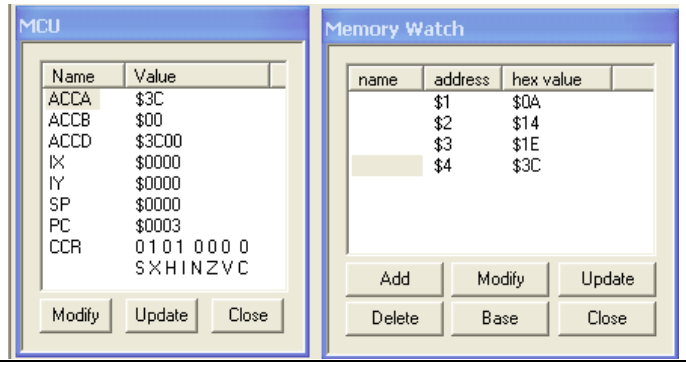


- **Run the simulation**. Run the full simulation by pressing the button that looks like a traffic light twice (turns green then red) (or select Simulation – Simulations Start/Stop)

PC reset to $0100
ACCA and M[$01] – M[$04] reset to $00



After running the entire simulation:



Simulating Programs that will be used on the MicroStamp11

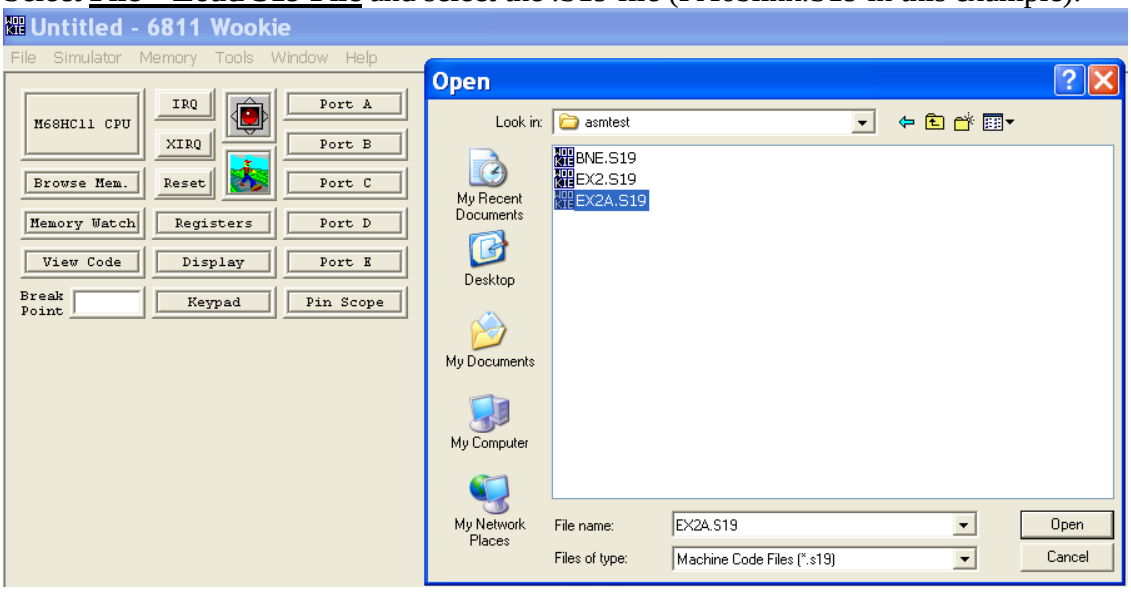
The example above was generic and was not geared for a specific version of the 68HC11. However, in lab we use the MicroStamp11, a microcontroller board based on the MC68HC11D0. In general the 68HC11 can be configured to operate in three modes: single-chip mode, bootstrap mode, and expanded chip mode. The MicroStamp11 operates in **expanded-chip mode**.

Example 2: Simulate the program PA6blink.asm

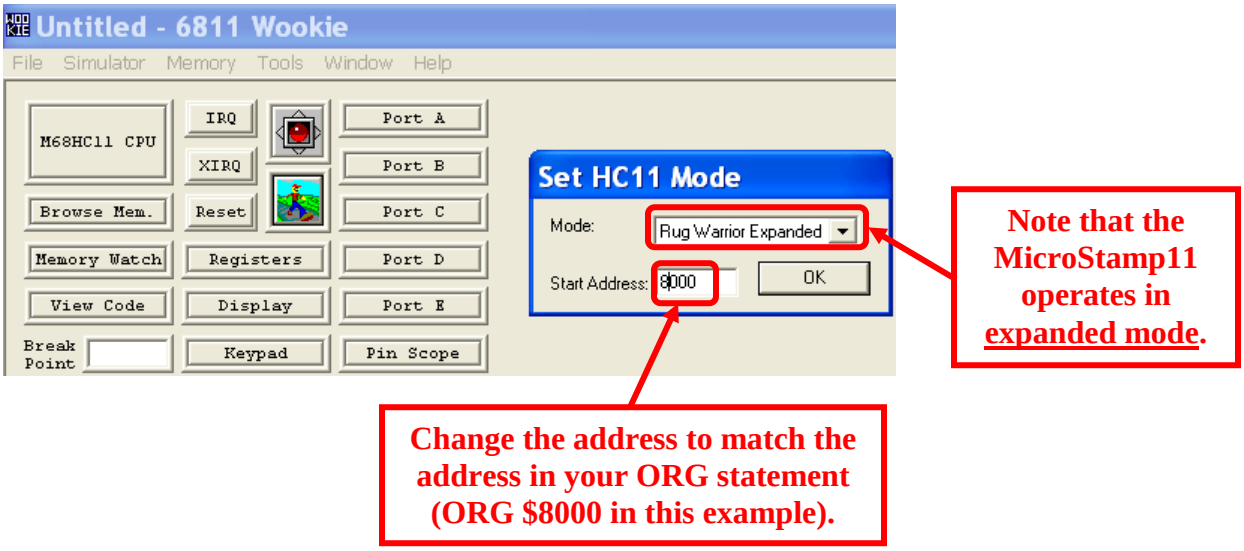
- This program is used in Lab 8.
- The program is designed to make an LED connected to output PA6 blink ON for ½ second and then OFF for ½ second indefinitely.
- Let's use Wookie to simulate it before downloading it into the MicroStamp11 to insure that the program is correct.
- Before beginning the simulation, use Mini IDE to assemble PA6blink.asm in order to create the machine code, PA6blink.s19.

1) **Launch the Wookie simulator**
   - Locate Wookie171.exe using Windows Explorer or MyComputer and launch the program.

2) **Loading the S19 file**
   - Select **File – Load S19 File** and select the .S19 file (PA6blink.S19 in this example).

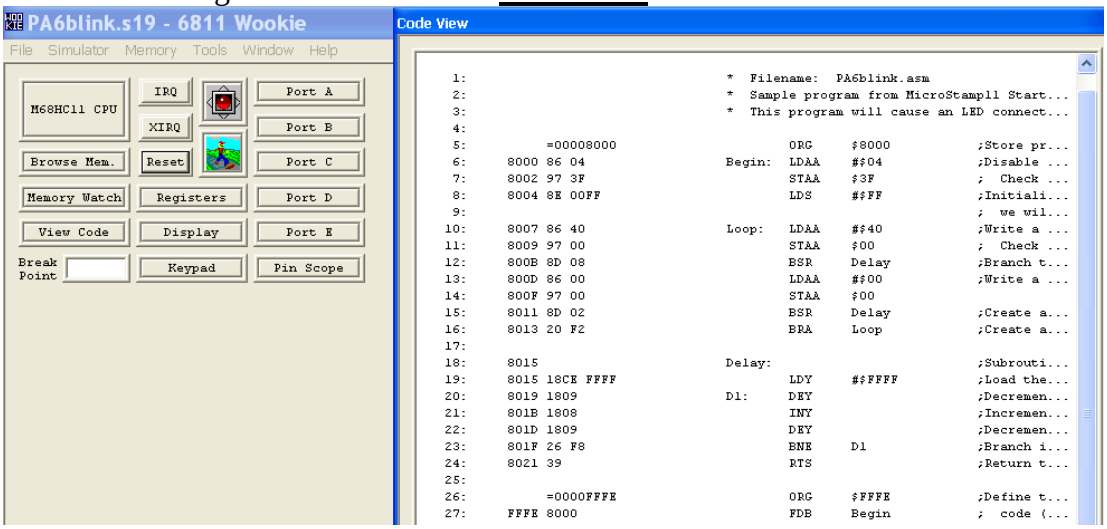


- After loading the S19 file above, the **Set HC11 Mode** window will automatically appear. Recall that the MicroStamp11 operates in expanded-chip mode, so change the mode to **Rug Warrior Expanded** and change the starting address to the address in the ORG statement in the asm file). In PA6blink.asm, the statement ORG $8000 gives the starting address.



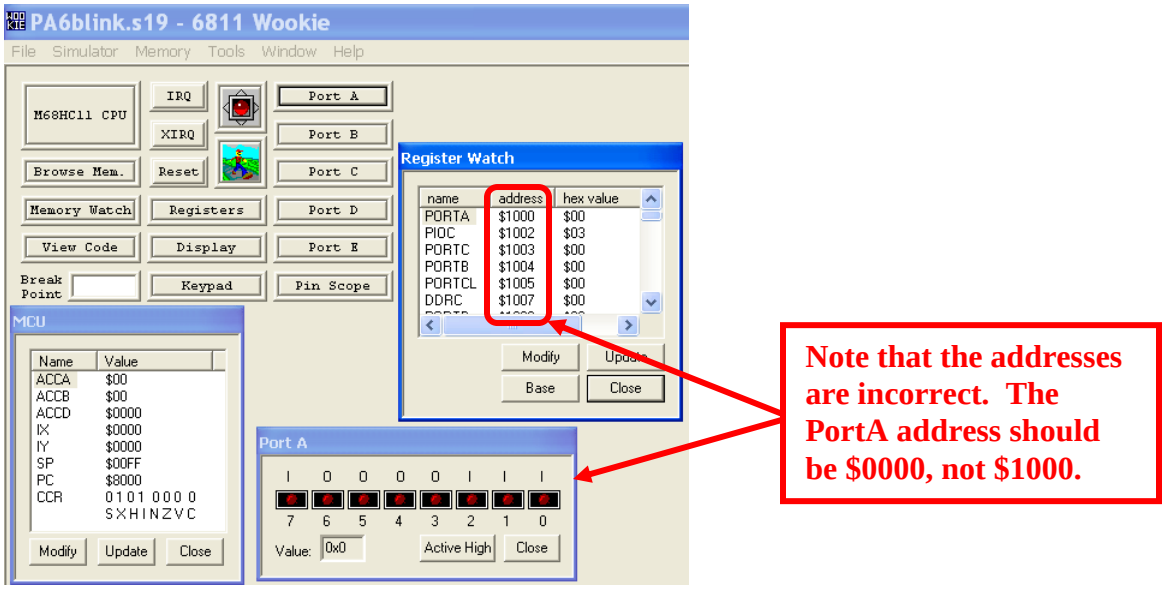
- The **Choose LST file format** window will now automatically appear. Change the **File Type** to **MGTEK ASM11 [line: addr code]** and leave the **Address Offset** at **0** (default).

- Wookie should now display your code in the **Code View** window. You might need to drag the window to the right and resize it. The **View Code** button turns this window off and on (leave it on).

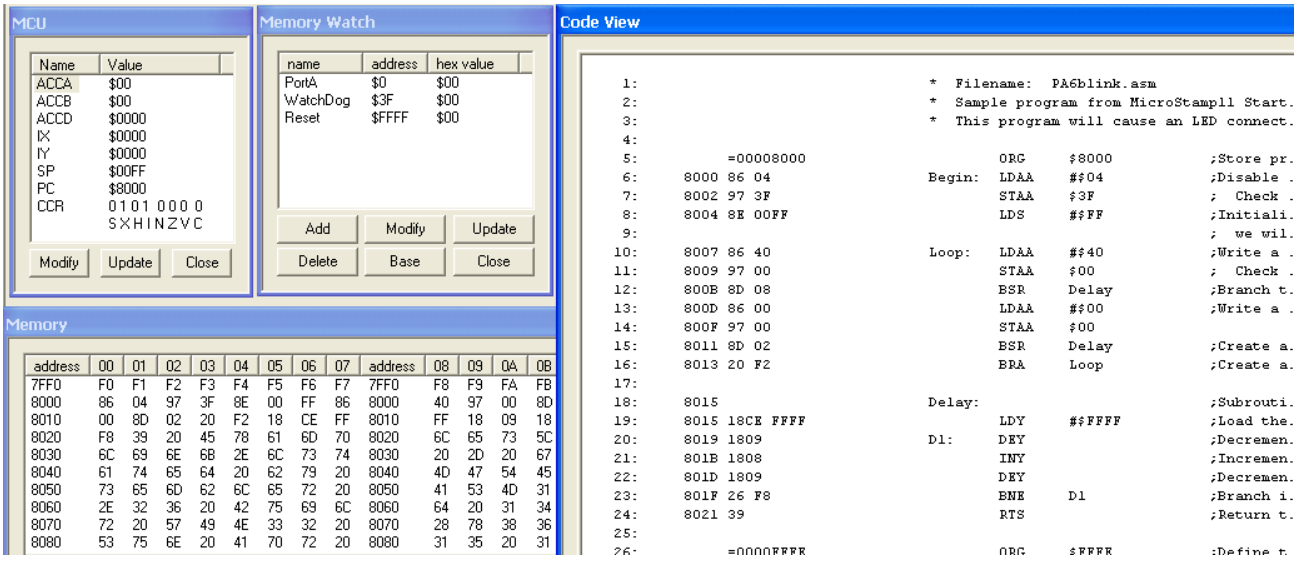


3) **Opening Useful Windows in Wookie**

- After loading the .S19 file, the **Code View** window should be open (if not, click **View Code**).
- Also click on the **M68HC11 CPU** button to open the MCU window which will show the contents of some key registers.
- The instructor noted a problem in selecting the Registers window and the PortA window: The addresses aren't correct for the MicroStamp11. Table 4-1 (see class notes) indicates that the address for PORTA is $0000, not $1000. Similarly, the address for PORTD is $0008, not $1008. In fact, all of the table 4-1 addresses have the form $00xx, not $10xx as shown in Wookie. It doesn't seem that there is any way to change them in Wookie either. If anyone can solve this problem, let the instructor know! It is disappointing that we can't use the PortA window as it has a clever let of 8 LED's that show the values on PortA. The 1's and 0's above the LED's indicate whether each pin is configured as an output (1) or an input (0).



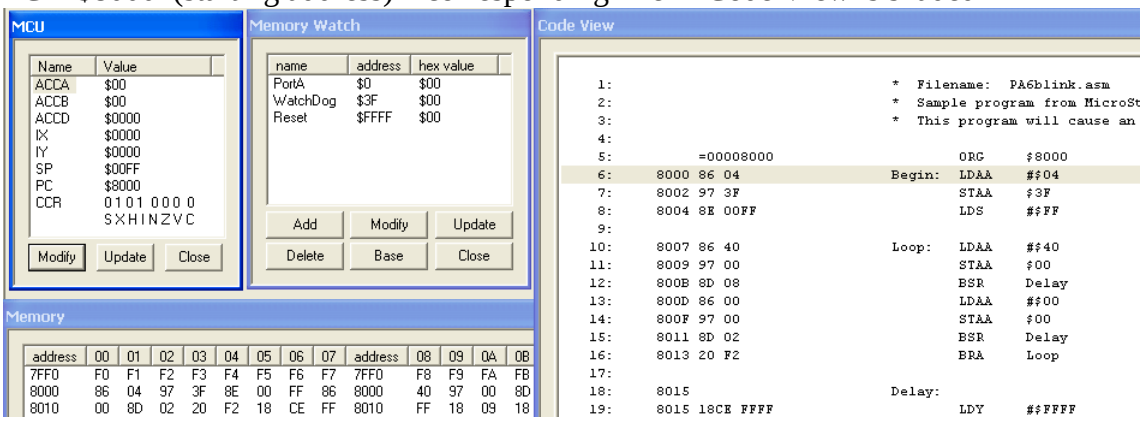
- Even though the addresses in the Register Watch window are incorrect, we can still use the **Memory Watch** window to check any addresses we wish (and name them). Select **Add** and enter the name PortA and the address $00. Similarly, we can check the address for disabling the watchdog timer ($FE) and the address for the Reset interrupt ($FFFE). Also open the **Browse Memory** window. It is interesting to scroll to the memory address $8000 and see the machine code for the program that are stored here.



4) **Single Step Through the Simulation.** Each time you click on the icon of the person walking (or select **Simulator – Simulator Step** or press the space bar), the Program Counter (PC) should advance to the address of the next instruction. Note that the Program Counter should begin at $8000.

A) Before starting:

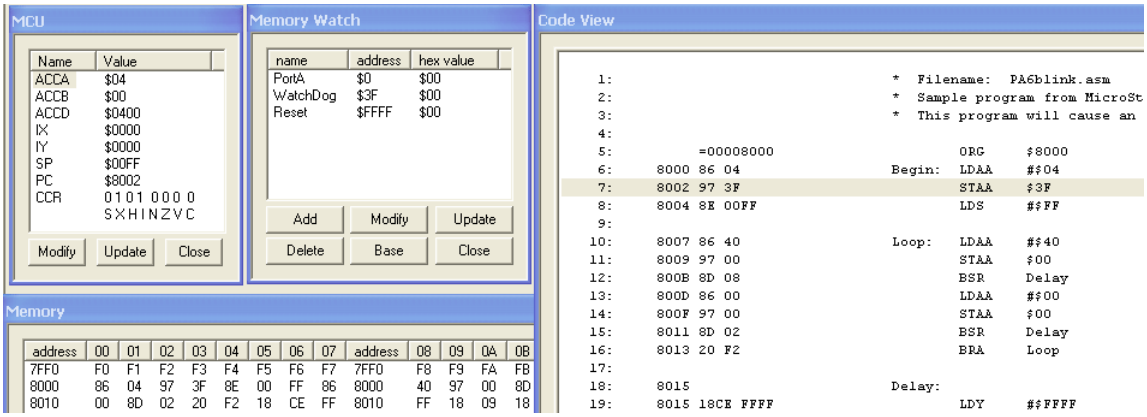
PC = $8000 (starting address) – corresponding line in Code View is shaded



B) After 1 step:

PC = $8002

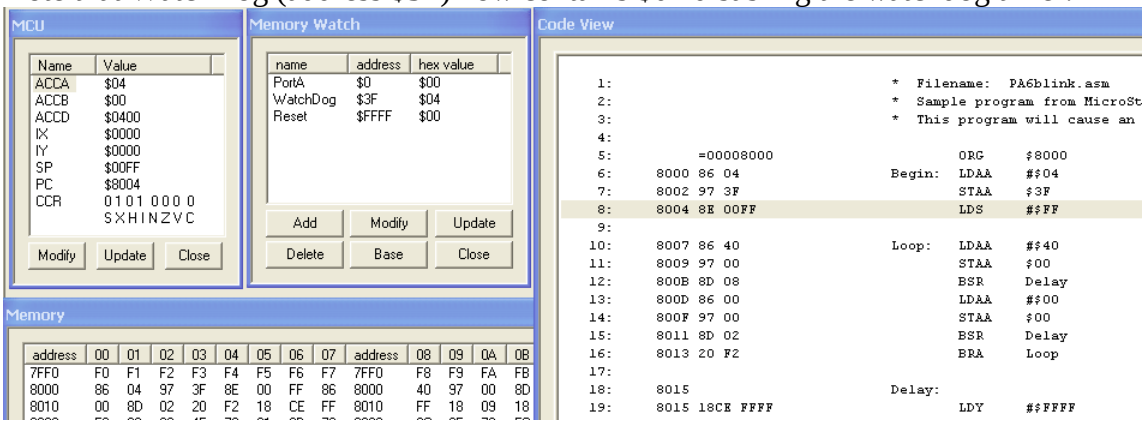
Note that ACCA now contains $04



C) After 2 steps:

PC = $8004

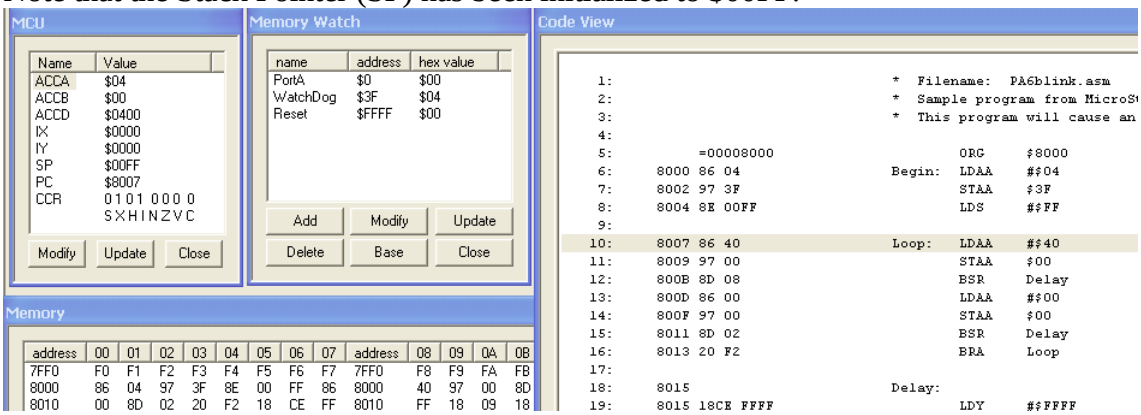
Note that WatchDog (address $3F) now contains $04 disabling the watchdog timer.



D) **After 3 steps**:

PC = $8004

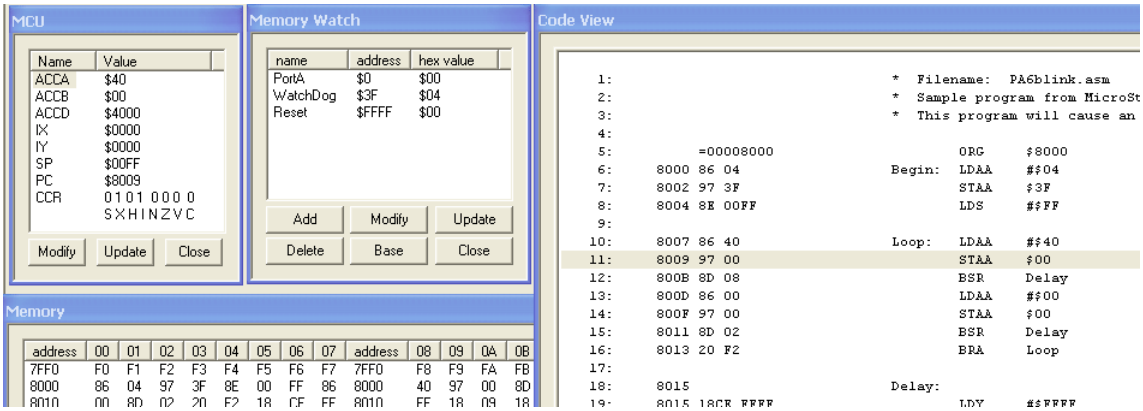
Note that the Stack Pointer (SP) has been initialized to $00FF.



E) After 4 steps:

PC = $8006

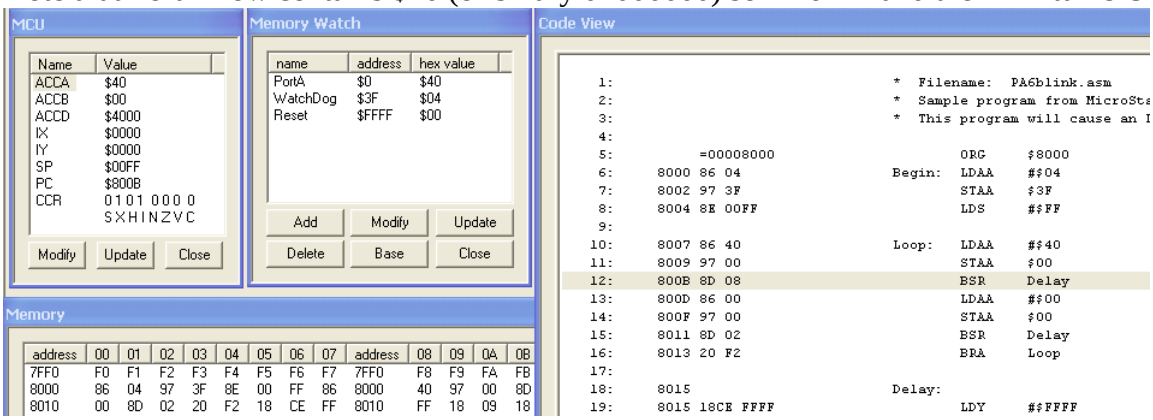
Note that ACCA now contains $40.



F) After 5 steps:

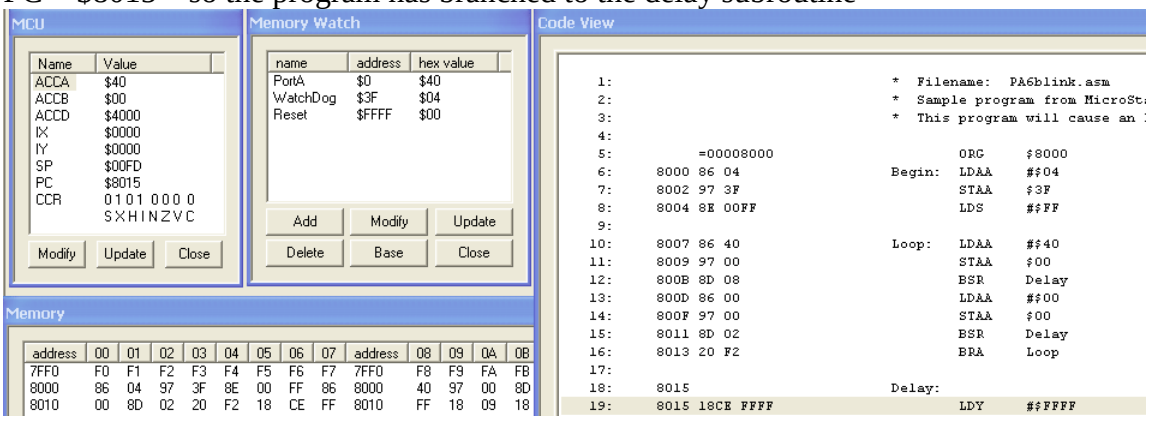
PC = $8008

Note that PortA now contains $40 (or binary 01000000) so PA6 = 1 and the LED turns ON.



G) After 6 steps:

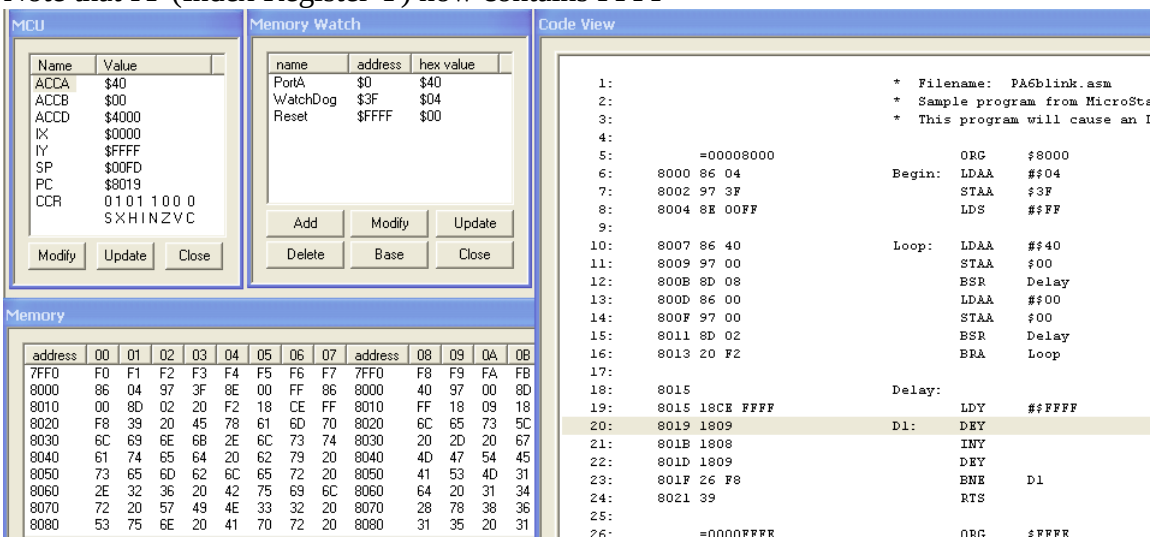
PC = $8015 – so the program has branched to the delay subroutine



H) After 7 steps:

PC = $8019

Note that IY (Index Register Y) now contains FFFF

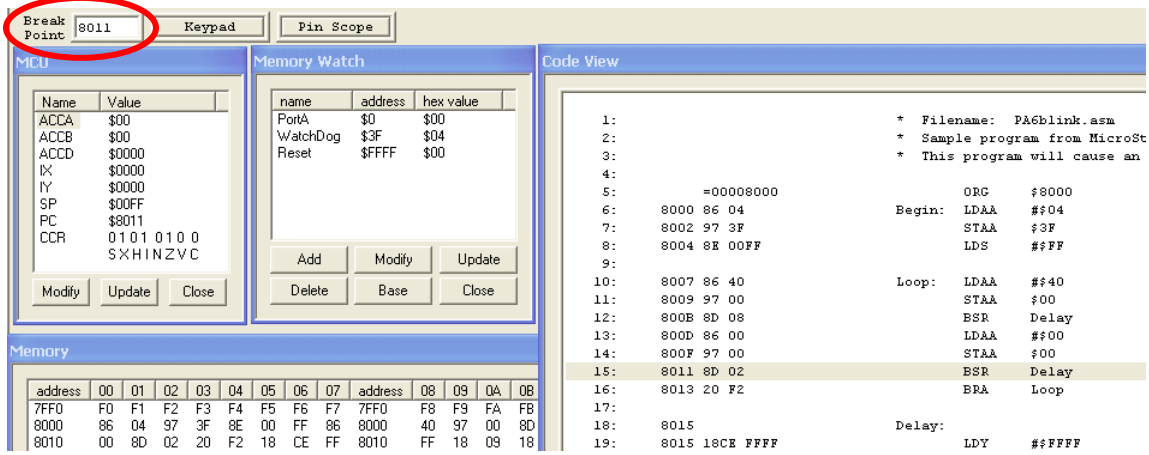


We have a problem! IY is decremented each time through the delay loop until it equals 0. Its initial value is $FFFF so we have to go through the loop 65,535 times! That is a lot of single stepping! We need a better way – by adding **BreakPoint**. If we specify a breakpoint (address) and click on the stoplight icon, the simulation will run up to that point.

I) <u>Add a BreakPoint of $8011 and click the traffic light</u>:

$8011 is the point where the delay subroutine is called the second time, so we have loaded a new value into PortA.

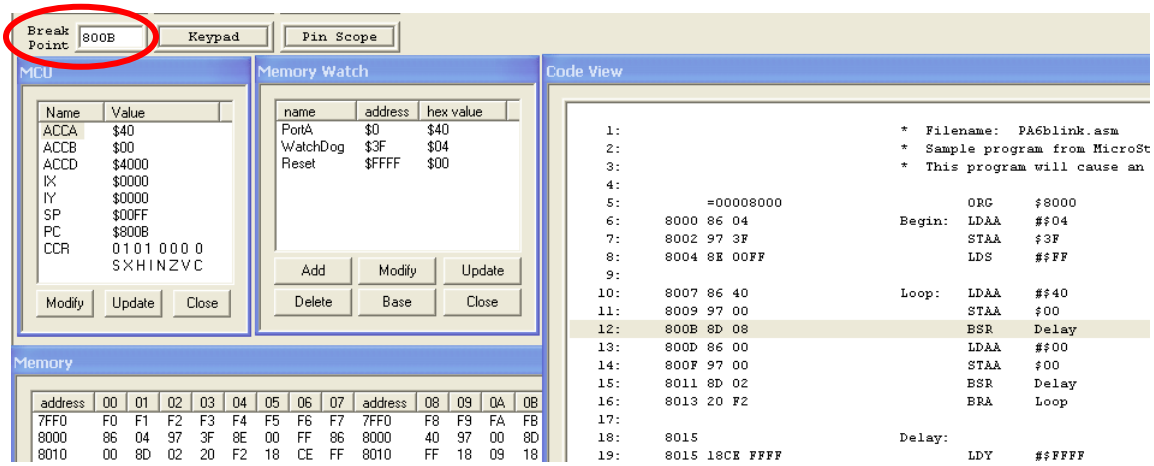
PortA now contains $00 (=00000000 in binary) so PA6 = 0 and the LED is OFF.



J) <u>Add a BreakPoint of $800B and click the traffic light</u>:

$800B is the point where the delay subroutine is called the first time, so we have loaded a new value into PortA.

PortA now contains $40 (=01000000 in binary) so PA6 = 1 and the LED is ON.



K) <u>Repeat steps I and J indefinitely to simulate the LED turning OFF and ON</u>

Note that this program contains an infinite loop, so the LED will blink ON and OFF indefinitely. In other words, the simulation never ends! However, we have found out enough to be sure that it will work correctly when we download it into the MicroStamp11.

L) Reset Vector

Recall that the last two lines of the program (ORG $FFFF and FDB Begin) are used to specify the address of where the program is to be redirected if the reset button is pressed on the MicroStamp11 board. Note that these lines are never reached by single-stepping or by starting/stopping the simulation. The commands are assembler directives and they are executed before the program ever runs. Also note that the Memory Watch named Reset was poorly defined earlier as address $FFFF which is only the lower byte of the reset vector. It was replaced here using ResetUB (upper byte) and ResetLB (lower byte) and we can see that it contains the address $8000 which corresponds to the label Begin at the start of the program.