# Practice Report

Name : XXXXX

Student Number: XXXXXX

Class: SDUST/UTAS-XXXX

## Content



# 1.Introduction

Target :

1. I use the Atmel Studio 7 simulator to run a C programme
2. to inspect the compiled assembly language produced.

Requirement:

1. explain how each line of C code has been complied to assembly code (Task A)
2. explain what the assembly code is doing (Task B)
3. explain how it is the same functionality as the C code. (Task C)
4. Need comments on the use of registers (Task D)
5. Need details of the corresponding hardware including memory and I/O (Task E)

Description:

I first analyzed the C code, followed by the compiled C code. Then to get a clearer overview of assembly language, I wrote assembly language with the same functionality based on the C code. Throughout the process I Debugged to see the exact details.

# 2.C code

## 2.1 Build C code

According to the ‘Debug Instructions ‘ in documentation, I successfully compiled the C code. As shown in Figure 1 .I know it through the tips in the bottom right corner “`Build succeeded.`”

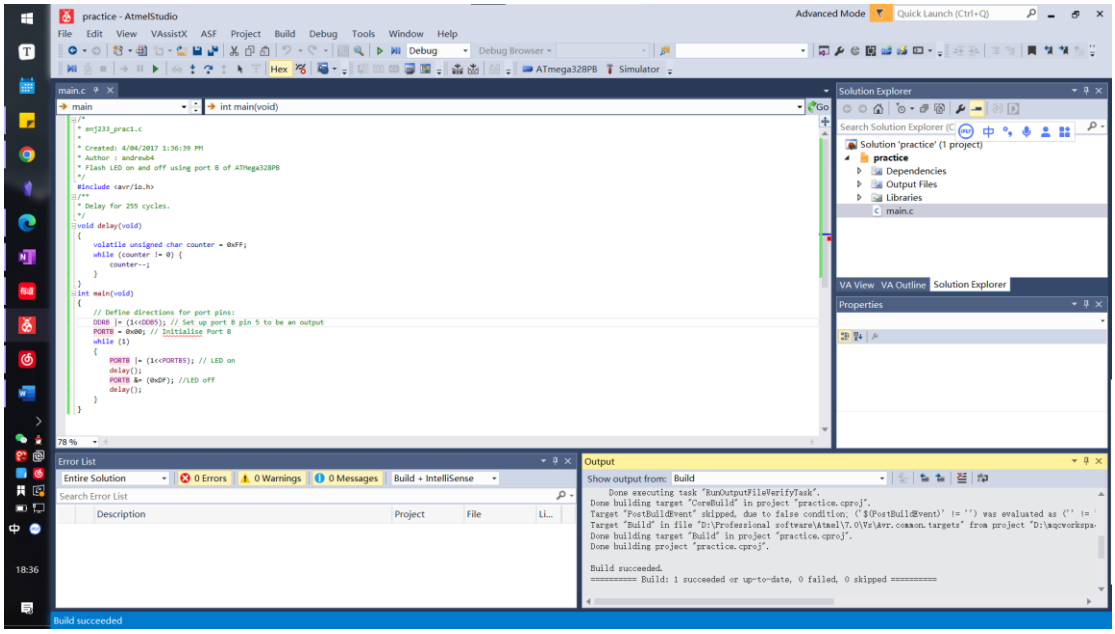



Figure 1

## 2.2 Explain C code

For simplicity, I add comments directly to the code. See Figure 2.and Figure 3 .

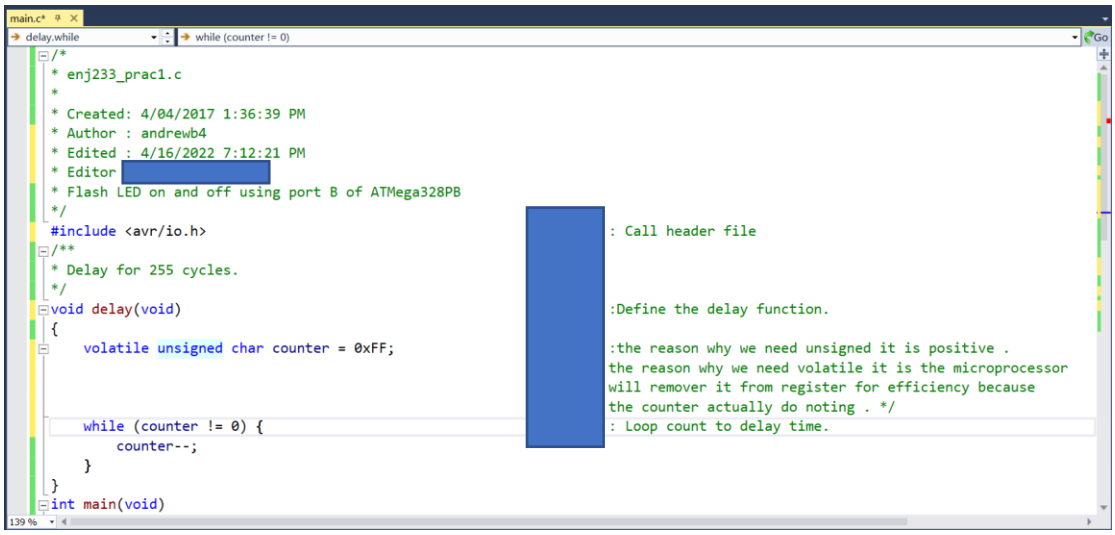



Figure 2

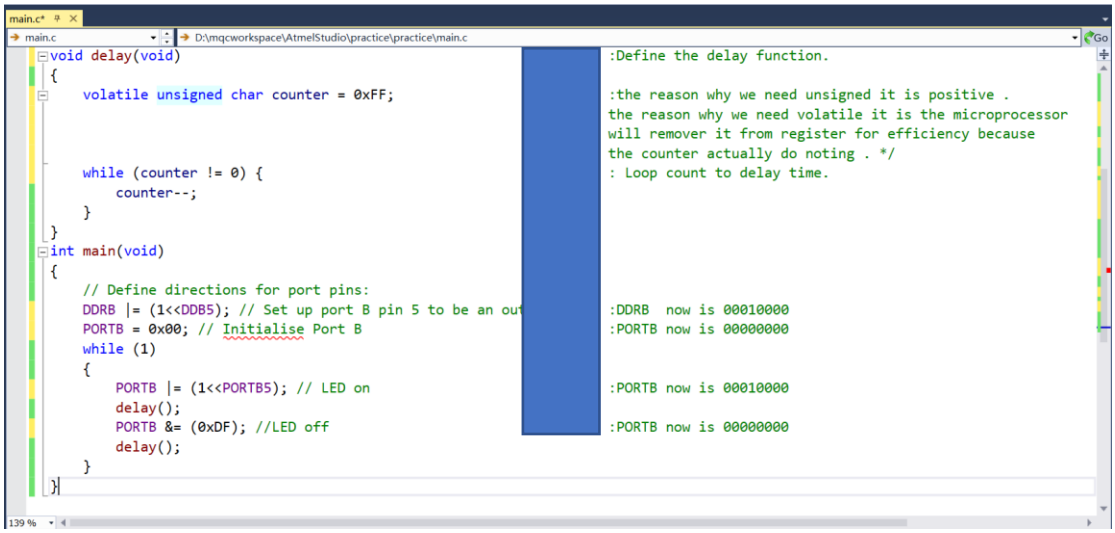



Figure 3

By the way, MQC is an abbreviation of my name.

The reason why we need **unsigned** it is positive . the reason why we need **volatile** it is the microprocessor will remover it from register for efficiency because the counter actually do noting .

## 2.3 The use of registers on Disassembly language (Task D)

The program used **register 24** to store the counter . See Figure 4.

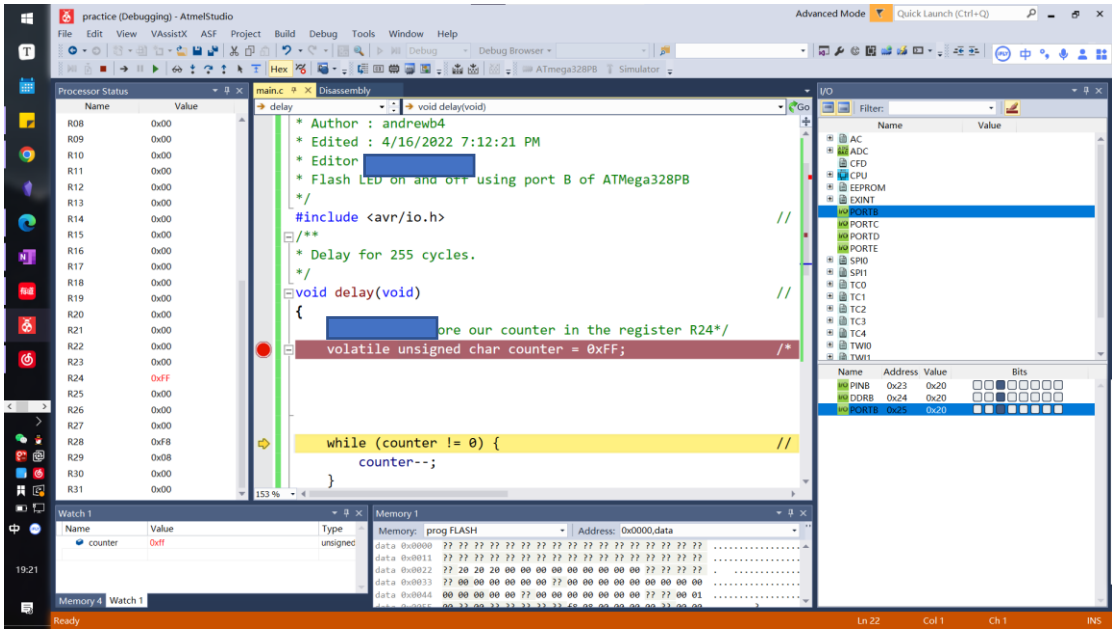



Figure 4

Obviously, we also make use of the **Data Direction Register** to specify PortB as an output. See Figure 5

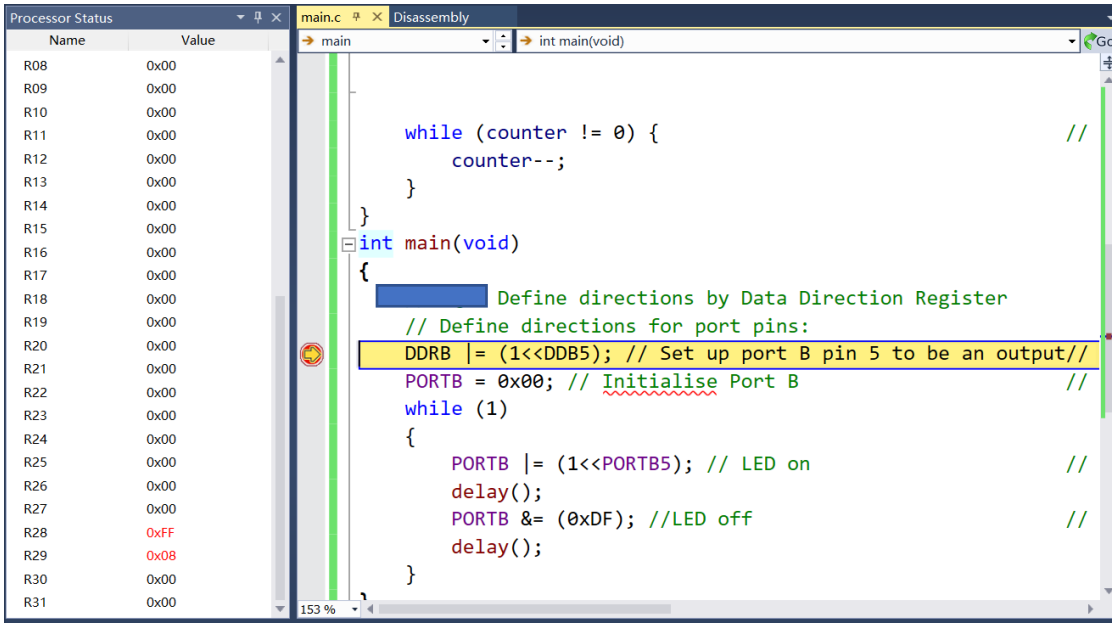



Figure 5

We can see exactly what happened in register 24 in our Disassembly language .
We used SER ,STD ,LDD ,TST ,SUBI ,CPSE instruction to operative R24 .See Figure6. (For more detail see Appendix)

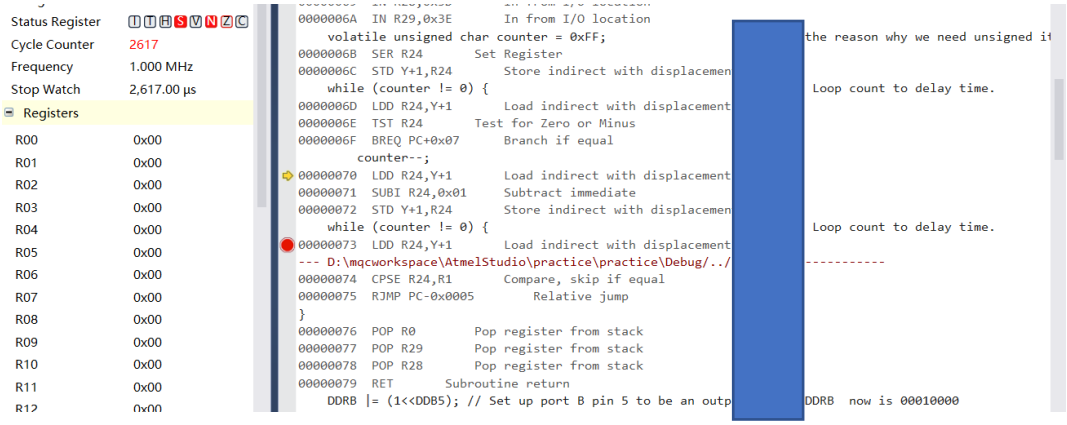



Figure 6

SER (Set all Bits in Register) is loads $FF directly to register Rd. It can be used as Initial Setup.
STD (Store Indirect From Register to Data Space using Index Y) It can store one byte indirect with or without displacement from a register to data space.
LDD （Load Indirect from Data Space to Register using Index Y） It can load one byte indirect with or without displacement from the data space to a register.
TST(Test for Zero or Minus) Tests if a register is zero or negative. Performs a logical AND between a register and itself.

**SUBI**(Subtract Immediate).We learn it before. And **this is what we exactly interested** . By using SUBI introduction , the value of R24 will minus one . It is the `counter--` in C code.
CPSE

**RJUMP** (relative jump) ,depending on this we can jump to the address we want . In this case ,we jump to the PC-0x0005 and it is the beginning of our loop .So we can minus the value of R24 until it is equal to 0.

In my view of the process status on the left. I found that the values of **register 28** and **register 29** had also changed. By looking at the assembly language, I found that the code performed IN and OUT operations on R28 and R29 and pushed and popped their values against the stack. See figure7 and figure 8.

And the SER, LDI operation was also performed before the code was executed.
In fact, **register 0** and **register 1** also performs the operation, but the value of them are always 0x00 .

From the AVR Instruction Set Manual , I find that the OUT and IN instructions are used to operate the I/O register , in order to they have the address of each other and they can exchange data successfully.

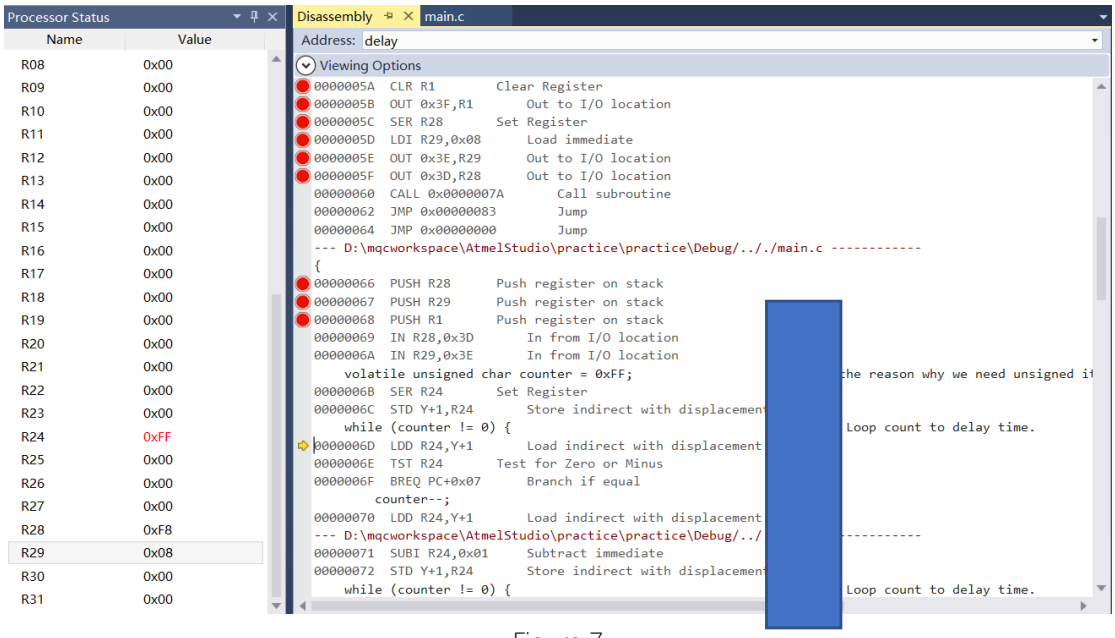



Figure 7

(I have marked all the instructions that operate on R1,R28 and R29 with breakpoints to make it easier to see them during the period.)

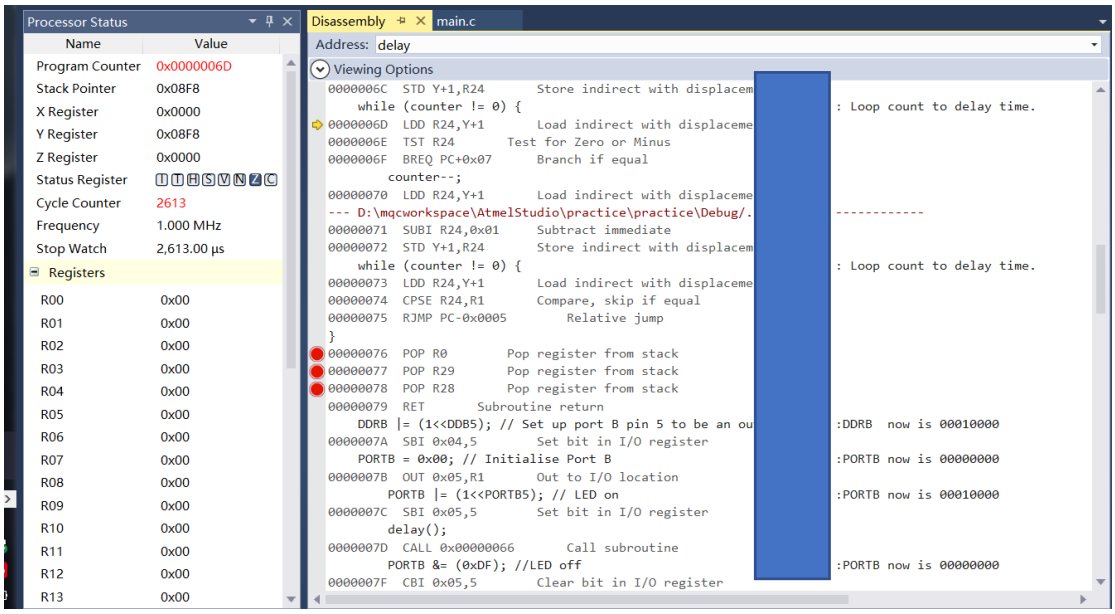



Figure 8

# 2.4 Corresponding hardware (Task E )

## I/O

We can see what hardware we used from the Figure . And we can see which pin we need to use in microprocessor . it is PB5 .See figure 9.

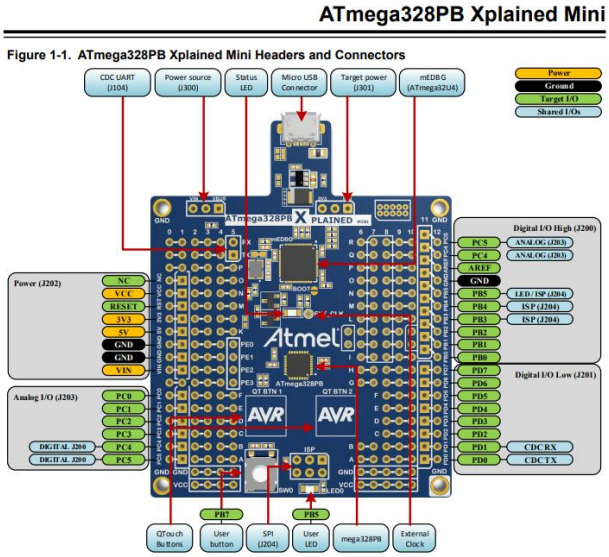



**Table 1-1. Default Configurations**

| Function | Default configuration | Other settings |
|---|---|---|
| Kit power source (J300) | 5.0V USB powered | External input VIN (1) |
| Target power (J301) | 5.0V USB powered | 3.3V from on-board regulator (1) |
| ATmega328PB clock | 16 MHz mEDBG clock (2) | Internal oscillator (3) |

Figure 9

We can see that the PB5 is connect with the User LED . See figure 10.

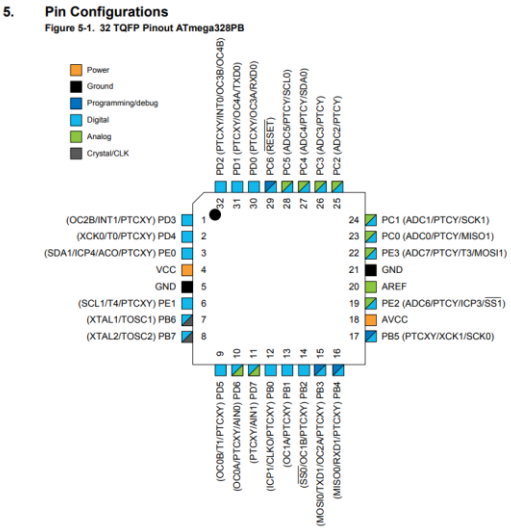



Figure 10

# Memory

From ‘The use of registers “ , we already know that during the program we need to exchange our address to the I/O register , and we can see it from Figure 11.

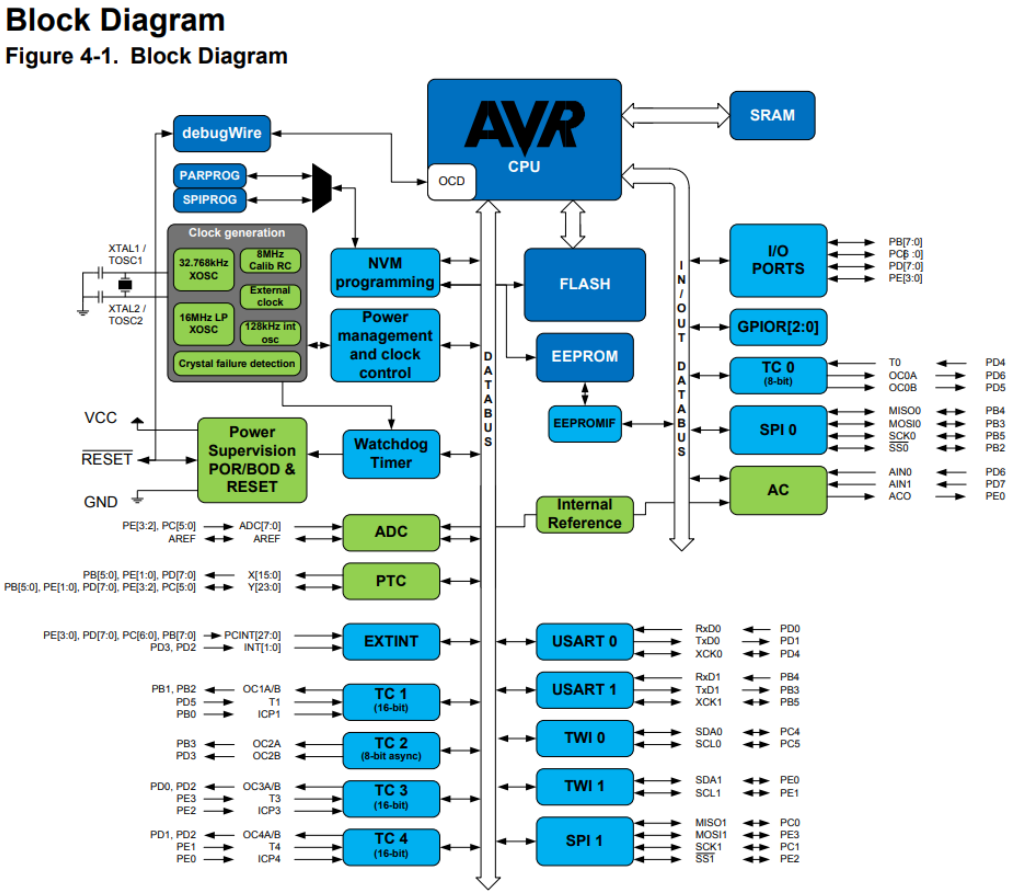



Figure 11

And obviously we make use of the stack in our handling of loops and calls to functions as well, we can also see the details from assembly language in ‘The use of registers “.

Above all .our program is stored in Flash , it cost 0x0108 * 8 bits . And the variables stored in IRAM .see Figure . But Unfortunately, the number in memory is an unreadable state for us. See Figure 12 and Figure 13.

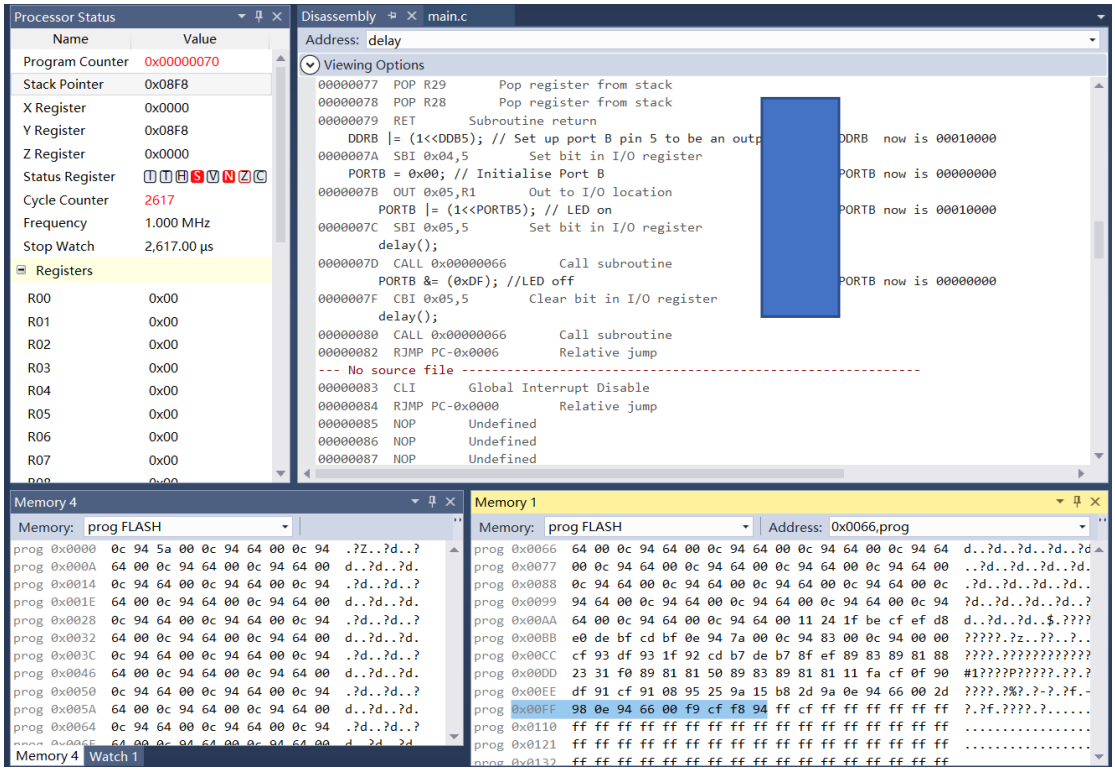



Figure 12

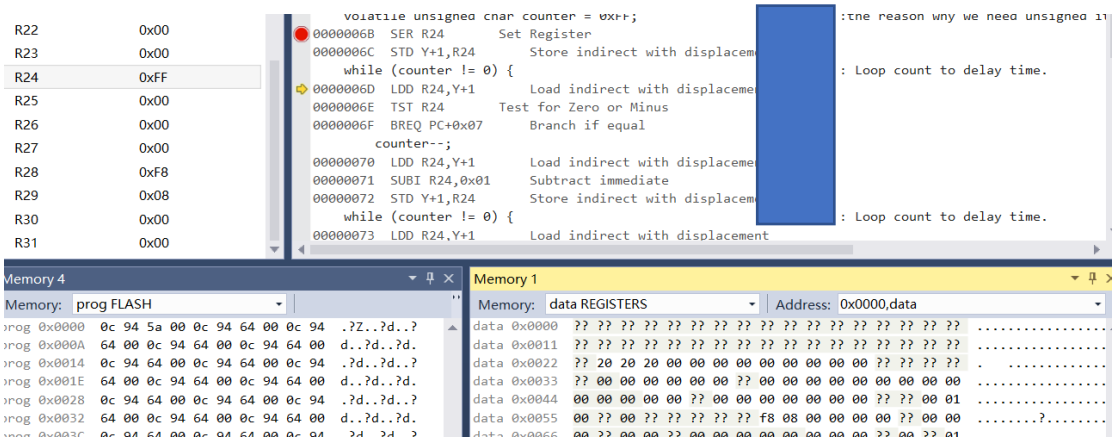



Figure 13

# 3.Assembly language

## 3.1 Covert to assembly code (Task A)

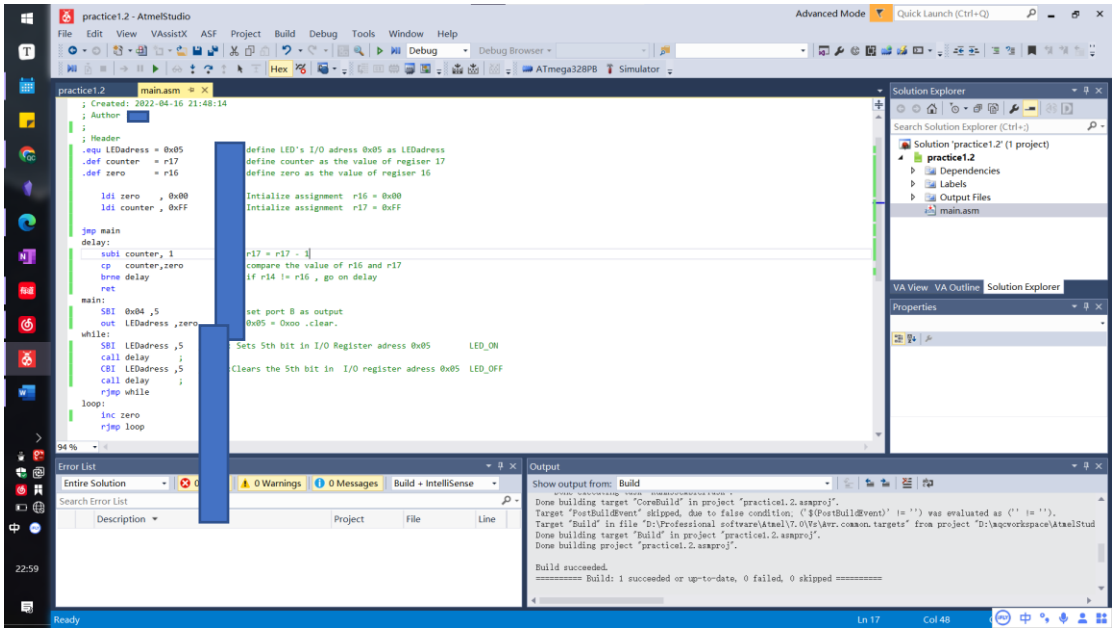



Figure 14(The assembly code write by the C code)

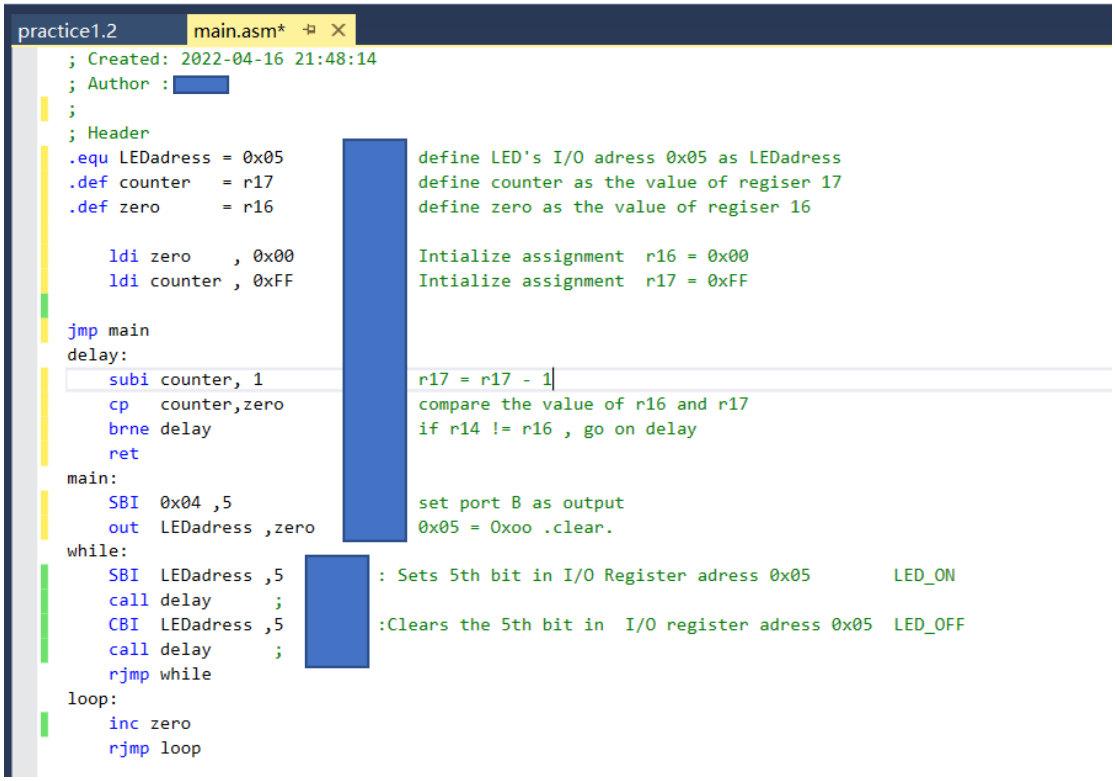



Figure 15(The assembly code write by the C code, see more clearly)

## 3.2 The use of registers on Assembly language (Task D)

From my assembly code , we can easily fine that we used Register 17 and Register 16 and our Statue Register in our comparing routine. It is because we need Z flag for equal and other Flag for the situation of zero ,Carry and so on. See figure 16.

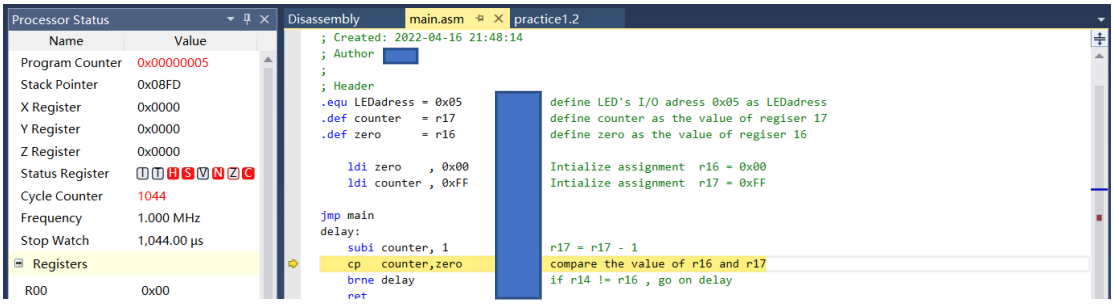



Figure 16

## 3.3 Explain assembly code (Task B)

In Header routine . I defined some code in order let the procedure clearly. You can see we replace the Register 17 and Register 16 by Counter and Zero . The reason why we do not use the instruction .equ it is because the .equ is able to operate Register and .def can not do it. We can see it in our previous video or operation manual.

In delay routine . I used SUBI in order to let the counter decrement every time in the loop until we return our value to the main routine. That make a function of delay time.

In main routine . I used the address of our DDRB and PORTB . It is necessary to do that .It tells our microprocessor where is the output and the value of each pin in the output.

In while routine . I called the delay routine in order to delay time and with different assignments, we can control the output of PINB to control the LED. See figure 17,18,19,20 .

Of course we have our loop routine ,just let the microprocessor do something to avoid the random instruction .

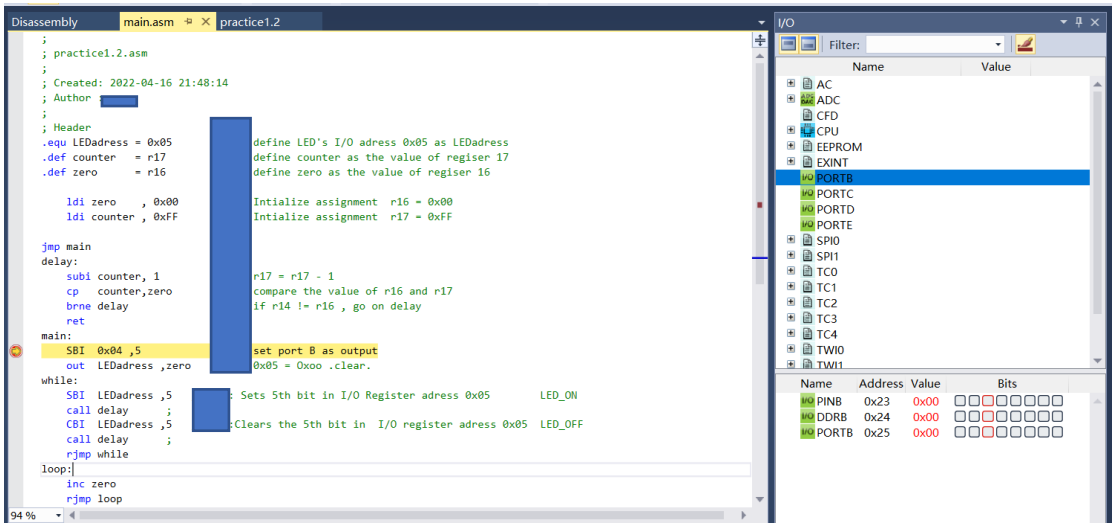



Figure 17

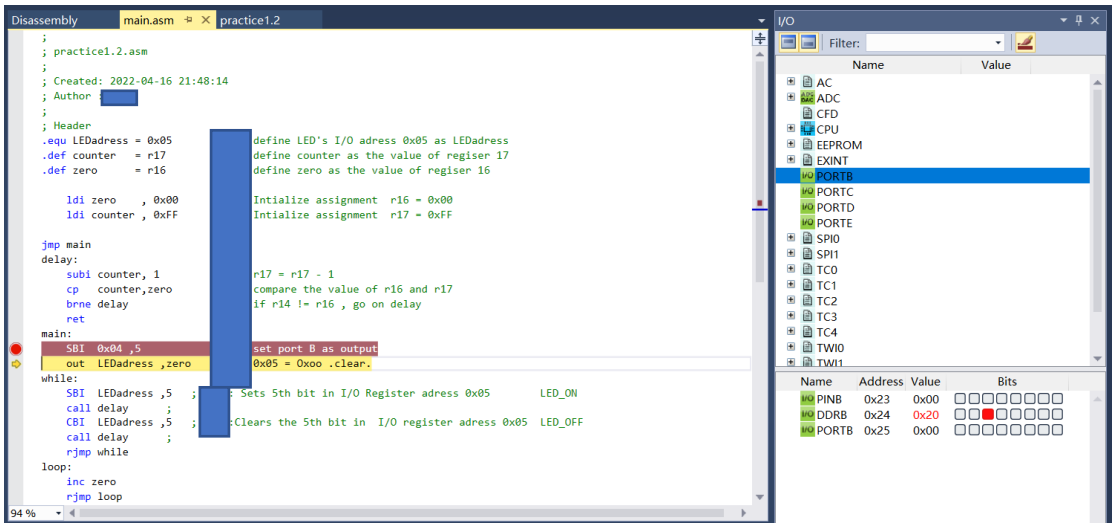



Figure 18

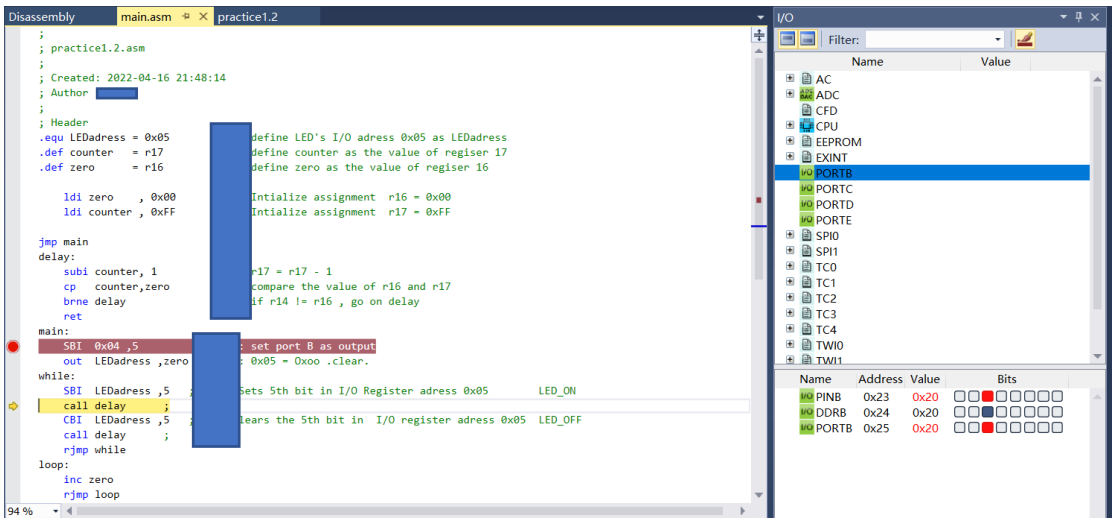



Figure 19

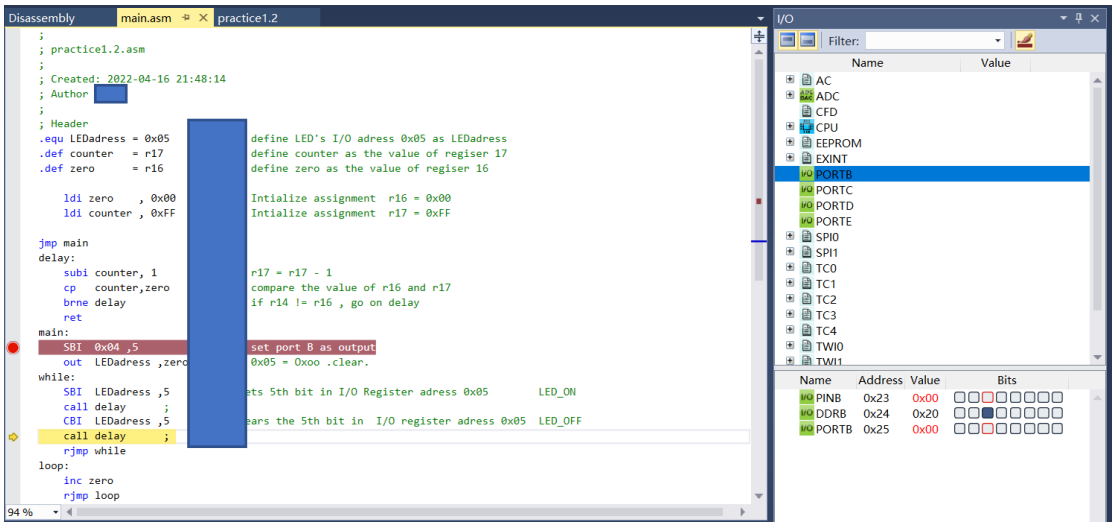


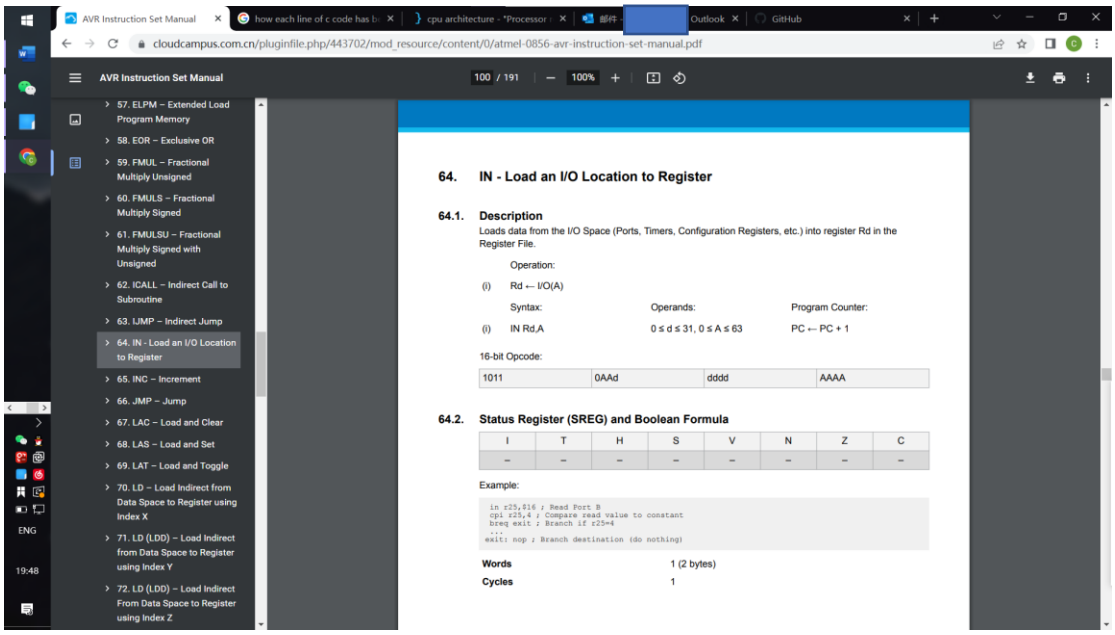

Figure 20

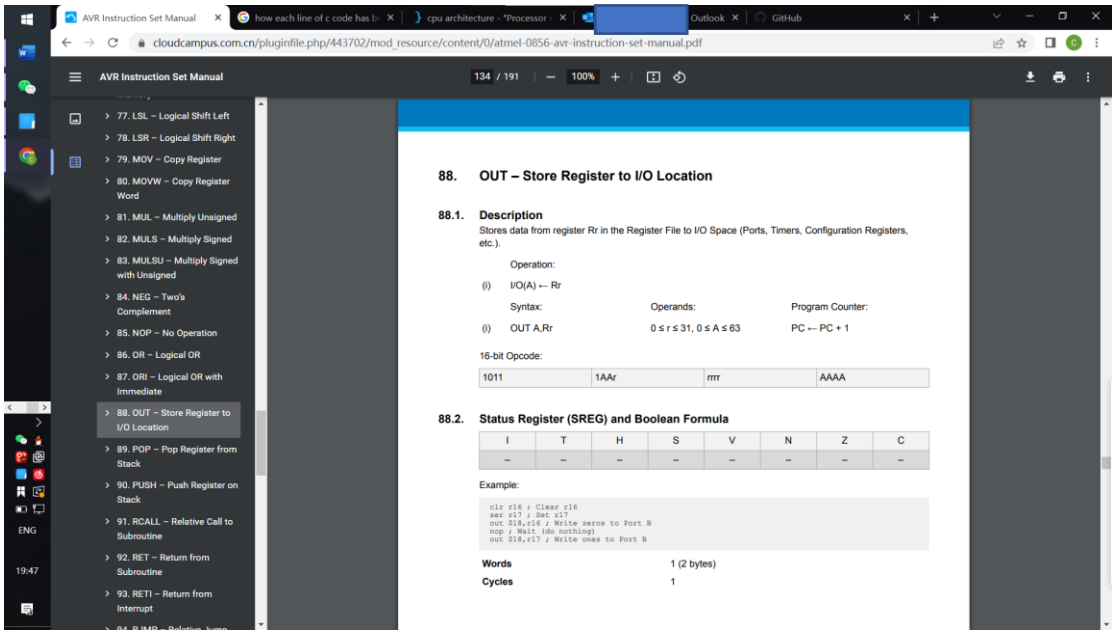
## 3.3 Compare to C code (Task C)

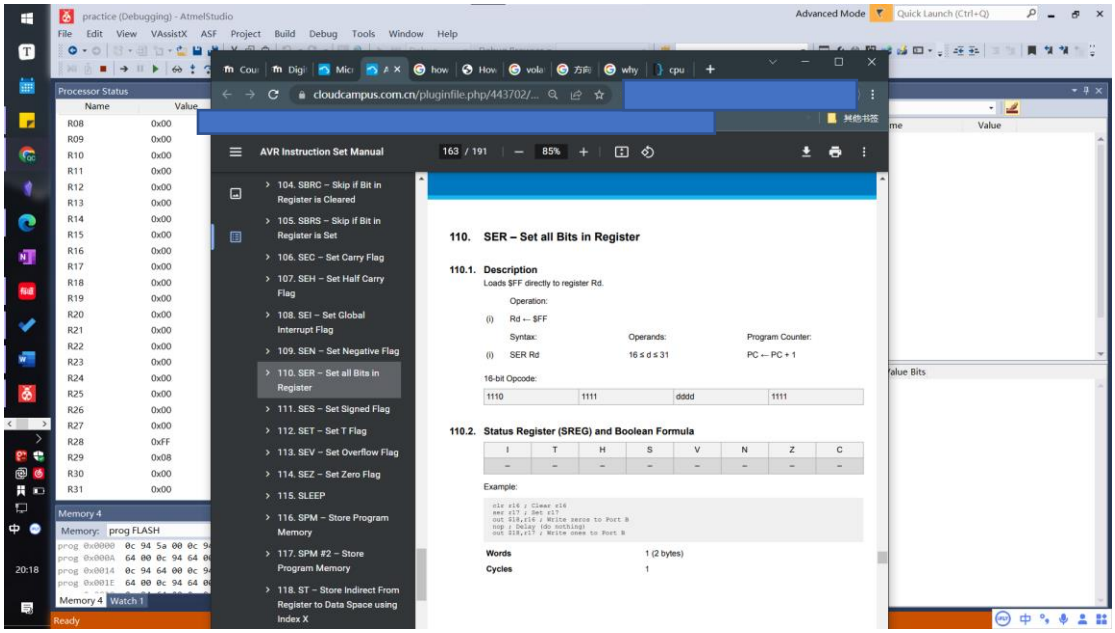
Let us look our C routine and Assembly routine. We can see obviously in comment and he content below are what we are interested especially. See figure 15.

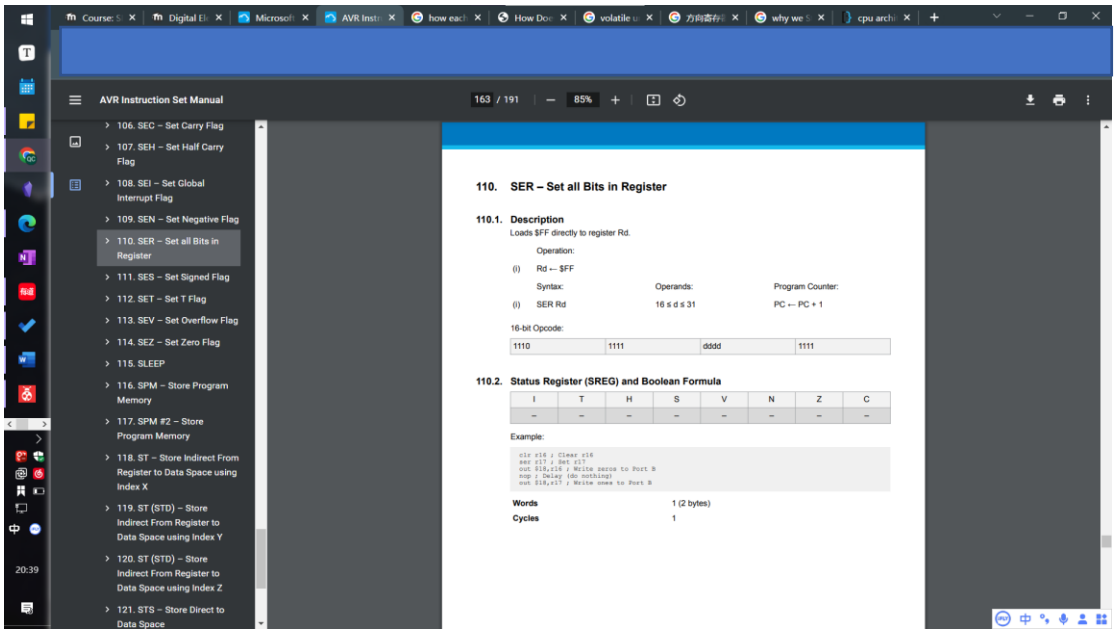
We can see there are many instruction have the same functionality. ldi it is equal as ‘ = ’ in C language . But ldi is Store data in registers.
And we have our cp instruction , it is work as ‘if( )” function , and SBI worked as ‘variable --‘.

# Appendix


AVR Instruction Set Manual
cloudcampus.com.cn/pluginfile.php/443702/mod_resource/content/0/atmel-0856-avr-instruction-set-manual.pdf
100 / 191
100%
57. ELPM – Extended Load Program Memory
58. EOR – Exclusive OR
59. FMUL – Fractional Multiply Unsigned
60. FMULS – Fractional Multiply Signed
61. FMULSU – Fractional Multiply Signed with Unsigned
62. ICALL – Indirect Call to Subroutine
63. IJMP – Indirect Jump
64. IN - Load an I/O Location to Register
65. INC – Increment
66. JMP – Jump
67. LAC – Load and Clear
68. LAS – Load and Set
69. LAT – Load and Toggle
70. LD – Load Indirect from Data Space to Register using Index X
71. LD (LDD) – Load Indirect from Data Space to Register using Index Y
72. LD (LDD) – Load Indirect From Data Space to Register using Index Z
64. IN - Load an I/O Location to Register
64.1. Description
Loads data from the I/O Space (Ports, Timers, Configuration Registers, etc.) into register Rd in the Register File.
Operation:
(i) Rd ← I/O(A)
Syntax:
(i) IN Rd,A
Operands:
0 ≤ d ≤ 31, 0 ≤ A ≤ 63
Program Counter:
PC ← PC + 1
16-bit Opcode:
1011 0AAd dddd AAAA
64.2. Status Register (SREG) and Boolean Formula
I T H S V N Z C
– – – – – – – –
Example:
in r25,$16 ; Read Port B
cpi r25,4 ; Compare read value to constant
breq exit ; Branch if r25=4
...
exit: nop ; Branch destination (do nothing)
Words 1 (2 bytes)
Cycles 1


AVR Instruction Set Manual
cloudcampus.com.cn/pluginfile.php/443702/mod_resource/content/0/atmel-0856-avr-instruction-set-manual.pdf
134 / 191
100%
77. LSL – Logical Shift Left
78. LSR – Logical Shift Right
79. MOV – Copy Register
80. MOVW – Copy Register Word
81. MUL – Multiply Unsigned
82. MULS – Multiply Signed
83. MULSU – Multiply Signed with Unsigned
84. NEG – Two's Complement
85. NOP – No Operation
86. OR – Logical OR
87. ORI – Logical OR with Immediate
88. OUT – Store Register to I/O Location
89. POP – Pop Register from Stack
90. PUSH – Push Register on Stack
91. RCALL – Relative Call to Subroutine
92. RET – Return from Subroutine
93. RETI – Return from Interrupt
88. OUT – Store Register to I/O Location
88.1. Description
Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers, etc.).
Operation:
(i) I/O(A) ← Rr
Syntax:
(i) OUT A,Rr
Operands:
0 ≤ r ≤ 31, 0 ≤ A ≤ 63
Program Counter:
PC ← PC + 1
16-bit Opcode:
1011 1AAr rrrr AAAA
88.2. Status Register (SREG) and Boolean Formula
I T H S V N Z C
– – – – – – – –
Example:
clr r16 ; Clear r16
ser r17 ; Set r17
out $18,r16 ; Write zeros to Port B
nop ; Wait (do nothing)
out $18,r17 ; Write ones to Port B
Words 1 (2 bytes)
Cycles 1


practice (Debugging) - AtmelStudio
cloudcampus.com.cn/pluginfile.php/443702/...
AVR Instruction Set Manual
163 / 191
85%
104. SBRC – Skip if Bit in Register is Cleared
105. SBRS – Skip if Bit in Register is Set
106. SEC – Set Carry Flag
107. SEH – Set Half Carry Flag
108. SEI – Set Global Interrupt Flag
109. SEN – Set Negative Flag
110. SER – Set all Bits in Register
111. SES – Set Signed Flag
112. SET – Set T Flag
113. SEV – Set Overflow Flag
114. SEZ – Set Zero Flag
115. SLEEP
116. SPM – Store Program Memory
117. SPM #2 – Store Program Memory
118. ST – Store Indirect From Register to Data Space using Index X
110. SER – Set all Bits in Register
110.1. Description
Loads $FF directly to register Rd.
Operation:
(i) Rd ← $FF
Syntax: Operands: Program Counter:
(i) SER Rd 16 ≤ d ≤ 31 PC ← PC + 1
16-bit Opcode:
1110 1111 dddd 1111
110.2. Status Register (SREG) and Boolean Formula
I T H S V N Z C
– – – – – – – –
Example:
clr r16 ; Clear r16
ser r17 ; Set r17
out $18,r16 ; Write zeros to Port B
nop ; Delay (do nothing)
out $18,r17 ; Write ones to Port B
Words 1 (2 bytes)
Cycles 1


AVR Instruction Set Manual
163 / 191
85%
106. SEC – Set Carry Flag
107. SEH – Set Half Carry Flag
108. SEI – Set Global Interrupt Flag
109. SEN – Set Negative Flag
110. SER – Set all Bits in Register
111. SES – Set Signed Flag
112. SET – Set T Flag
113. SEV – Set Overflow Flag
114. SEZ – Set Zero Flag
115. SLEEP
116. SPM – Store Program Memory
117. SPM #2 – Store Program Memory
118. ST – Store Indirect From Register to Data Space using Index X
119. ST (STD) – Store Indirect From Register to Data Space using Index Y
120. ST (STD) – Store Indirect From Register to Data Space using Index Z
121. STS – Store Direct to Data Space
110. SER – Set all Bits in Register
110.1. Description
Loads $FF directly to register Rd.
Operation:
(i) Rd ← $FF
Syntax: Operands: Program Counter:
(i) SER Rd 16 ≤ d ≤ 31 PC ← PC + 1
16-bit Opcode:
1110 1111 dddd 1111
110.2. Status Register (SREG) and Boolean Formula
I T H S V N Z C
– – – – – – – –
Example:
clr r16 ; Clear r16
ser r17 ; Set r17
out $18,r16 ; Write zeros to Port B
nop ; Delay (do nothing)
out $18,r17 ; Write ones to Port B
Words 1 (2 bytes)
Cycles 1