

เรื่อง Arithmetic and Logic Operation - Arduino Analog IO

จัดทำโดย

นางสาวสุนันท์ เป็ดโปง
รหัสนักศึกษา B6023973
สาขาวิชา วิศวกรรมคอมพิวเตอร์

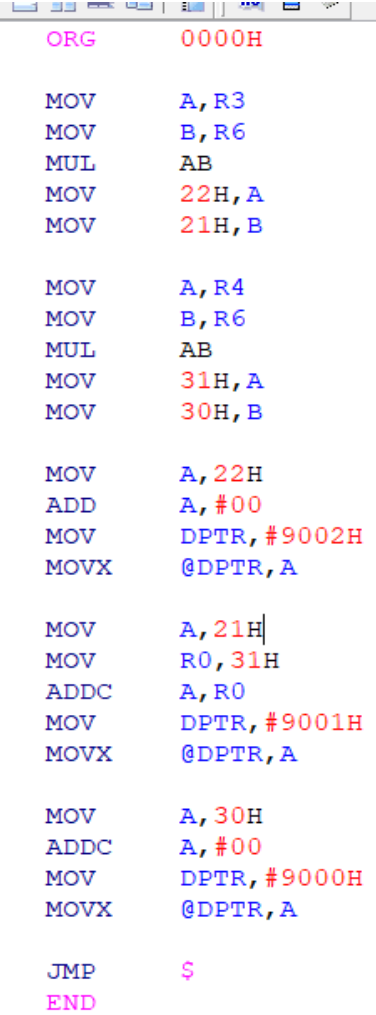
เสนอ

อาจารย์ นายวิชัย ศรีสุรักษ์
นายอำนวย ทีจันทิก
นายศพงษ์ ไชยฤกษ์
นายทองยศ ศรีเพ็ง

รายงานนี้เป็นส่วนหนึ่งของการเรียนวิชา ไมโครโปรเซสเซอร์
ภาคเรียนที่ 1 ปีการศึกษา 2562
มหาวิทยาลัยเทคโนโลยีสุรนารี

Part-A: การทดลอง เฉพาะข้อที่ทดลองในห้อง

ตอนที่ 1 การทดลองข้อ 5

 <pre> ORG 0000H MOV A,R3 MOV B,R6 MUL AB MOV 22H,A MOV 21H,B MOV A,R4 MOV B,R6 MUL AB MOV 31H,A MOV 30H,B MOV A,22H ADD A,#00 MOV DPTR,#9002H MOVX @DPTR,A MOV A,21H MOV R0,31H ADDC A,R0 MOV DPTR,#9001H MOVX @DPTR,A MOV A,30H ADDC A,#00 MOV DPTR,#9000H MOVX @DPTR,A JMP \$ END </pre>	<pre> ORG 0000H MOV A,R3 MOV B,R6 MUL AB ;นำค่าตำแหน่ง R3 คูณ R6 ผ่าน A B MOV 22H,A MOV 21H,B ;นำค่าที่คูณได้ไปเก็บตำแหน่ง 21H และ 22H MOV A,R4 MOV B,R6 MUL AB ;นำค่าตำแหน่ง R4 คูณ R6 ผ่าน A B MOV 31H,A MOV 30H,B ;นำค่าที่คูณได้ไปเก็บตำแหน่ง 31H และ 32H MOV A,22H ADD A,#00 MOV DPTR,#9002H MOVX @DPTR,A MOV A,21H MOV R0,31H ADDC A,R0 ;บวกเลขแต่ละตำแหน่งและ บวกแครี่ด้วย </pre>
---	--

ตอนที่ 4 การทดลองข้อ 4

```

1 //Analog Input
2 #define ANALOG_PIN_0 36
3 int analog_value = 0;
4
5 // PMW LED
6 #define LED_PIN 2
7 int freq = 5000;
8 int ledChannel = 0;
9 int resolution = 8;
10 int dutyCycle = 0;
11
12 void setup()
13 {
14     Serial.begin(115200);
15     delay(1000); // give me time to bring up serial
16     Serial.println("ESP32 Analog IN/OUT Test");
17
18     ledcSetup(ledChannel, freq, resolution);
19     ledcAttachPin(LED_PIN, ledChannel);
20     ledcWrite(ledChannel, dutyCycle);
21 }
22
23 void loop()
24 {
25     analog_value = analogRead(ANALOG_PIN_0);
26     Serial.println(analog_value);
27     dutyCycle = map(analog_value, 0, 4095, 0, 255);
28     ledcWrite(ledChannel, dutyCycle);
29     delay(500);
30 }

```

Done uploading.

```

#define ANALOG_PIN_0 36

int analog_value = 0;

// PMW LED

#define LED_PIN 2

int freq = 5000;

int ledChannel = 0;

int resolution = 8;

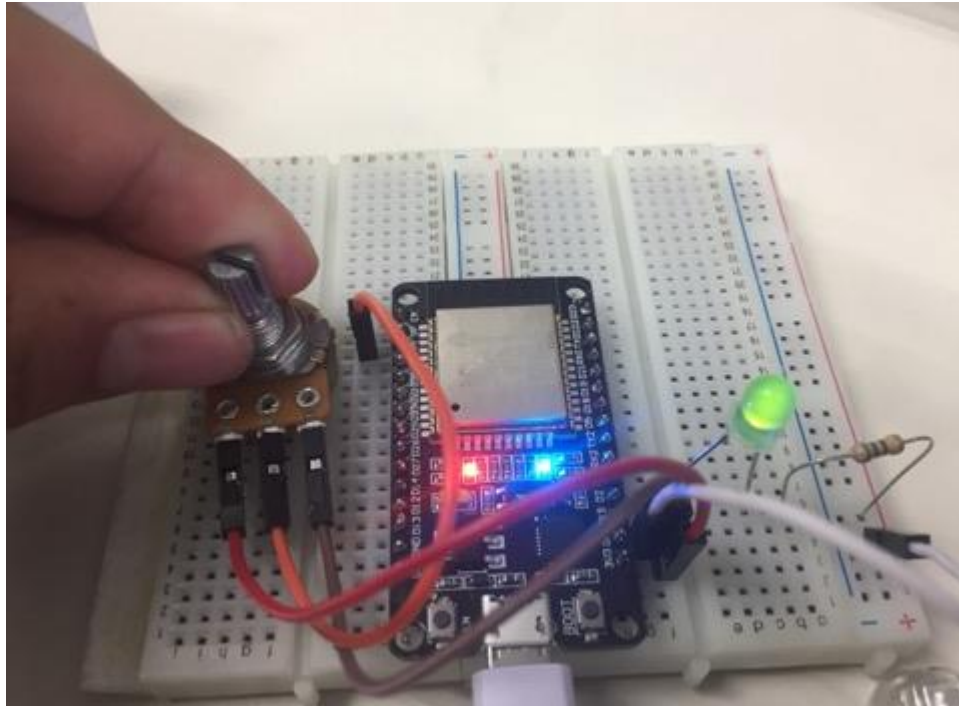
int dutyCycle = 0;

void setup()
{
    Serial.begin(115200);
    delay(1000);
    Serial.println("ESP32 Analog IN/OUT Test");

    ledcSetup(ledChannel, freq, resolution);
    ledcAttachPin(LED_PIN, ledChannel);
    ledcWrite(ledChannel, dutyCycle);
}

void loop()
{
    analog_value = analogRead(ANALOG_PIN_0);
    Serial.println(analog_value);
    dutyCycle = map(analog_value, 0, 4095, 0, 255);
    ledcWrite(ledChannel, dutyCycle);
    delay(500);
}

```



Part-B: คำถามท้ายการทดลองทุกข้อ

คำถามข้อ 1

1	Result_H EQU 20H	Result_H EQU 20H
2	Result_L EQU 21H	Result_L EQU 21H
3		
4	ORG 0000H	ORG 0000H
5	MOV Result_H,#00	MOV Result_H,#00
6	MOV Result_L,#00	MOV Result_L,#00
7		
8	MOV A,R0	MOV A,R0
9	CLR C	CLR C
10	ADDC A,Result_L	ADDC A,Result_L
11	DA A	DA A
12	MOV Result_L,A	MOV Result_L,A
13	MOV A,#00	MOV A,#00
14	ADDC A,Result_H	ADDC A,Result_H
15	DA A	DA A
16	MOV Result_H,A	MOV Result_H,A
17		
18	MOV A,R1	MOV A,R1
19	CLR C	CLR C
20	ADDC A,Result_L	ADDC A,Result_L
21	DA A	DA A
22	MOV Result_L,A	MOV Result_L,A
23	MOV A,#00	MOV A,#00
24	ADDC A,Result_H	ADDC A,Result_H
25	DA A	DA A
26	MOV Result_H,A	MOV Result_H,A
27		
28	MOV A,R2	MOV A,R2
29	CLR C	CLR C
30	ADDC A,Result_L	ADDC A,Result_L
31	DA A	DA A
32	MOV Result_L,A	MOV Result_L,A
33	MOV A,#00	MOV A,#00
34	ADDC A,Result_H	ADDC A,Result_H
35	DA A	DA A
36	MOV Result_H,A	MOV Result_H,A

36	MOV	Result_H,A	MOV	Result_H,A
37				
38	MOV	A,R3		
39	CLR	C		
40	ADDC	A,Result_L	MOV	A,R2
41	DA	A	CLR	C
42	MOV	Result_L,A	ADDC	A,Result_L
43	MOV	A,#00	DA	A
44	ADDC	A,Result_H	MOV	Result_L,A
45	DA	A	MOV	A,#00
46	MOV	Result_H,A	ADDC	A,Result_H
47			DA	A
48	MOV	A,R4	MOV	Result_H,A
49	CLR	C		
50	ADDC	A,Result_L		
51	DA	A	MOV	A,R3
52	MOV	Result_L,A	CLR	C
53	MOV	A,#00	ADDC	A,Result_L
54	ADDC	A,Result_H	DA	A
55	DA	A	MOV	Result_L,A
56	MOV	Result_H,A	MOV	A,#00
57			ADDC	A,Result_H
58	MOV	A,R5	DA	A
59	CLR	C	MOV	Result_H,A
60	ADDC	A,Result_L		
61	DA	A		
62	MOV	Result_L,A		
63	MOV	A,#00		
64	ADDC	A,Result_H		
65	DA	A		
66	MOV	Result_H,A		
67				
68	MOV	A,R6		
69	CLR	C		
70	ADDC	A,Result_L		
71	DA	A		
72	MOV	Result_L,A		
73	MOV	A,#00		
74	ADDC	A,Result_H		
75	DA	A		
76	MOV	Result_H,A		
77				
78	MOV	A,R7		
79	CLR	C		
80	ADDC	A,Result_L		
81	DA	A		
82	MOV	Result_L,A		
83	MOV	A,#00		
84	ADDC	A,Result_H		
85	DA	A		
86	MOV	Result_H,A		
87				
88	MOV	A,Result_H		
89	MOV	40H,A		
90	MOV	A,Result_L		
91	MOV	41H,A		
92				
93	JMP	\$		
94	END			

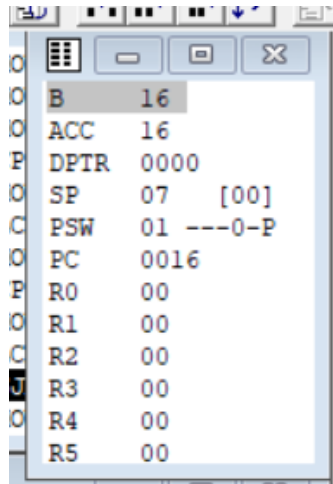
	<div>MOV Result_H,A</div> <div>MOV A,R5</div> <div>CLR C</div> <div>ADDC A,Result_L</div> <div>DA A</div> <div>MOV Result_L,A</div> <div>MOV A,#00</div> <div>ADDC A,Result_H</div> <div>DA A</div> <div>MOV Result_H,A</div> <div>MOV A,R6</div> <div>CLR C</div> <div>ADDC A,Result_L</div> <div>DA A</div> <div>MOV Result_L,A</div> <div>MOV A,#00</div> <div>ADDC A,Result_H</div> <div>DA A</div> <div>MOV Result_H,A</div> <div>MOV A,R7</div> <div>CLR C</div> <div>ADDC A,Result_L</div> <div>DA A</div> <div>MOV Result_L,A</div> <div>MOV A,#00</div> <div>ADDC A,Result_H</div> <div>DA A</div>
--	---

คำถามข้อ 3

<pre> HResult EQU 20H LResult EQU 21H ORG 0000H MOV HResult,#00 MOV LResult,#00 MOV R1,B MOV A,#00 LOOP: CLR C MOV A,R1 ADDC A,LResult MOV LResult,A MOV A,#00 ADDC A,HResult MOV HResult,A INC R1 CJNE R1,#0FFH,LOOP CLR C MOV A,R1 ADDC A,LResult MOV LResult,A MOV A,#00 ADDC A,HResult MOV HResult,A SJMP \$ END </pre>	<pre> HResult EQU 20H LResult EQU 21H ORG 0000H MOV HResult,#00 MOV LResult,#00 MOV R1,B MOV A,#00 LOOP: CLR C MOV A,R1 ADDC A,LResult MOV LResult,A MOV A,#00 ADDC A,HResult MOV HResult,A INC R1 CJNE R1,#0FFH,LOOP CLR C MOV A,R1 ADDC A,LResult MOV LResult,A MOV A,#00 ADDC A,HResult MOV HResult,A SJMP \$ END </pre>
---	---

คำถามข้อ 5

<pre> MyData EQU 20H ORG 0000H MOV MyData,#00 MOV B,#00 MOV A,DPH CPL A MOV MyData,A CALL COUNT1 MOV A,DPL CPL A MOV MyData,A CALL COUNT1 JMP \$ COUNT1: MOV R6,#8 _XLOOP: MOV A,MyData RLC A MOV MyData,A MOV A,B ADDC A,#00 DA A MOV B,A DJNZ R6,_XLOOP RET END </pre>	<pre> MyData EQU 20H ORG 0000H MOV MyData,#00 MOV B,#00 MOV A,DPH CPL A MOV MyData,A CALL COUNT1 MOV A,DPL CPL A MOV MyData,A CALL COUNT1 JMP \$ COUNT1: MOV R6,#8 _XLOOP: MOV A,MyData RLC A MOV MyData,A MOV A,B ADDC A,#00 DA A MOV B,A DJNZ R6,_XLOOP RET END </pre>
--	---



B	16
ACC	16
DPTR	0000
SP	07 [00]
PSW	01 ---0-P
PC	0016
R0	00
R1	00
R2	00
R3	00
R4	00
R5	00

คำถามข้อ 6

<pre> MyData EQU 20H HResult EQU 21H LResult EQU 22H ORG 0000H MOV R0,#40H MOV R2,#40H MOV MyData,#00 MOV HResult,#00 MOV LResult,#00 LOOP: MOV A,@R0 MOV MyData,A CALL COUNT1 INC R0 DJNZ R2,LOOP MOV DPTR,#9000H MOV A,HResult MOVX @DPTR,A MOV DPTR,#9001H MOV A,LResult MOVX @DPTR,A JMP \$ COUNT1: MOV R6,#8 _XLOOP: MOV A,MyData RLC A MOV MyData,A MOV A,LResult ADDC A,#00 DA A MOV LResult,A MOV A,HResult ADDC A,#00 DA A MOV HResult,A DJNZ R6,_XLOOP RET FINT </pre>	<pre> MyDataEQU 20H HResult EQU 21H LResultEQU 22H ORG 0000H MOV R0,#40H MOV R2,#40H MOV MyData,#00 MOV HResult,#00 MOV LResult,#00 LOOP: MOV A,@R0 MOV MyData,A CALL COUNT1 INC R0 DJNZ R2,LOOP MOV DPTR,#9000H MOV A,HResult MOVX @DPTR,A MOV DPTR,#9001H MOV A,LResult MOVX @DPTR,A JMP \$ COUNT1: MOV R6,#8 </pre>
---	--

```

_XLOOP:      MOV     A,MyData

              RLC     A

              MOV     MyData,A


              MOV     A,LResult
              ADDC    A,#00
              DA      A

              MOV     LResult,A


              MOV     A,HResult
              ADDC    A,#00
              DA      A

              MOV     HResult,A
              DJNZ    R6,_XLOOP
              RET
              END

```

[illegible]

คำถามข้อ 8.1

คุณรู้หรือเปล่าว่าปัญหา memory เกิดขึ้นเมื่อไหร่? แน่นอนว่าหลาย ๆ คนคงมองข้ามสำหรับปัญหานี้ ซึ่งเราอย่าลืมว่าการเขียนโปรแกรมบน Microcontroller มันไม่ได้มีหน่วยความจำอะไรมากมาย เหมือนกับคอมพิวเตอร์เลย มันเลยกลายเป็นเรื่องราวที่ทำให้เจ้าของบล็อกอยากเขียนบทความนี้ขึ้นมา

" Working in this minimalist environment, you must use your resources wisely. การเขียนโปรแกรมบนสภาพแวดล้อมที่จำกัด คุณต้องใช้ทรัพยากรอย่างฉลาด "

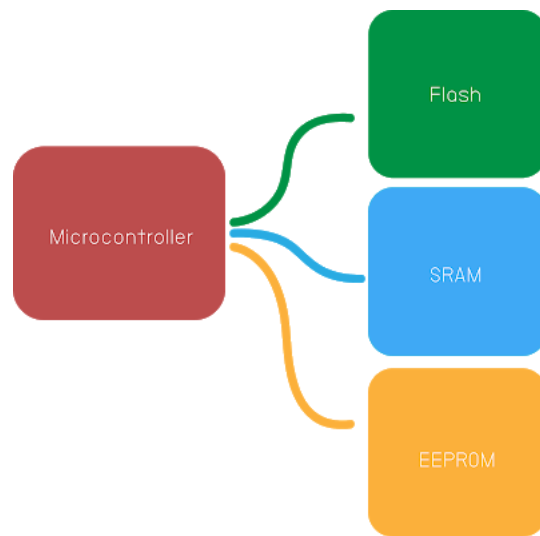
สำหรับสิ่งที่เจ้าของบล็อกอยากจะทำก่อนเลยไม่ใช่การแก้ปัญหา แต่เป็นความเข้าใจว่าอะไรทำให้เกิดปัญหา สิ่งสำคัญในการพัฒนา embedded คือคุณต้องใช้ทรัพยากรที่มีอยู่จำกัดนี้ อย่างคุ้มค่าที่สุด และสิ่งที่คุณต้องรู้ก่อนเลยคือ ต้องรู้ว่าขีดจำกัดของ platform ที่เรากำลังพัฒนาอยู่มีเท่าไร

Arduino	Processor	Flash	SRAM	EEPROM
UNO, Uno Ethernet, Menta, Boarduino	Atmega328	32K	2K	1K
Leonardo, Micro, Flora, 32U4 Breakout, Teensy, Esplora	Atmega 32U4	32K	2.5K	1K
Mega, MegaADK	Atmega2560	256K	8K	4K

<https://cdn-learn.adafruit.com/downloads/pdf/memories-of-an-arduino.pdf>

Arduino memory

ในเรื่อง memory ของ Arduino จะถูกแบ่งออกเป็น 3 อย่างก็คือ *Flash* ,*SRAM* และ *EEPROM* ในตัวอย่างของบทความนี้เจ้าของบล็อกจะใช้ Processor เป็น Atmega328 หรือก็คือ Arduino UNO นั่นเอง จะเห็นว่า Flash มีพื้นที่ 32K byte SRAM 2K byte และ EEPROM อีก 1K byte เมื่อเรารู้ทรัพยากรของบอร์ดเราแล้ว เราก็มามาทำความรู้จัก memory ในแต่ละแบบกัน



Flash

Flash คือ memory ส่วนที่จะเก็บโปรแกรมที่เราเขียนไว้รวมทั้ง boot loader ในส่วนนี้เมื่อบอร์ดไม่มีไฟเลี้ยงก็ยังสามารถเก็บข้อมูลได้ นั่นเป็นเหตุว่าทำไมเรา reset ใหม่กี่ครั้งหรือไฟดับกี่ครั้ง เมื่อบอร์ดจัมไฟ ก็ยังสามารถทำงานโค้ดเดิมที่เคย Flash โปรแกรมไว้ได้ แต่ข้อจำกัดของการ Flash ก็คือสามารถ Flash ได้สูงสุดประมาณ 10,000 ครั้ง จริง ๆ เท่านี้ก็เพียงพอแล้ว เพราะบางทีบอร์ดอาจจะเจ๊งก่อนที่จะ Flash ครบก็ได้นะ (ฮา ๆ ๆ)

เราสามารถเก็บข้อมูลต่าง ๆ ลงใน Flash ได้ แต่ว่าเราจะไม่สามารถแก้ไขข้อมูลที่อยู่ใน Flash ได้เลย ในระหว่างที่บอร์ดกำลังทำงาน แต่ถ้าอยากจะใช้ข้อมูลใน Flash จริง ๆ ก็ต้อง copy ค่าใน Flash เก็บ

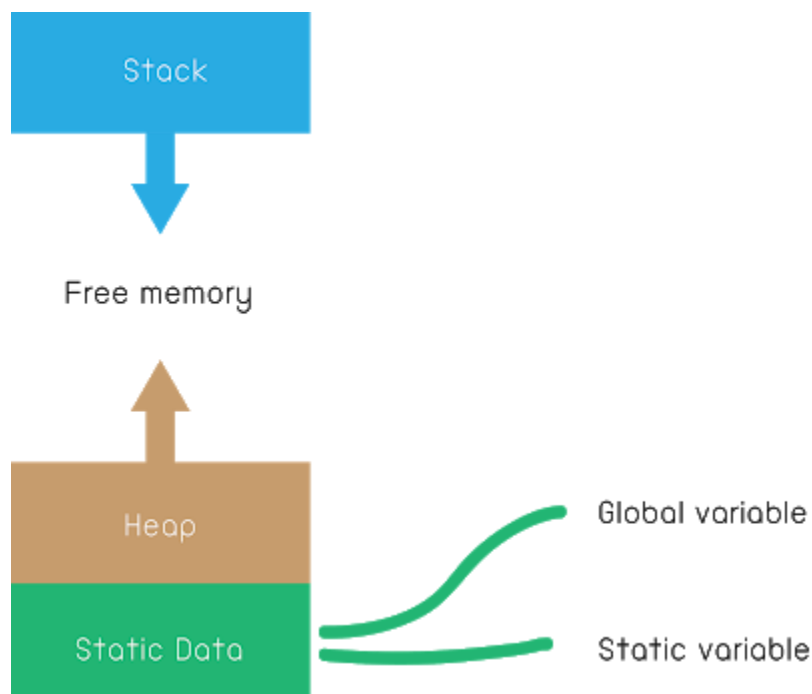
ไว้ใน SRAM หรือก็คือเก็บไว้ในซีกตัวแปรนั้นแหละ แล้วค่อยแก้ค่าในตัวแปรที่เก็บไว้

SRAM

ชื่อเต็ม ๆ ก็คือ Static Random Access Memory เจ้าตัว SRAM นี้สามารถทั้งอ่านทั้งเขียนข้อมูลได้ การสร้างตัวแปรต่าง ๆ สร้าง function ต่าง ๆ หรือแม้แต่การจองพื้นที่ขณะที่กำลังทำงาน ในส่วนนี้จะถูกทำใน SRAM ทั้งหมด ทั้งนี้ SRAM ยังแยกชนิดการเก็บไปอีก 3 แบบนะเนี่ย มี Static Data , Heap แล้วก็ Stack

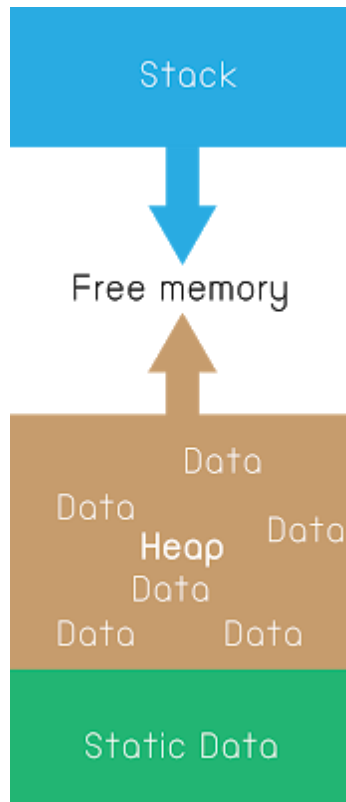
Static Data

ในส่วน Static Data ถ้าเรามองเป็นภาพรวม มันก็คือบล็อก ๆ นึง(สีเขียว) ที่เก็บพวกตัวแปร Global และ ตัวแปร static พวกนี้จะมีขนาดที่คงที่เพราะตัวแปรในนี้มันจะถูกประกาศครั้งเดียวตอนเริ่มต้นโปรแกรม ในขณะที่โปรแกรมทำงาน เจ้าของบล็อกสามารถเปลี่ยนแปลงค่าได้ตามปรกติ แต่ไม่สามารถเพิ่มลดขนาดของข้อมูลได้อีกแล้ว แต่ถ้าอยากเพิ่มมันก็จะไปอยู่ในเรื่องของพวก Heap กับ Stack แทน



Heap

ในส่วน heap คือส่วนที่เป็น Dynamic memory หรือก็คือเป็นหน่วยความจำที่เพิ่มลดขนาดตัวเองได้(สื่อน้ำตาล) ต่างจาก static memory ที่มีขนาดคงที่



การทำงานบางครั้ง เราไม่สามารถรู้ได้ว่า เราจะต้องใช้หน่วยความจำเท่าไร การขอพื้นที่หน่วยความจำตามจำนวนข้อมูลที่เราใช้ก็ถือว่าเป็นวิธีที่ดี แต่จะดีกว่าถ้าไม่ทำในงาน Embedded เพราะบางทีมันอาจจะสิ้นเปลือง Memory มากเกินความจำเป็น และเป็นสิ่งที่ควบคุมยาก เพราะเวลาเราจองพื้นที่มาแล้ว ในส่วนของ Heap ก็จะมีสูงขึ้นเรื่อย ๆ สำคัญเลยคือ เราต้องคืนพื้นที่หน่วยความจำทุกครั้งที่เราใช้งานเสร็จด้วย

Stack

Stack เป็นส่วนที่เก็บข้อมูลจำพวกตัวแปร local ต่าง ๆ เวลาที่เราเรียกใช้ function หรือสร้างตัวแปร local ข้อมูลทุกอย่างจะถูกเก็บไว้ใน Stack ทั้งหมด

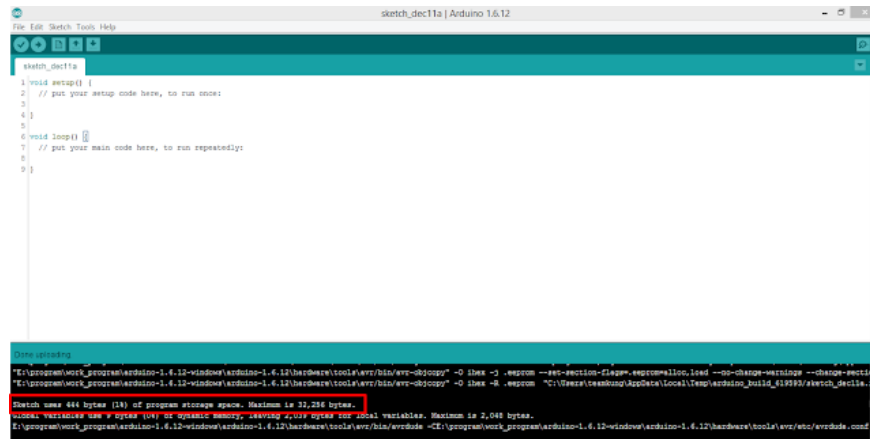
ความคิดเห็นส่วนตัวของเจ้าของบล็อกคิดว่าตรงนี้ไม่ต้องระวังอะไรมาก ไม่เหมือนส่วน Heap เพราะเป็นส่วนที่ต้องจัดการดี ๆ ใช้งานเสร็จคืนพื้นที่ ไม่งั้นส่วนอื่นก็จะมีทรัพยากรน้อยลง แต่เมื่อมาลองดูในส่วน of Stack ซึ่งการทำงานของ local มันจะตายเองหลังจากที่ออกจากขอบเขตการทำงานของมัน ทำให้ส่วนนี้ไม่จำเป็นต้องจัดการอะไรมาก แต่ก็ไม่ใช่ที่เราจะวางใจได้เลย เพราะ stack ยังมีปัญหาอย่าง stack overflow ที่จะพูดถึงในบทความต่อ ๆ ไป

EEPROM

EEPROM เป็นหน่วยความจำแบบ non-volatile memory หรือก็คือเป็นหน่วยความจำถาวร เวลาไม่มีไฟเลี้ยงมันก็สามารถเก็บข้อมูลไว้ได้อยู่ คล้าย ๆ กับ Flash แต่ข้อดีของมันคือ มันสามารถอ่านเขียนข้อมูลได้ในขณะที่บอร์ดกำลังทำงานอยู่เหมือน SRAM แต่ว่าในการอ่านเขียนข้อมูลแต่ละครั้งจะทำได้เพียงทีละ byte เท่านั้น ทำให้มันช้ากว่า SRAM มาก และมันสามารถอ่านเขียนข้อมูลได้สูงสุดประมาณ 100,000 ครั้ง

คำถามข้อ 8.2

Flash



ตรงที่เป็นกรอบสีแดงนั้นแหละที่บอกว่ามีข้อมูลเก็บลง flash เท่าไหร่ จากตัวอย่างเจ้าของบล็อก flash โปรแกรมเปล่า ๆ ที่มีแต่ setup() กับ loop() แค่นั้น ใช้พื้นที่ flash ไป 444 byte หรือแค่ 1% จากพื้นที่ทั้งหมด 32k byte เห็นว่าเยอะอย่างงั้นแต่พอเขียนกับงานจริง ๆ พื้นที่ flash หมดไปง่ายมาก

Static data

เราลองมาเล่นอะไรดูหน่อยแล้วกัน ยังจำข้อมูลในส่วนของ static data กันได้อยู่หรือเปล่าเอ่ย.. ข้อมูลจำพวก static data ก็คือพวกตัวแปร global variable กับ static variable นั่นเอง

```
int data1 = 10;
```

```
void setup(){
```

```
}
```

```
void loop(){
```

```
}
```

ถ้าหากว่าเจ้าของบล็อกเขียนโปรแกรมประมาณนี้แล้ว flash ลงไปในบอร์ด คิดว่าขนาด size ของโปรแกรมจะเป็นยังไง เพิ่มขึ้น ลดลง หรือ เท่าเดิม? ..

```
Sketch uses 444 bytes (1%) of program storage space. Maximum is 32,256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2,039 bytes for local variables. Maximum is 2,048 bytes.
```

นี่คือผลของโปรแกรม จากที่ถามไปถ้าเราเขียนเพิ่มตัวแปรลงไปในโค้ดมันคงไม่ลดลงหรอกเนอะ เพราะฉะนั้นตัดตรงนั้นทิ้งไปได้เลย เหลือแค่ เท่าเดิม กับ เพิ่มขึ้น แต่ผลที่ได้คือขนาด flash เท่าเดิมกับที่เป็นแบบนี้เพราะว่า เราประกาศตัวแปรขึ้นมา.. แล้วไงละ เราแค่ประกาศขึ้นมาแต่ไม่ได้ใช้อะไรเลย compiler ก็เลยมองว่าตัวแปรตัวนี้ก็เป็นแค่ค่าคงที่

```
int data1 = 10;

void setup(){
    data1 = 20;
}

void loop(){

}
```

เจ้าของบล็อกลองเพิ่มเข้าไปอีกนิดนึงดูนะ และคราวนี้ตัวแปรที่เราสร้างขึ้นถูกใช้งานโดยการ assign ค่าเข้าไปใหม่ และนี่คือผล หลังจากที่เราใช้งานตัวแปรที่เราสร้างขึ้นมา

```
Sketch uses 482 bytes (1%) of program storage space. Maximum is 32,256 bytes.
Global variables use 11 bytes (0%) of dynamic memory, leaving 2,037 bytes for local variables. Maximum is 2,048 bytes.
```

คราวนี้เราลองมาเล่นกันอีก ถ้าหากว่าเจ้าของบล็อกใส่ตัวแปรอาเรย์ขนาดเท่านี้ดู

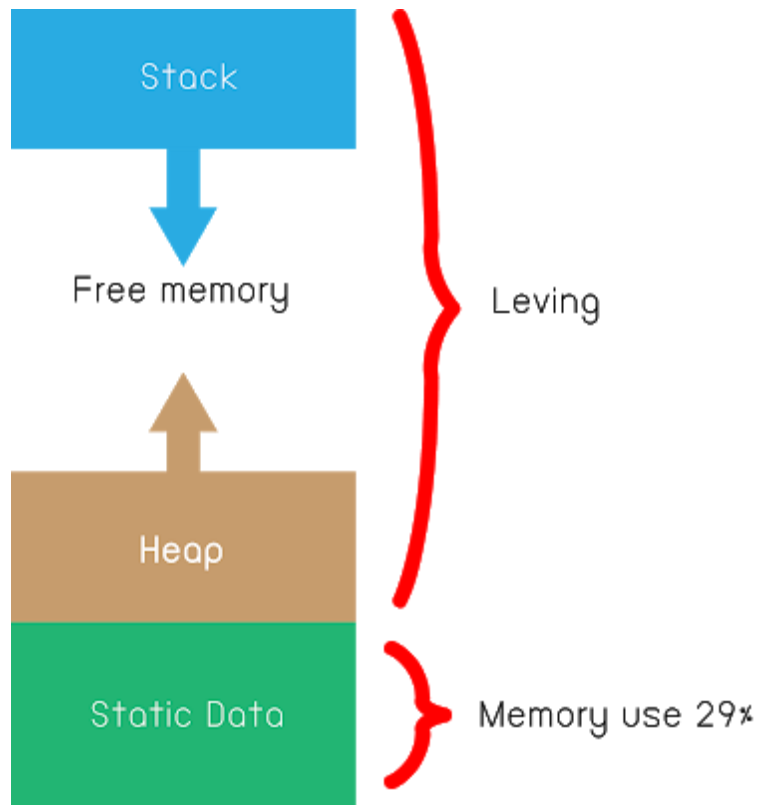
```
int data1[100];
float data2[100];
void setup(){
  data1[0] = 20;
  data2[0] = 10.0;
}

void loop(){
}
```

ผลลัพธ์ที่ได้ก็จะตามรูปข้างล่างนี้

```
Sketch uses 486 bytes (1%) of program storage space. Maximum is 32,256 bytes.
Global variables use 609 bytes (29%) of dynamic memory, leaving 1,439 bytes for local variables. Maximum is 2,048 bytes.
```

จะเห็นว่า memory ถูกใช้ไปแล้วตั้ง 29% อันนี้แค่ลองให้ดูเฉย ๆ นะ เพราะบางคนอาจจะบอกว่าคงไม่ได้เขียน array อะไรขนาดนี้หรอก แต่ก็อย่าลืมว่าบางงานเราก็ต้องใช้ library เหมือนกันซึ่งบางครั้งเราก็ไม่ได้เข้าไปดูหรอกว่า library เขาเขียนยังไง บางที memory ส่วนใหญ่ที่เสียไป อาจจะไม่ได้มาจากที่ผู้อ่านเอง แต่อาจจะมาจาก library ที่ผู้อ่านใช้อยู่ก็ได้



จากที่ลองเขียนไปทั้งหมดอยู่ในส่วนของ static data ทั้งหมด ซึ่งจะสังเกตว่า static data เป็นข้อมูลที่มีพื้นที่ตายตัว มีมากมีน้อยก็ขึ้นอยู่กับที่เราสร้าง และเราสามารถรับรู้ได้ด้วยว่าพื้นที่ใช้ไปเท่าไร และเหลือเท่าไร

ทำไม static data ต้องเป็นตัวแปรประเภท global variable และ static variable?

จริง ๆ แล้วสิ่งที่ตัวแปร 2 แบบนี้เหมือนกันคือ มันจะมีชีวิตตลอดการทำงานของโปรแกรม ก็หมายความว่ามันจะไม่ถูกทำลายขณะที่โปรแกรมทำงาน และการที่มันไม่ถูกทำลายไป นั่นแหละคือสาเหตุ มันก็เลยต้องการจองพื้นที่ตลอดการทำงาน

Heap

ถ้าแปลตามตัว heap ก็แปลว่า กอง เป็นข้อมูลที่ทับ ๆ กันสูงขึ้นเรื่อย ๆ อะไรประมาณนี้ จากที่เคยบอกไปในบทความที่แล้วว่าเราควรระวังการใช้งาน memory ในส่วนของ heap กัน เพราะว่าเราต้องเขียนจองพื้นที่เองและต้องคืนพื้นที่เอง

เราจะมาลองทดสอบกันดูหน่อย ซึ่งการทดสอบนี้เจ้าของบล็อกจะเช็ค memory ที่เหลือจากการใช้งาน ที่แรกเจ้าของบล็อกก็จะเขียนเอง แต่มีคนทำ library ไว้เช็ค free memory เหมือนกัน ดูจาก ที่นี้ นั่นเราก็มาร่วมกันเลยละกัน

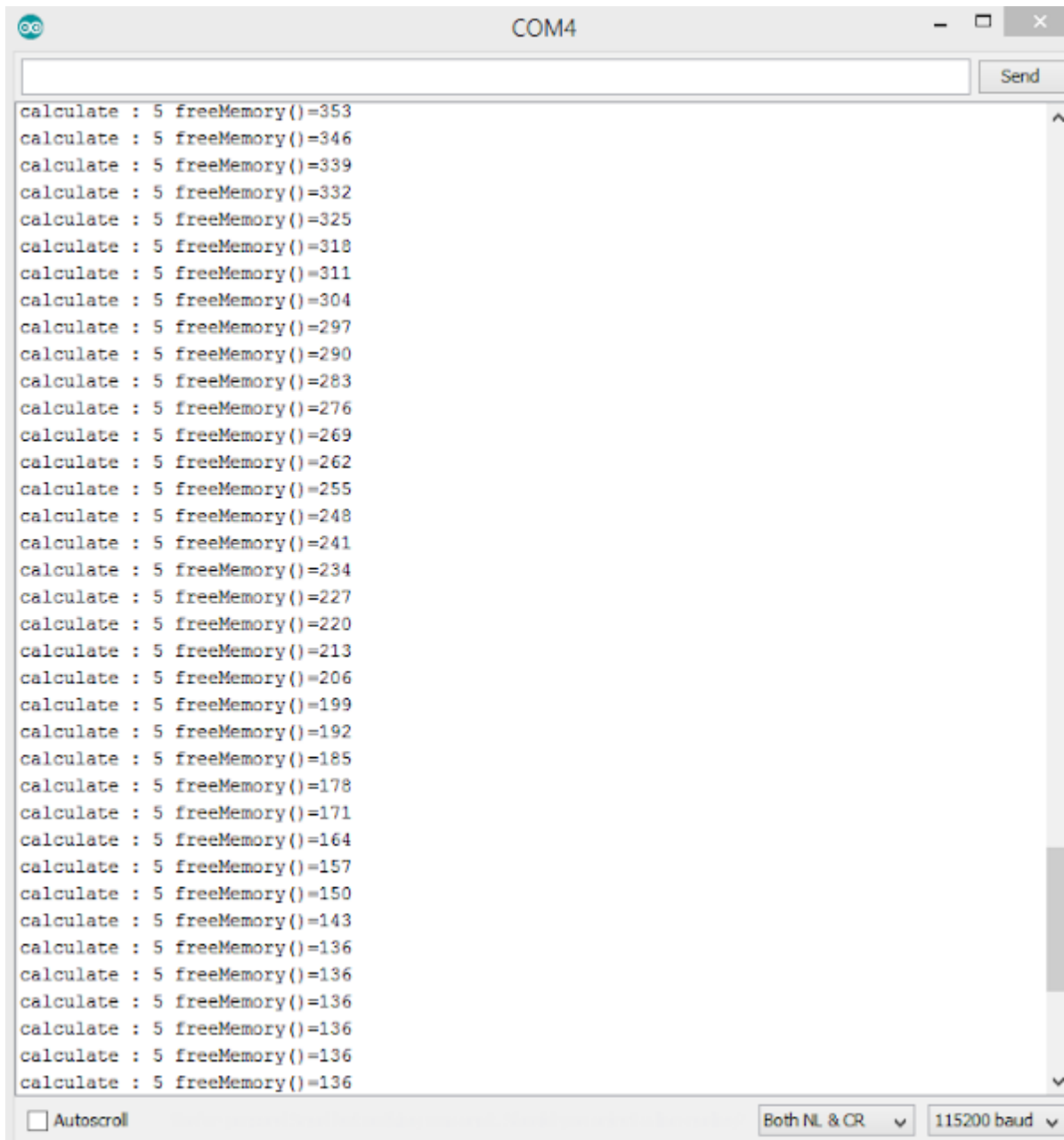
```
#include <MemoryFree.h>

int data1[100];

void setup(){
    Serial.begin(115200);
    data1[0] = 20;
}

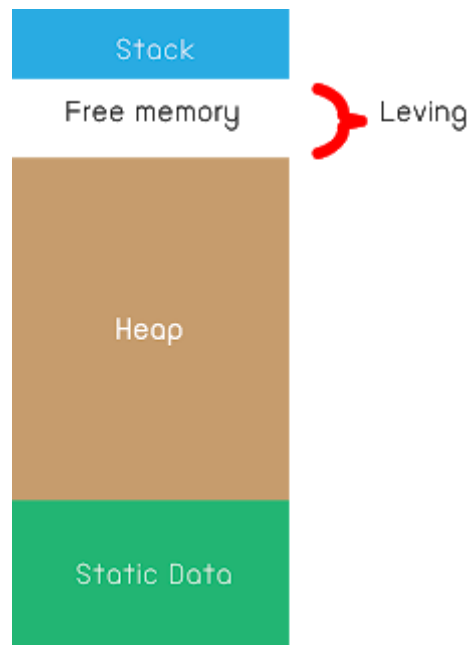
void loop(){
    int *data2 = (int*)malloc(5);
    *data2 = 10;
    Serial.print("calculate : ");
    Serial.print(*data2/2);
    Serial.print(" ");
    Serial.print("freeMemory()=");
    Serial.println(freeMemory());
    delay(500);
}
```

สิ่งที่เจ้าของบล็อกพยายามทำคือ จองพื้นที่ในหน่วยความจำ แต่ไม่มีการคืนค่ากลับไป ซึ่งผลลัพธ์ที่ได้ ผู้อ่านคิดว่าจะเป็นอย่างไรละ



```
calculate : 5 freeMemory()=353
calculate : 5 freeMemory()=346
calculate : 5 freeMemory()=339
calculate : 5 freeMemory()=332
calculate : 5 freeMemory()=325
calculate : 5 freeMemory()=318
calculate : 5 freeMemory()=311
calculate : 5 freeMemory()=304
calculate : 5 freeMemory()=297
calculate : 5 freeMemory()=290
calculate : 5 freeMemory()=283
calculate : 5 freeMemory()=276
calculate : 5 freeMemory()=269
calculate : 5 freeMemory()=262
calculate : 5 freeMemory()=255
calculate : 5 freeMemory()=248
calculate : 5 freeMemory()=241
calculate : 5 freeMemory()=234
calculate : 5 freeMemory()=227
calculate : 5 freeMemory()=220
calculate : 5 freeMemory()=213
calculate : 5 freeMemory()=206
calculate : 5 freeMemory()=199
calculate : 5 freeMemory()=192
calculate : 5 freeMemory()=185
calculate : 5 freeMemory()=178
calculate : 5 freeMemory()=171
calculate : 5 freeMemory()=164
calculate : 5 freeMemory()=157
calculate : 5 freeMemory()=150
calculate : 5 freeMemory()=143
calculate : 5 freeMemory()=136
calculate : 5 freeMemory()=136
calculate : 5 freeMemory()=136
calculate : 5 freeMemory()=136
calculate : 5 freeMemory()=136
```

ดูจาก Serial monitor แสดงให้เห็นว่า memory ถูกจองไปเรื่อย ๆ จนเหลือแค่ 136 byte เท่านั้น ซึ่งตรงนี้ไม่แน่ใจว่าทำไมถึงเหลือ 136 byte เพราะเจ้าของบล็อกก็กำลังจะดูอยู่เหมือนกันว่า ถ้ามันเหลือน้อยกว่านี้จะเป็นไง หรือว่าในส่วนของ heap มันใช้ได้แค่นี้



แต่นี้ก็เป็นผลลัพธ์ของการลืมนั่น memory นั้นแหละ และคราวนี้ลองใช้แล้วคืน memory ด้วย โดยเจ้าของบล็อกจะใส่ free เข้าไปแบบในโค้ด

