2019 국가암호공모전 Ⅱ 분야 답안 제출 양식

소속 : 서울시립대학교 **대표자 이름 : 방수민**

문제 03

답) Arduino-IDE 전체 프로젝트로 따로 제출하였으며, 최종적으로 아래 표에서 언급한 개선 사항을 적용한 코드를 사용하여 350ms가 걸렸다.

순번	개선 사항	속도 측정 시 적용한 사항	속도
1	Hint 들을 이용한 기본적인 코드	1	717ms
2	메인함수의 구조 변경	1,2	546ms
3	MIN 함수를 사용하지 않는 add 함수	1,2,3	ms
4-1	add 함수의 for문 삭제	1,2,3,4-1	408ms
4-2	mod 함수의 for문 삭제	1,2,3,4-1,4-2	354ms

< 표 1. 요약 표 >

개요 :

1. 문제 분석

$R = (A+B+C+D+E) \bmod 2^{56}-1$ 구현하기

Hint 1 : $2^{56} = 1 \pmod{2^{56}-1}$

Hint 2 : 54-비트 연산은 8-비트 프로세서에서 한 번에 수행할 수 없으므로 8-비트 단위로 쪼개어서 계산하도록 한다.

Hint 3 : 54-비트 덧셈 연산은 특정한 경우에 56-비트 결과 값을 도출한다.

2. 구현

보드 모델은 UNO R3를 사용하였으며 아두이노 1.85 에서 구현하였다. 속도를 측정한 컴퓨터는 window 환경이며, 프로세서 Intel(R) Core(TM) i5-2500 CPU, RAM 8.0GB, 64비트 운영 체제이다.

풀이 :

1. 기본적인 방법으로 짜기

1-1. add 짜기 (Hint 2 이용)

```
unsigned char add (unsigned char* R, unsigned char* IN1, unsigned char* IN2){
        //add 함수는 덧셈 코드를 구현함
        //함수 출력 값인 문자열 (R)은 7바이트임
        //함수 입력 값인 문자열 (IN1 그리고 IN2)은 모두 7바이트임
        //함수 리턴 값 (1바이트)은 덧셈에 대한 오버플로우 값 저장용으로 활용
}
```

< 그림 1. 문제에서 주어진 add 조건 >

Hint 2에 따르면 54-비트 연산은 8-비트 프로세서에서 한 번에 수행할 수 없으므로 8-비트 단위로 쪼개어서 계산하도록 한다. 이를 이용하여 다음 장의 그림처럼 코드를 구현해보았다.

```
unsigned char add(unsigned char* R, unsigned char* IN1, unsigned char* IN2)
{
  unsigned char CARRY = 0;


  for (int i = 6;i >= 0;i--)
  {
    R[i] = IN1[i] + IN2[i] + CARRY;
    if (R[i] <= MIN(IN1[i], IN2[i]))
      CARRY = 1;
    else
      CARRY = 0;
  }

  return CARRY;
}
```

< 그림 2. 1-1 add 함수 >

이 때 IN1, IN2는 배열의 앞부분부터 숫자의 상위비트이다. 그리고 MIN 함수는 아래와 같다.

```
#define MIN(a, b) (((a) < (b)) ? (a) : (b))
```

< 그림 3. MIN 함수 >

if문에서 그냥 두 숫자 중 아무 숫자나 사용하지 않고 MIN 함수를 사용하는 데에는 2가지 이유가 있다. 이해하기 쉽도록 10진수로 설명하도록 하겠다.

① 9+9+(carry=1) 인 case

이때는 결과 값이 9가 나오므로 두 숫자 중 아무 숫자를 사용했을 시에 등호를 붙여줘야 한다.

② 0+5 인 case

이 때는 결과 값이 5가 나오므로 두 숫자 중 아무 숫자를 사용했을 시에 등호를 붙이 안 된다.

따라서 두 숫자 중 아무 숫자를 사용했을 시 이 두 case가 충돌하므로 MIN 함수를 사용하였다.

1-2. mod 짜기 (Hint 1 이용)

```
void mod (unsigned char* OUT, unsigned char* IN, unsigned char CARRY){
        //mod 함수는 모듈러 리덕션 코드를 구현함
        //함수 출력 값인 문자열 (OUT)은 7바이트임
        //함수 입력 값 중 문자열 (IN)은 7바이트임
        //함수 입력 값 중 문자 (CARRY)는 1바이트임
}
```

< 그림 4. 문제에서 주어진 mod 함수 조건 >

Hint 1에 따르면 $2^{56} = 1 \pmod{2^{56}-1}$이다. 이를 이용하여 다음 장의 그림처럼 코드를 구현해보았다.

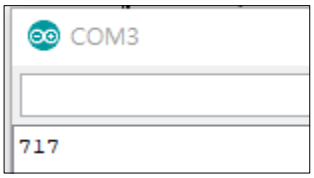
```
void mod(unsigned char* OUT, unsigned char* IN, unsigned char CARRY)
{
  int i;

  if (CARRY)
  {
    OUT[6] = IN[6] + 1;
    int i=5;
    while (i)
    {
      if (OUT[i])
        break;
      else
        OUT[i - 1] = IN[i - 1] + 1;
      i--;
    }
    for (i; i >= 0; i--)
    {
      OUT[i] = IN[i];
    }
  }
  else
  {
    for (i = 0;i <= 6;i++)
    {
      OUT[i] = IN[i];
    }
  }
}
```

<그림 5. 1-2 mod 함수 >

힌트에 의해 캐리가 있다면 그 CARRY는 $2^{56} = 1$이므로 그냥 IN에 1을 더해주는 것과 같다.

1-3. 연산속도 체크

1-1과 1-2, 그리고 문제에서 주어진 메인함수로 구현한 코드의 시간을 측정해보았다.



< 그림 6. 1 연산속도 체크 >

717ms가 걸렸다.

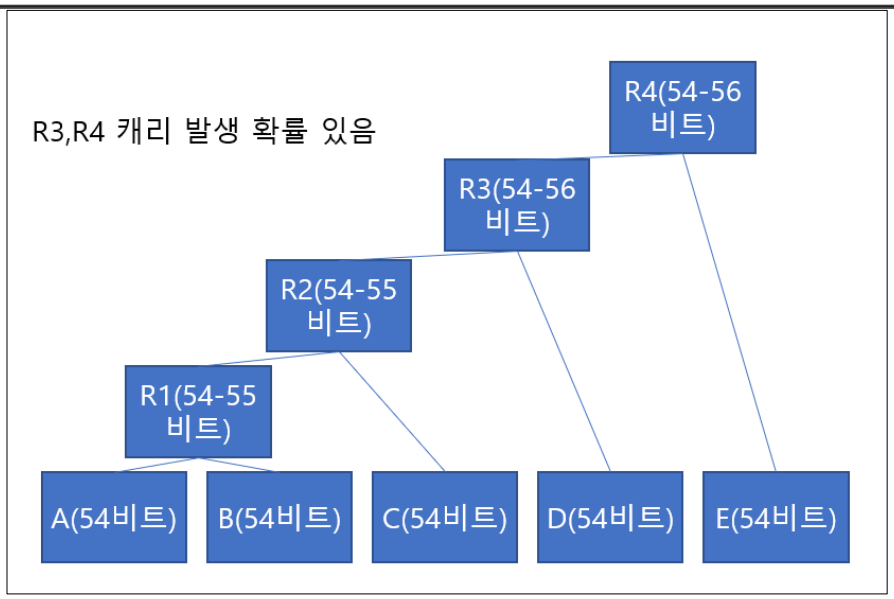
2. 메인함수 수정

2-1. 메인함수 수정의 이론

```
CARRY=add (R1,A,B);
mod (R1,R1,CARRY);
CARRY=add (R2,R1,C);
mod (R2,R2,CARRY);
CARRY=add (R3,R2,D);
mod (R3,R3,CARRY);
CARRY=add (R4,R3,E);
mod (R,R4,CARRY);
```

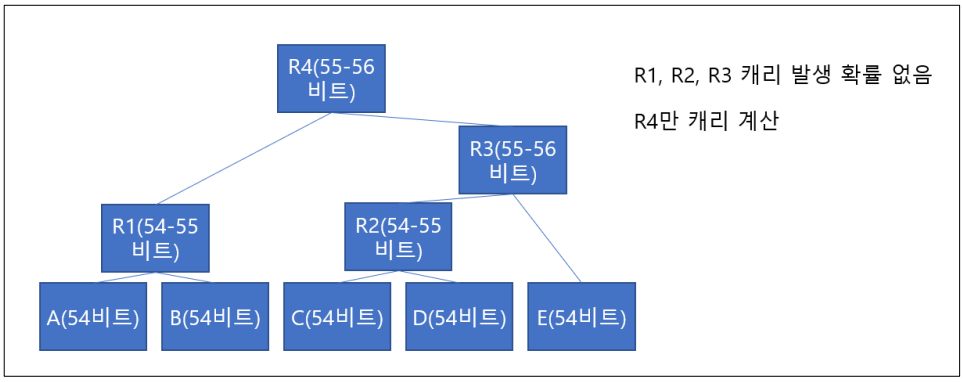
< 그림 7. 문제에서 주어진 메인함수 >

문제에서 주어진 메인함수는 위와 같고 그 구조를 그림으로 그려보면 아래의 그림과 같다.



< 그림 8. 문제에서 주어진 메인함수 구조 >

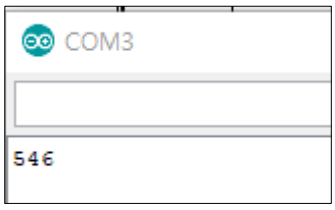
이 때, R3, R4에서만 CARRY가 발생할 가능성이 있으므로 mod를 4번 하는 것은 비효율적이다.



< 그림 9. 수정한 메인함수 구조 >

구조를 다음과 같이 수정하면, R4에서만 CARRY가 발생하므로 mod를 한 번만 사용할 수 있다. 54비트 5개를 더할 때, mod를 한 번도 사용하지 않을 수 없으므로 1번 사용하는 것이 최소이다.

2-2. 메인함수 수정 후의 연산속도 체크 결과



< 그림 10. 2 연산속도 체크 결과 >

546ms가 걸렸다.

3. add 함수의 변화

3-1. add 함수의 변화

1-1에서의 add 함수는 MIN 함수를 호출해야하므로 비효율적이다. 따라서 MIN 함수를 호출하지 않는 방법을 생각해보았다.

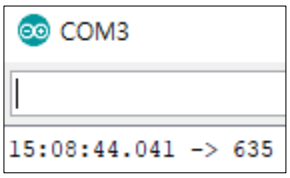
```
unsigned char add(unsigned char* R, unsigned char* IN1, unsigned char* IN2)
{
  unsigned char CARRY = 0;

  R[6] = IN1[6] + IN2[6];
  if (R[6] < IN1[6])
    CARRY = 1;

  for (int i = 5;i >= 0;i--)
  {
    if (CARRY)
    {
      R[i] = IN1[i] + IN2[i] + 1;
      if (R[i] <= IN1[i])
        CARRY = 1;
      else
        CARRY = 0;
    }
    else
    {
      R[i] = IN1[i] + IN2[i];
      if (R[i] < IN1[i])
        CARRY = 1;
      else
        CARRY = 0;
    }
  }

  return CARRY;
}
```

< 그림 11. 수정한 add 함수 >

이러한 방식으로 case를 나누면 MIN 함수를 사용하지 않아도 된다.

3-2. add 함수의 변화 후의 연산속도 체크 결과

2번의 코드에서 add 함수만 변화하였다.



< 그림 12. 3 연산속도 체크 >

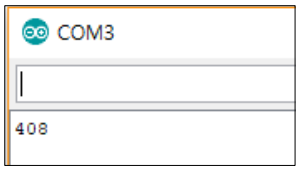
635ms가 걸렸다.

4. 간단하게 속도 줄이기

4-1. add 함수 for문 삭제

3-1의 add 함수에서 for문을 풀어썼다.

4-2. add 함수 for문 삭제 후의 연산속도 체크 결과



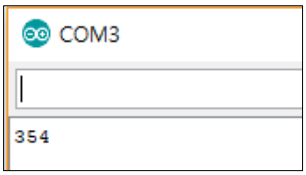
< 그림 13. 4-1 연산속도 체크 >

408ms가 걸렸다.

4-3 mod 함수 for문 삭제

1-2의 mod 함수에서 for문을 풀어썼다.

4-4. mod 함수 for문 삭제후의 연산속도 체크 결과



< 그림 14. 4-2 연산속도 체크 >

354ms가 걸렸다.