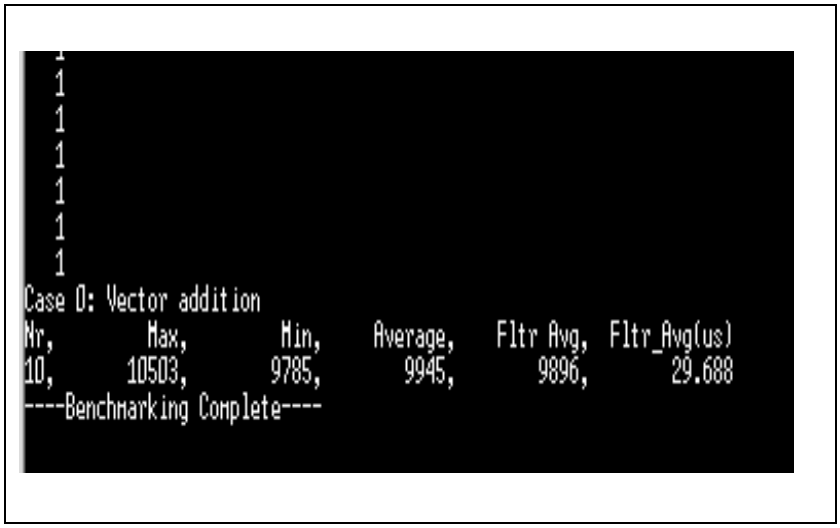
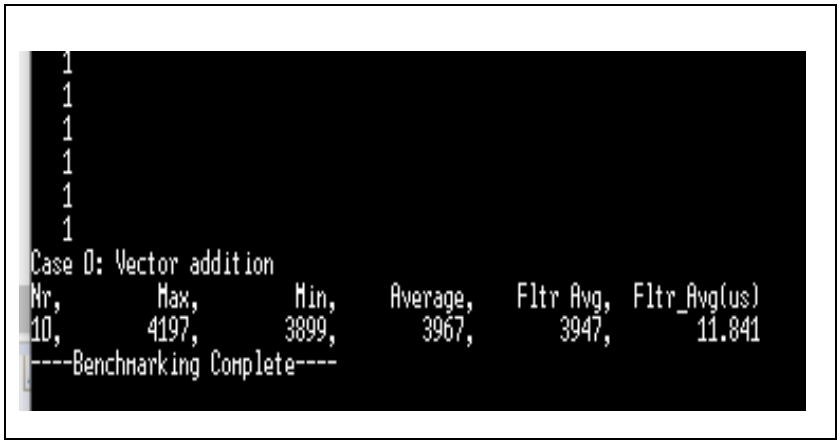
201810528 고려욱 Lab3 과제

1. (p8) main.c.1 base

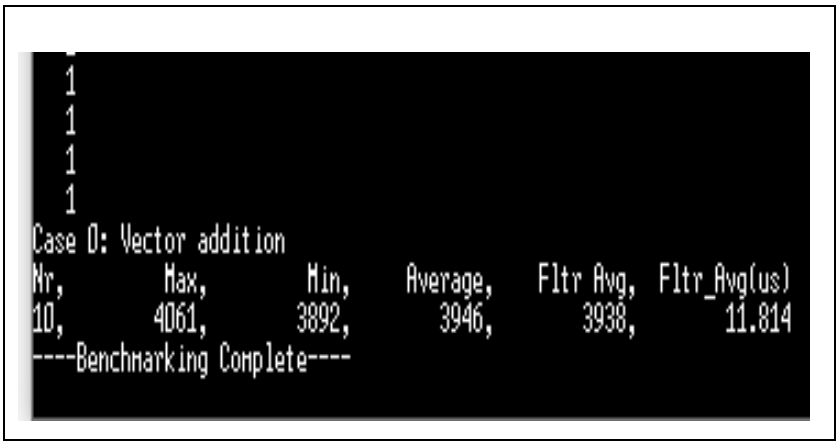


2. (p14) main.c.1 NEON



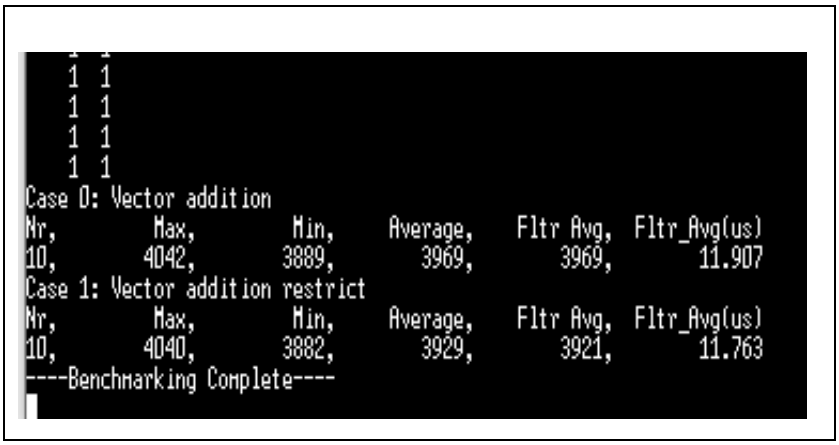
P8과의 차이는 Optimization을 O3 level로 변경하고 NEON Chip을 사용하여 Vector Addition을 진행한 결과이다. O3 level의 경우에는 NEON code에 대해서 자동으로 vectorize 연산을 진행시켜 주고 공격적인 최적화를 가능하게 하기 때문에 O0인 zero optimzation보다 시간이 빨리 지게 된다. 특히, iteration횟수를 따로 계속 메모리에서 가져오는 것이 아닌 point regsister에서 가져오기 때문에 속도 향상이 일어난다. 또한, vector 연산이 32bit data를 d15, d16에 4개씩 올려 한꺼번에 현산하기 때문에 P8과 비교해서 iteration 당 pb+x를 하나씩 진행하여 pa에 저장한 것과 달리 한 iteration 당 4개의 연산을 진행하기 때문에 연산에 걸리는 시간이 줄어들게 된다. 그러나, assembly code 상에서는 1개씩 계산되는 예외처리 부분으로 branch될 수 없다. 그 이유는, iteration의 횟수를 의미하는 i를 4의 배수로 강제화 시켜주지 않았기 때문에 4개 이하로 남은 것을 더할 때 필요 외의 횟수가 더 진행되게 되어 완전히 4배로 빨라 지지 않는다.

3. (p16) main.c.1 NEON, multiple of 4



P14와의 차이에는 multiple of 4에 있다. bic명령어를 통해 iteration의 횟수를 결정하는 i와 ~3을 and연산을 취하여 iteration의 횟수를 4의 배수로 강제화 시켜주게 된다. P14와 비교하였을 때, 각 iteration의 횟수를 미리 앞에서 지정해 두어, iteration의 횟수를 비교하는데 걸리는 시간이 줄어 앞의 시간에 비해 소폭 감소한 결과가 보인다고 생각된다. (SBFX??)

4. (p16) main.c.2 NEON, multiple of 4, restrict



앞서 main.c.1과의 차이는 restrict를 각 배열의 시작 pa, pb에 지정한 것이다. Restrict 포인터는 각 포인터가 서로 다른 메모리 공간을 가리키고 있고, 다른 곳에서 접근하지 않으니 컴파일러가 최적화를 하라는 뜻이다. 따라서, 기존 main.c.1에서와 같이 restrict가 없는 경우 pa, pb가 계속해서 증가되는 메모리에 접근하여 검사하는 과정이 감소되었다고 생각된다. Assembly code만을 보더라도 loop-carride dependecy를 점검하기 위해 메모리에 접근하는 코드가 감소하였으며 따라서, 속도의 조금의 차이는 메모리에 접근하는 것을 컴파일러가 최대한 최적화함에 따라 달라졌다고 생각한다.

5. (p.27) main.c.3 NEON, multiple of 4, restrict

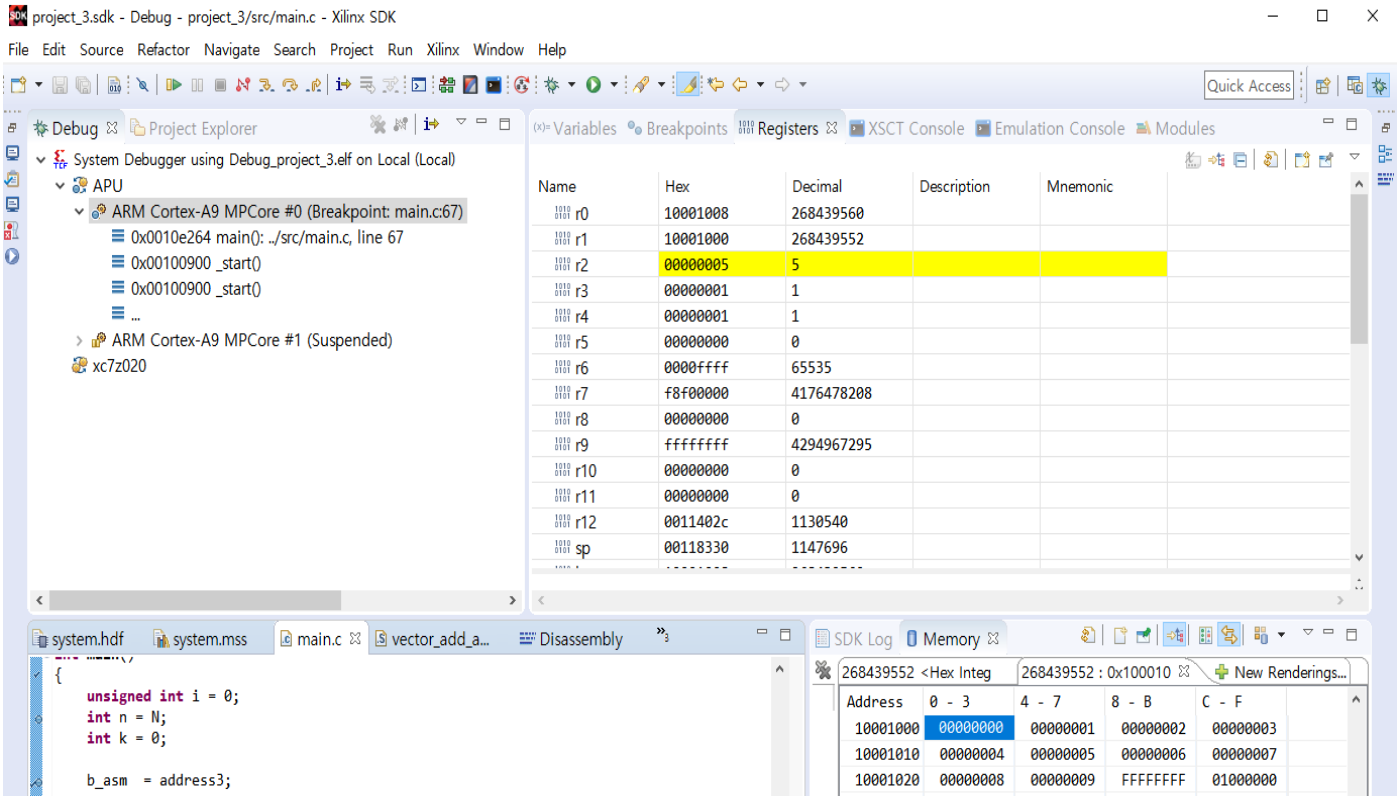
먼저, 앞서와 비교하여서는 assembly code로 직접 구현하는 것이 앞서와의 차이였다. 여러 번 돌려보아도 강의자료와 같이 assembly code가 더 빠르게 동작하지 않았다. 아마도, assembly code를 구현하는데 있어 느리게 동작하는 요인이 있었을 것이라고 생각한다. 구현은 NEON을 이용한 vector addition으로 구현하였으나 오히려 vector addition을 적용해 주지 않은 Case0와 비슷한 속도를 얻게 되었다.

6. (p.30) main.c.3 NEON, multiple of 4, restrict, overlapping

앞서 결과들과의 차이로는 overlapping을 했다는 것에 있다. 해당 기능은 이전의 메모리 영역을 계속해서 참조하는 것을 말한다. 따라서, 이전에 불러 들어왔던 값에 계속해서 1이 더해졌기 때문에 add_int의 경우에 하나씩 더해져 1이 되었던 것이다. 하지만, add_int_restrict의 경우에는 4개씩 restrict를 통해 컴파일러에게 메모리공간이 다르다고 인식시켜 4개의 메모리를 올릴 때 처음 메모리에만 접근하여 기존의 값을 모두 불러오는 최적화작업이 진행되어서 결과가 다르게 나왔다고 생각한다. 마지막으로, overlapping 시에 가장 시간이 NEON을 쓰지 않은 것이 제일 적게 걸린 이유로는 restirct를 쓰지 않았음에 있었다고 생각한다. Restrict는 compiler가 array가 overlap되었는지를 체크하는 시간을 줄여주는데 명시적으로 overlap되었다고 코드상에서 작성하였으므로, overlap을 체크하는데 따로 시간이 소요되지 않았으며, 해당 메모리주소가 cache에 바로 직전에 올라가 있어 해당 값에 접근하는데 가장 적게 걸렸다고 생각된다. 이후의 두 코드는 모두 restrict를 사용하였으므로 사용자가 원하는 결과를 불러일으키지 못하였고 cache상에서도 메모리 주소를 바로 읽어올 수 없어 시간이 오래 걸렸다고 생각된다.

7. (p.31) modify C application

<실행 전>



<실행 후>