[Microprocessor Applications]
Lab 3: NEON Programming

Chester Sungchung Park (박성정)

SoC Design Lab, Konkuk University

Webpage: http://soclab.konkuk.ac.kr

Outline

- Creating C Applications
- Running C applications
- Debugging C Applications
- Optimizing C applications
- Programming assembly codes

Creating C Applications

❑ Repeat the previous steps

- Follow pp. 4~7 of the following lab workbook: **Lab_MP2022_2_work.pdf**
- Add the source files by those attached below.
    - ✓ main.c.1, benchmarking.c, benchmarking.h

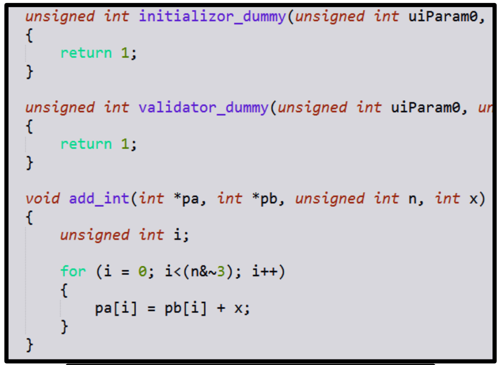
main.c.1 benchmarking.c benchmarking.h

Running C Applications

❑ Check the source files

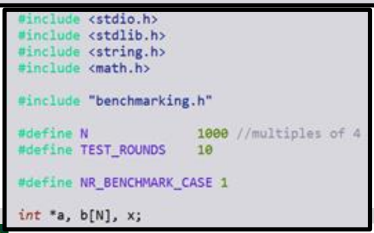
- Expand <***your project name>*** to see all of the source files that are part of this project by clicking the ***‘src’*** icon.
- Double-click the ***<file names>*** to open them.

Function



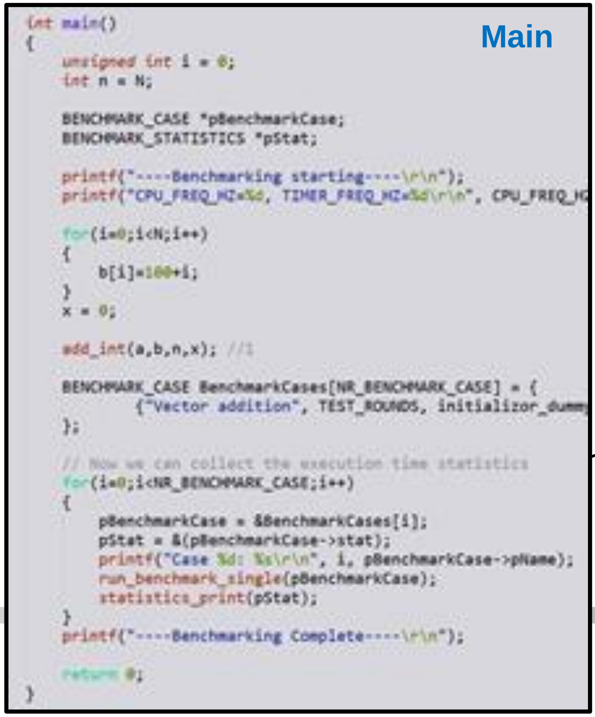
```c
unsigned int initializor_dummy(unsigned int uiParam0,
{
    return 1;
}

unsigned int validator_dummy(unsigned int uiParam0, un
{
    return 1;
}

void add_int(int *pa, int *pb, unsigned int n, int x)
{
    unsigned int i;

    for (i = 0; i<(n&~3); i++)
    {
        pa[i] = pb[i] + x;
    }
}
```

Header files and global variables

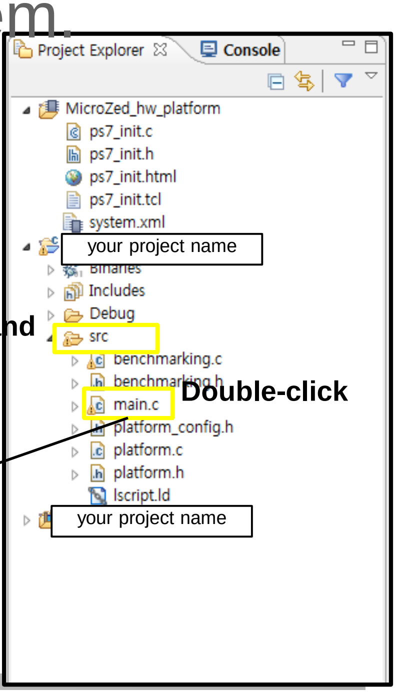


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "benchmarking.h"

#define N               1000 //multiples of 4
#define TEST_ROUNDS     10

#define NR_BENCHMARK_CASE 1

int *a, b[N], x;
```

Main



```c
int main()
{
    unsigned int i = 0;
    int n = N;

    BENCHMARK_CASE *pBenchmarkCase;
    BENCHMARK_STATISTICS *pStat;

    printf("----Benchmarking starting----\r\n");
    printf("CPU_FREQ_HZ=%d, TIMER_FREQ_HZ=%d\r\n", CPU_FREQ_H

    for(i=0;i<N;i++)
    {
        b[i]=100+i;
    }
    x = 0;

    add_int(a,b,n,x); //1

    BENCHMARK_CASE BenchmarkCases[NR_BENCHMARK_CASE] = {
            {"Vector addition", TEST_ROUNDS, initializor_dumm
    };

    // Now we can collect the execution time statistics
    for(i=0;i<NR_BENCHMARK_CASE;i++)
    {
        pBenchmarkCase = &BenchmarkCases[i];
        pStat = &(pBenchmarkCase->stat);
        printf("Case %d: %s\r\n", i, pBenchmarkCase->pName);
        run_benchmark_single(pBenchmarkCase);
        statistics_print(pStat);
    }
    printf("----Benchmarking Complete----\r\n");

    return 0;
}
```



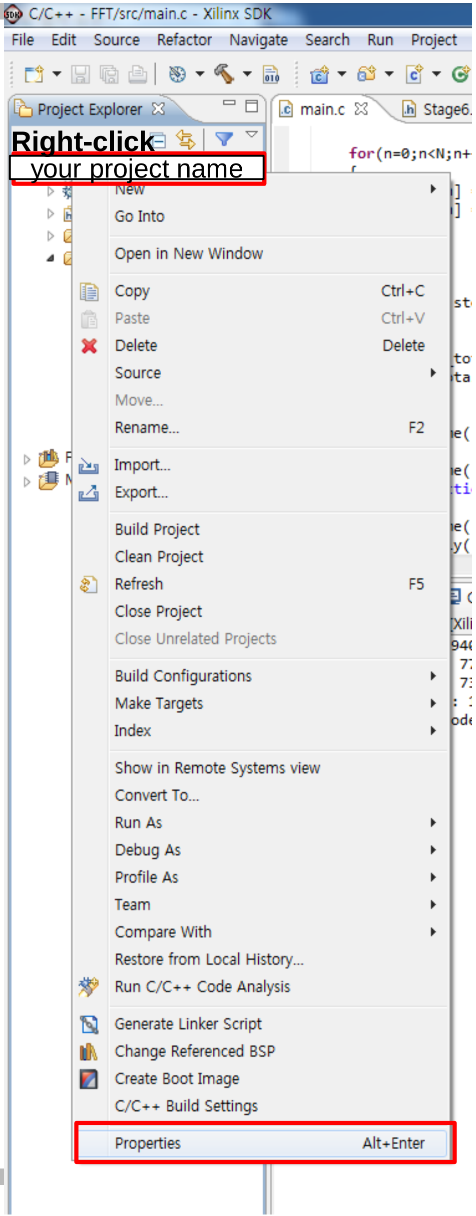
```
MicroZed_hw_platform
    ps7_init.c
    ps7_init.h
    ps7_init.html
    ps7_init.tcl
    system.xml
your project name
    Binaries
    Includes
    Debug
    src
        benchmarking.c
        benchmarking.h
        main.c
        platform_config.h
        platform.c
        platform.h
        lscript.ld
your project name
```

Expand

Double-click

Running C Applications

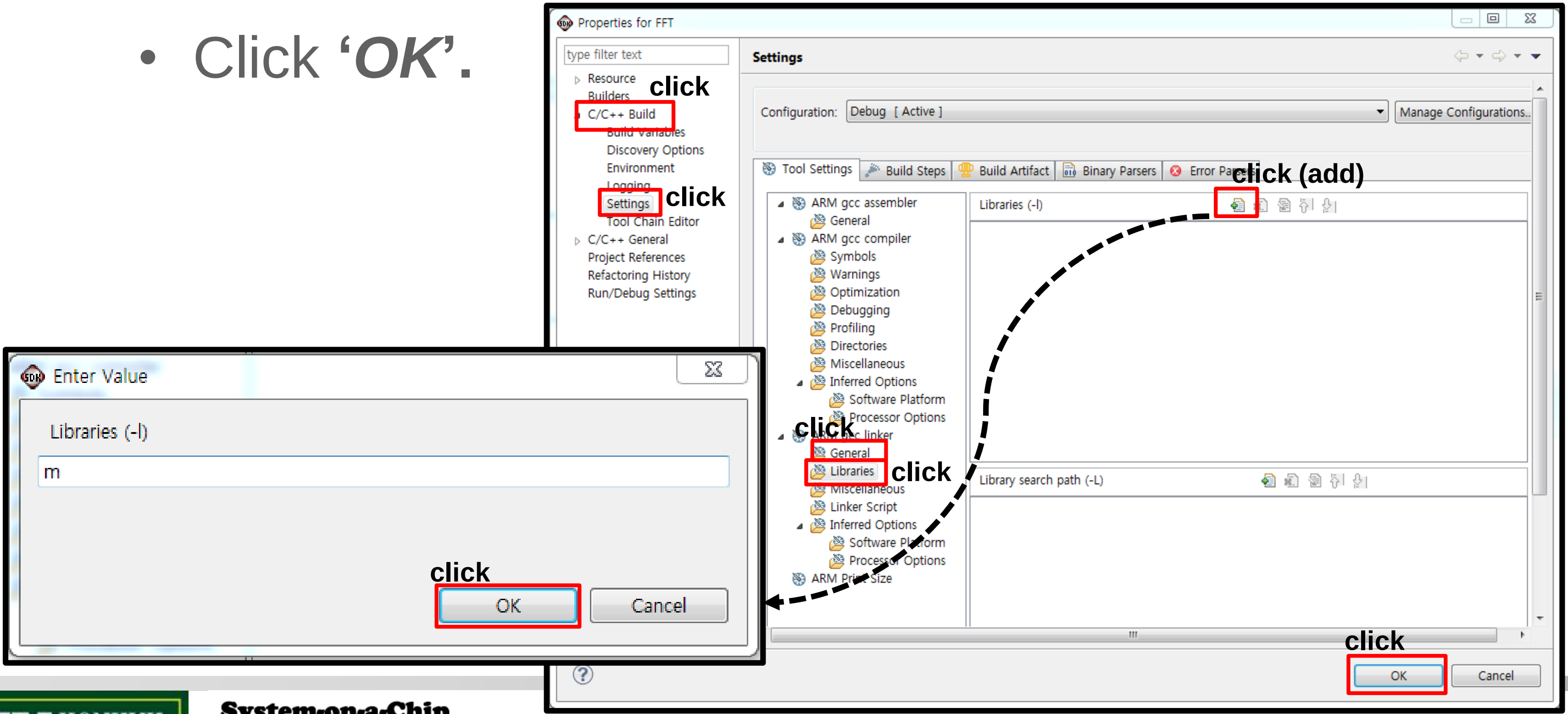
- ❑ Set up the ***'–lm library'***
  - Project must have ***'–lm library'*** to use ***'math.h'*** header file.
  - Right-click ***'your project name'*** in ***'Project Explorer' > 'Properties'***



Right-click your project name

Running C Applications

- Set up the ***'–lm library'*** (cont'd)
  - Click ***'C/C++ Build'*** > ***'Settings'*** > ***'ARM gcc linker'*** > ***'libraries'*** > ***'add'***
  - Add the value ***'m'***
  - Click ***'OK'***.



Properties for FFT

Settings

Configuration: Debug [Active]

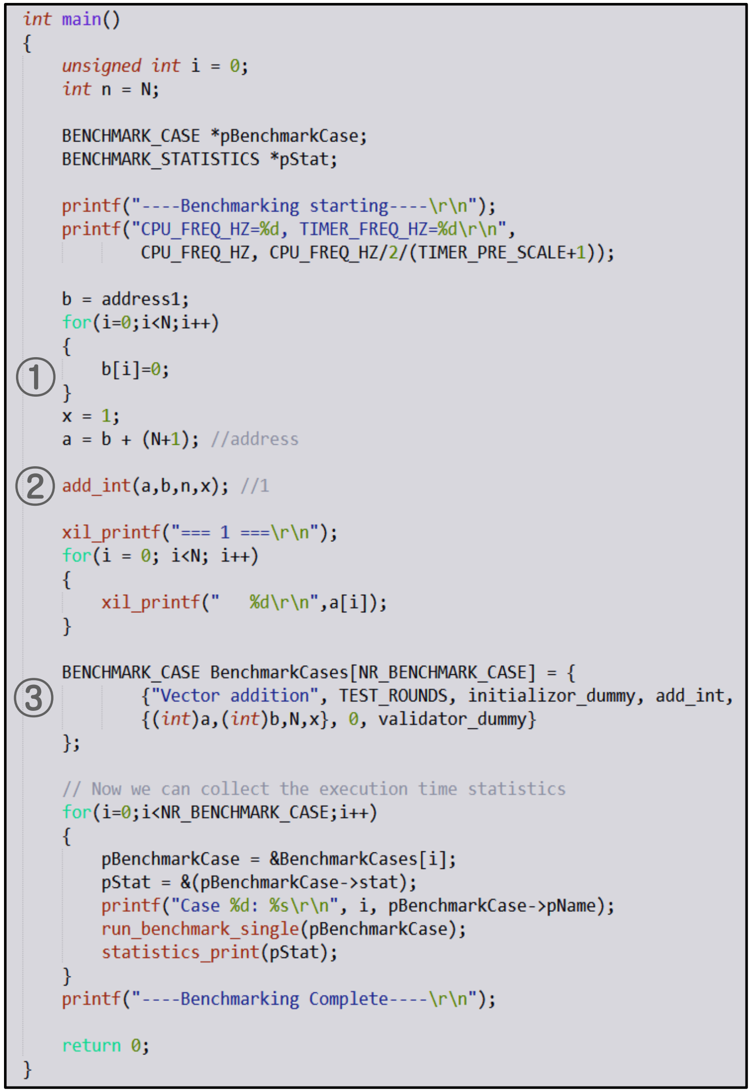
Enter Value

Libraries (-l): m

Running C Applications

- ❑ Review the source code: ***‘main.c’***
  - ① Input a sequence
  - ② Call ‘***add_int()***’
  - ③ Measure execution time

- ❑ Review the remaining source codes: ***‘benchmarking.h’*** & ***‘benchmarking.c’***

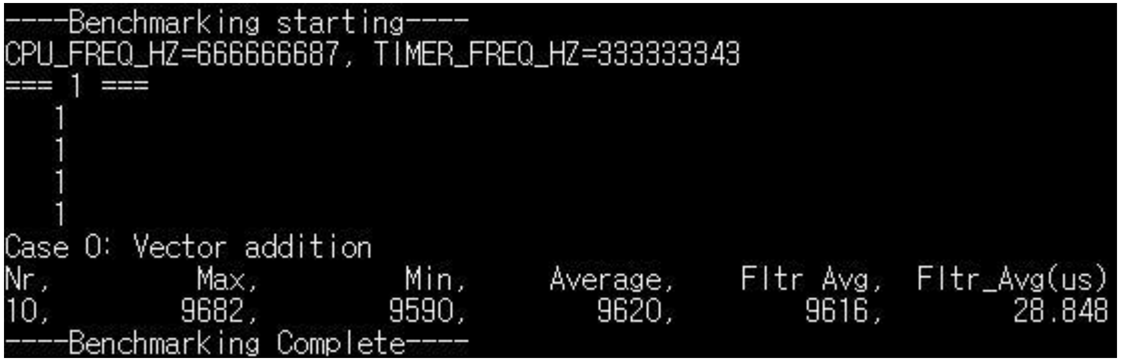


```c
int main()
{
    unsigned int i = 0;
    int n = N;

    BENCHMARK_CASE *pBenchmarkCase;
    BENCHMARK_STATISTICS *pStat;

    printf("----Benchmarking starting----\r\n");
    printf("CPU_FREQ_HZ=%d, TIMER_FREQ_HZ=%d\r\n",
            CPU_FREQ_HZ, CPU_FREQ_HZ/2/(TIMER_PRE_SCALE+1));

    b = address1;
    for(i=0;i<N;i++)
    {
        b[i]=0;
    }
    x = 1;
    a = b + (N+1); //address

    add_int(a,b,n,x); //1

    xil_printf("=== 1 ===\r\n");
    for(i = 0; i<N; i++)
    {
        xil_printf("   %d\r\n",a[i]);
    }

    BENCHMARK_CASE BenchmarkCases[NR_BENCHMARK_CASE] = {
            {"Vector addition", TEST_ROUNDS, initializor_dummy, add_int,
            {(int)a,(int)b,N,x}, 0, validator_dummy}
    };

    // Now we can collect the execution time statistics
    for(i=0;i<NR_BENCHMARK_CASE;i++)
    {
        pBenchmarkCase = &BenchmarkCases[i];
        pStat = &(pBenchmarkCase->stat);
        printf("Case %d: %s\r\n", i, pBenchmarkCase->pName);
        run_benchmark_single(pBenchmarkCase);
        statistics_print(pStat);
    }
    printf("----Benchmarking Complete----\r\n");

    return 0;
}
```

Running C Applications

❑ Repeat the previous steps

- Follow pp. 30~34 of the following lab workbook: **Lab_MP2022_1_work.pdf**
- Check the output on ***‘Tera Term’***
  - ✓ Measure the execution time



```
----Benchmarking starting----
CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343
=== 1 ===
   1
   1
   1
   1
Case 0: Vector addition
Nr,         Max,         Min,     Average,    Fltr Avg,  Fltr_Avg(us)
10,        9682,        9590,        9620,        9616,        28.848
----Benchmarking Complete----
```

- Nr: Function execution count.
- Max: The longest time in the function execution count. (unit: cycles)
- Min: The shortest time in the function execution count. (unit: cycles)
- Average: Average time except Max and Min. (unit: cycles)
- Fltr_Avg: Average / TIMER_FREQ_HZ (unit: usecs)

Debugging C Applications

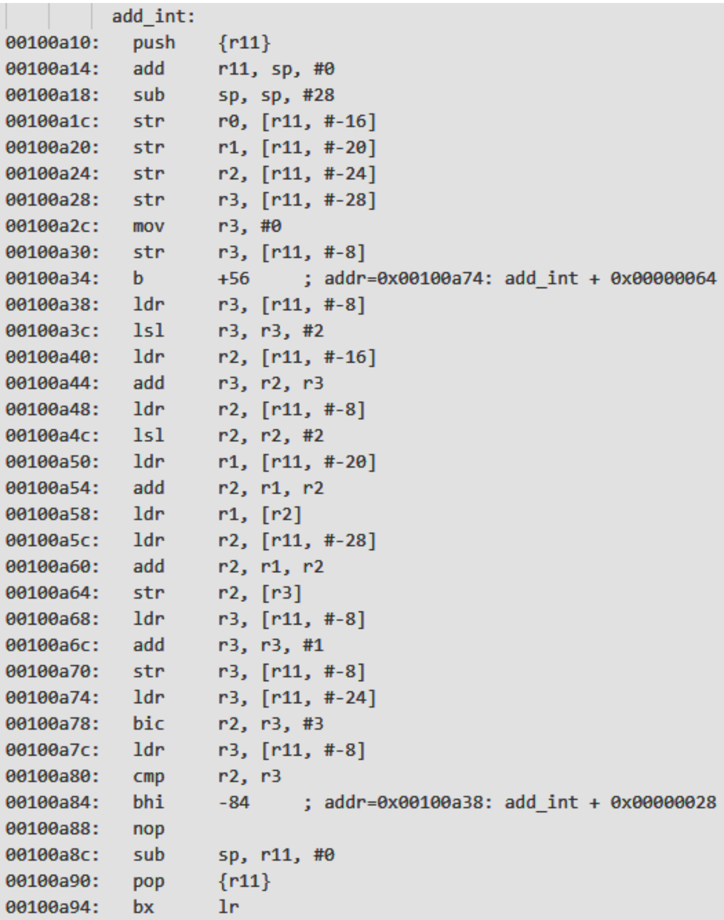
❑ Repeat the previous steps

- Follow pp. 14~19 of the following lab workbook: **Lab_MP2022_2_work_r1.pdf**

Debugging C Applications

❑ Review the disassembly

- Check how efficiently the assembly code runs
- Figure out how to **speed up** the assembly code



```
                add_int:
00100a10:   push    {r11}
00100a14:   add     r11, sp, #0
00100a18:   sub     sp, sp, #28
00100a1c:   str     r0, [r11, #-16]
00100a20:   str     r1, [r11, #-20]
00100a24:   str     r2, [r11, #-24]
00100a28:   str     r3, [r11, #-28]
00100a2c:   mov     r3, #0
00100a30:   str     r3, [r11, #-8]
00100a34:   b       +56      ; addr=0x00100a74: add_int + 0x00000064
00100a38:   ldr     r3, [r11, #-8]
00100a3c:   lsl     r3, r3, #2
00100a40:   ldr     r2, [r11, #-16]
00100a44:   add     r3, r2, r3
00100a48:   ldr     r2, [r11, #-8]
00100a4c:   lsl     r2, r2, #2
00100a50:   ldr     r1, [r11, #-20]
00100a54:   add     r2, r1, r2
00100a58:   ldr     r1, [r2]
00100a5c:   ldr     r2, [r11, #-28]
00100a60:   add     r2, r1, r2
00100a64:   str     r2, [r3]
00100a68:   ldr     r3, [r11, #-8]
00100a6c:   add     r3, r3, #1
00100a70:   str     r3, [r11, #-8]
00100a74:   ldr     r3, [r11, #-24]
00100a78:   bic     r2, r3, #3
00100a7c:   ldr     r3, [r11, #-8]
00100a80:   cmp     r2, r3
00100a84:   bhi     -84      ; addr=0x00100a38: add_int + 0x00000028
00100a88:   nop
00100a8c:   sub     sp, r11, #0
00100a90:   pop     {r11}
00100a94:   bx      lr
```

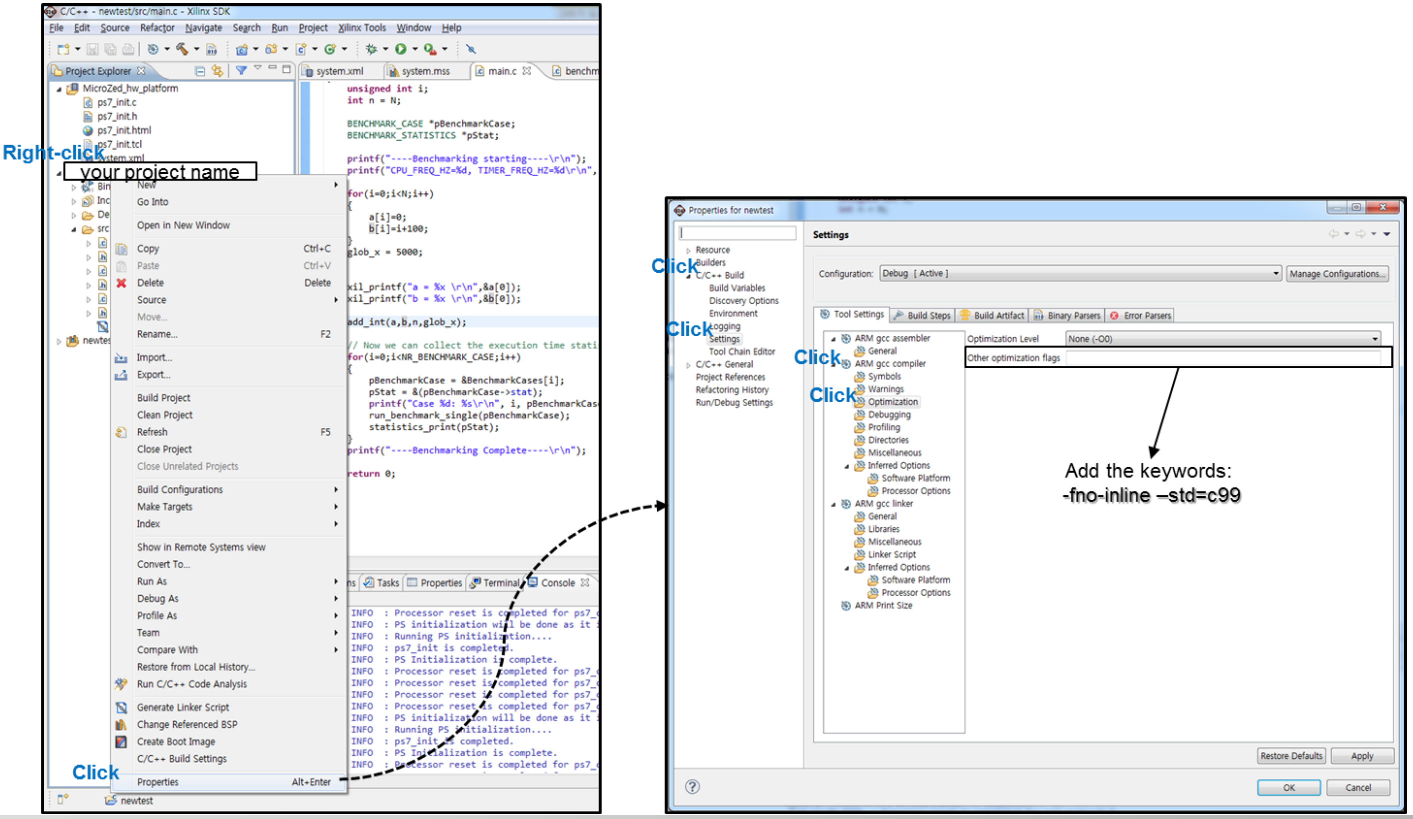
Optimizing C Applications

❑ Set the compiler optimization level to **-O3**

- -O0: No optimization is performed.
- -O1: Enables the most common forms of optimization that do not require decisions regarding size or speed.
- -O2: Enables further optimizations, such as instruction scheduling.
- -O3: Enables more aggressive optimizations, such as aggressive function inlining, and it typically increases speed at the expense of image size. Moreover, this option enables -ftree-vectorize, causing the compiler to attempt to automatically generate NEON code.
- -Os: Selects optimizations that attempt to minimize the size of the image, even at the expense of speed.

Optimizing C Applications

- ❑ Repeat the previous steps
  - Follow p. 23 of the following lab workbook: **Lab_MP2022_2_work_r1.pdf**



```
unsigned int i;
int n = N;

BENCHMARK_CASE *pBenchmarkCase;
BENCHMARK_STATISTICS *pStat;

printf("----Benchmarking starting----\r\n");
printf("CPU_FREQ_HZ=%d, TIMER_FREQ_HZ=%d\r\n",

for(i=0;i<N;i++)
{
    a[i]=0;
    b[i]=i+100;
}
glob_x = 5000;


xil_printf("a = %x \r\n",&a[0]);
xil_printf("b = %x \r\n",&b[0]);

add_int(a,b,n,glob_x);

// Now we can collect the execution time stati
for(i=0;i<NR_BENCHMARK_CASE;i++)
{
    pBenchmarkCase = &BenchmarkCases[i];
    pStat = &(pBenchmarkCase->stat);
    printf("Case %d: %s\r\n", i, pBenchmarkCas
    run_benchmark_single(pBenchmarkCase);
    statistics_print(pStat);
}
printf("----Benchmarking Complete----\r\n");

return 0;
```

```
INFO  : Processor reset is completed for ps7_
INFO  : PS initialization will be done as it i
INFO  : Running PS initialization....
INFO  : ps7_init is completed.
INFO  : PS Initialization is complete.
INFO  : Processor reset is completed for ps7_
INFO  : Processor reset is completed for ps7_
INFO  : Processor reset is completed for ps7_
INFO  : Processor reset is completed for ps7_
INFO  : PS initialization will be done as it i
INFO  : Running PS initialization....
INFO  : ps7_init is completed.
INFO  : PS Initialization is complete.
INFO  : Processor reset is completed for ps7_
```

Configuration: Debug [Active]

Optimization Level: None (-O0)

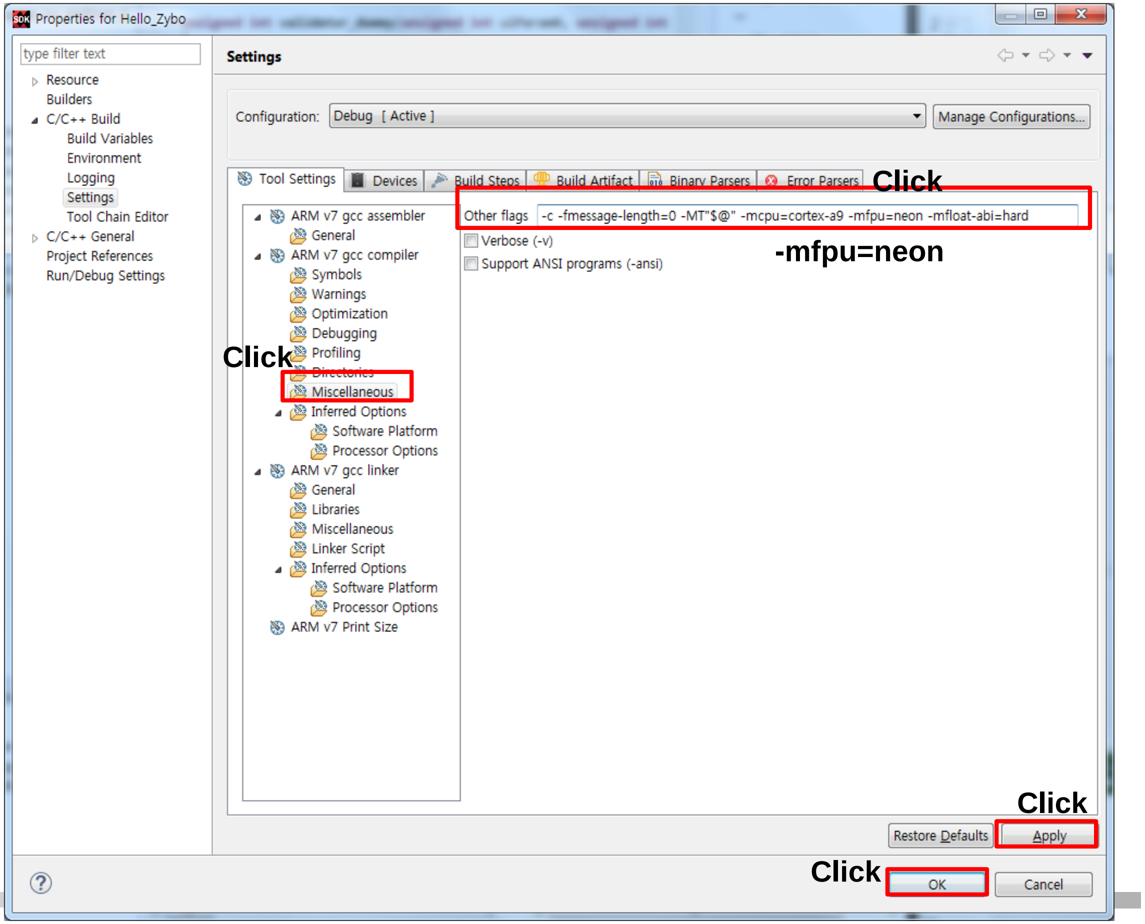
Other optimization flags:

Add the keywords:
-fno-inline –std=c99

Optimizing C Applications

❑ Set the FPU to NEON

- Select ***'Miscellaneous'*** and then modify the ***'–mfpu'*** flag
- Click ***'OK'***



Properties for Hello_Zybo

Settings

Configuration: Debug [Active]

Other flags: -c -fmessage-length=0 -MT"$@" -mcpu=cortex-a9 -mfpu=neon -mfloat-abi=hard

☐ Verbose (-v)

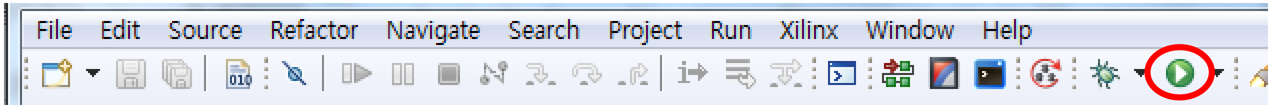
☐ Support ANSI programs (-ansi)

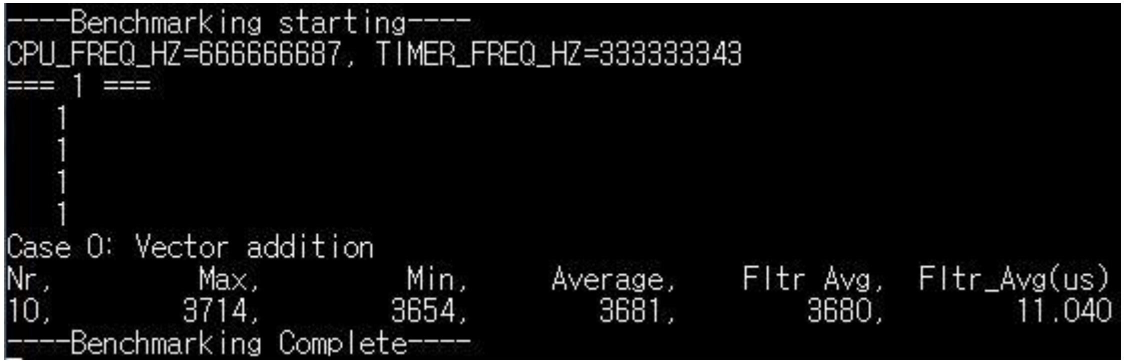
-mfpu=neon

Optimizing C Applications

❑ Run the application

- Click the ***'Run As'*** icon to run the application again
- Check the output on ***'Tera Term'***.
  - ✓ Compare the outputs and check the performance gain.

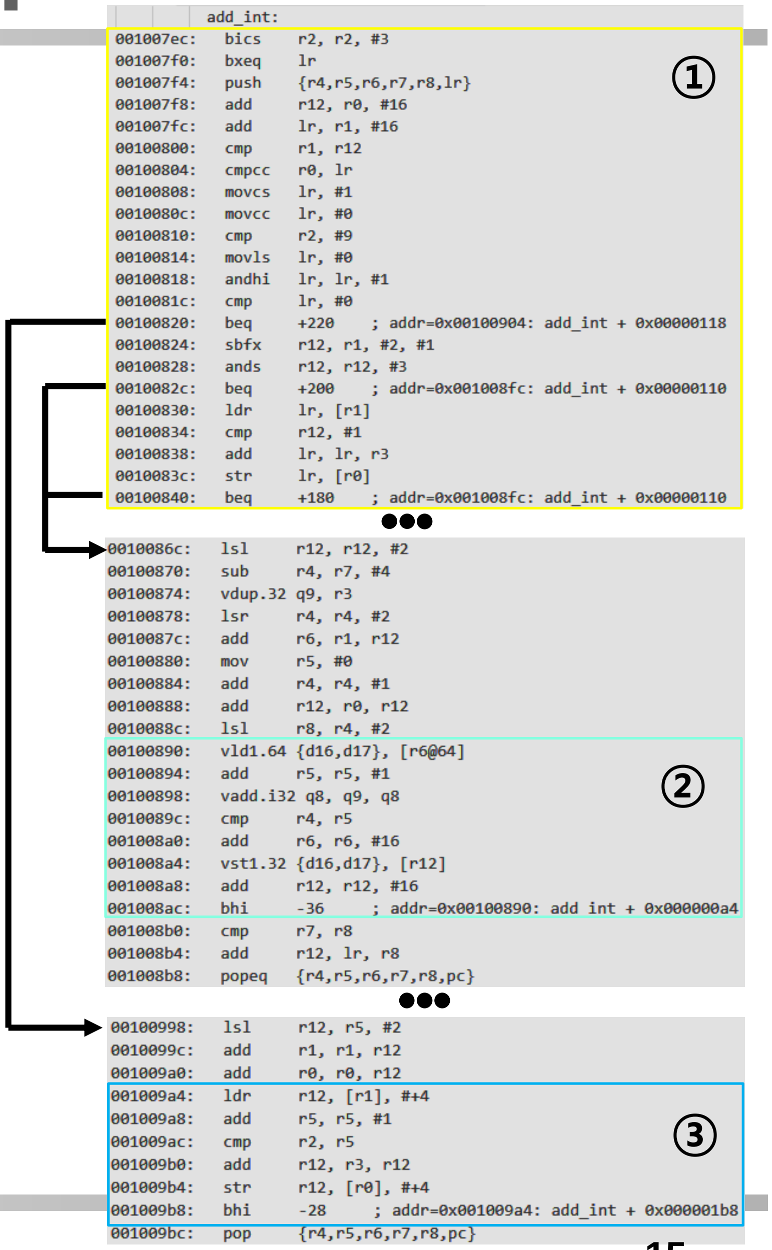




```
----Benchmarking starting----
CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343
=== 1 ===
    1
    1
    1
    1
Case 0: Vector addition
Nr,         Max,        Min,      Average,    Fltr Avg,  Fltr_Avg(us)
10,        3714,       3654,        3681,       3680,       11.040
----Benchmarking Complete----
```

Optimizing C Applications

- ❑ Review the disassembly
  - ① Check on loop-carried dependency.
  - ② Four 32-bit additions per loop (no dependence)
  - ③ One 32-bit addition per loop



```
         add_int:
001007ec:   bics    r2, r2, #3
001007f0:   bxeq    lr
001007f4:   push    {r4,r5,r6,r7,r8,lr}
001007f8:   add     r12, r0, #16
001007fc:   add     lr, r1, #16
00100800:   cmp     r1, r12
00100804:   cmpcc   r0, lr
00100808:   movcs   lr, #1
0010080c:   movcc   lr, #0
00100810:   cmp     r2, #9
00100814:   movls   lr, #0
00100818:   andhi   lr, lr, #1
0010081c:   cmp     lr, #0
00100820:   beq     +220     ; addr=0x00100904: add_int + 0x00000118
00100824:   sbfx    r12, r1, #2, #1
00100828:   ands    r12, r12, #3
0010082c:   beq     +200     ; addr=0x001008fc: add_int + 0x00000110
00100830:   ldr     lr, [r1]
00100834:   cmp     r12, #1
00100838:   add     lr, lr, r3
0010083c:   str     lr, [r0]
00100840:   beq     +180     ; addr=0x001008fc: add_int + 0x00000110
```

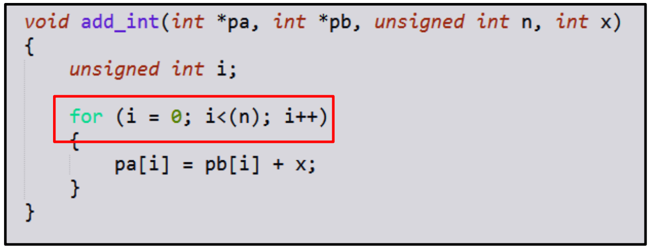
```
0010086c:   lsl     r12, r12, #2
00100870:   sub     r4, r7, #4
00100874:   vdup.32 q9, r3
00100878:   lsr     r4, r4, #2
0010087c:   add     r6, r1, r12
00100880:   mov     r5, #0
00100884:   add     r4, r4, #1
00100888:   add     r12, r0, r12
0010088c:   lsl     r8, r4, #2
00100890:   vld1.64 {d16,d17}, [r6@64]
00100894:   add     r5, r5, #1
00100898:   vadd.i32 q8, q9, q8
0010089c:   cmp     r4, r5
001008a0:   add     r6, r6, #16
001008a4:   vst1.32 {d16,d17}, [r12]
001008a8:   add     r12, r12, #16
001008ac:   bhi     -36      ; addr=0x00100890: add int + 0x000000a4
001008b0:   cmp     r7, r8
001008b4:   add     r12, lr, r8
001008b8:   popeq   {r4,r5,r6,r7,r8,pc}
```

```
00100998:   lsl     r12, r5, #2
0010099c:   add     r1, r1, r12
001009a0:   add     r0, r0, r12
001009a4:   ldr     r12, [r1], #+4
001009a8:   add     r5, r5, #1
001009ac:   cmp     r2, r5
001009b0:   add     r12, r3, r12
001009b4:   str     r12, [r0], #+4
001009b8:   bhi     -28      ; addr=0x001009a4: add_int + 0x000001b8
001009bc:   pop     {r4,r5,r6,r7,r8,pc}
```

Optimizing C Applications

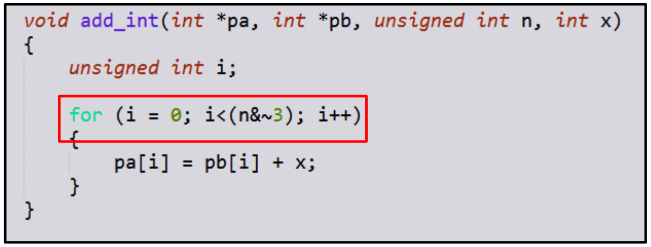
❑ Modify the C application

- Such that the number of iterations is ***‘multiple of 4’***



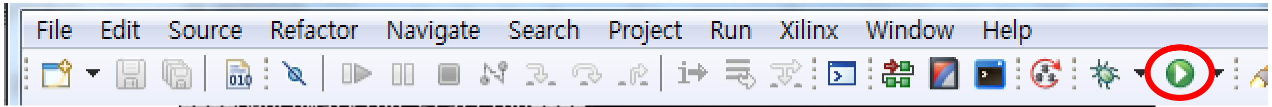
```
void add_int(int *pa, int *pb, unsigned int n, int x)
{
    unsigned int i;

    for (i = 0; i<(n); i++)
    {
        pa[i] = pb[i] + x;
    }
}
```

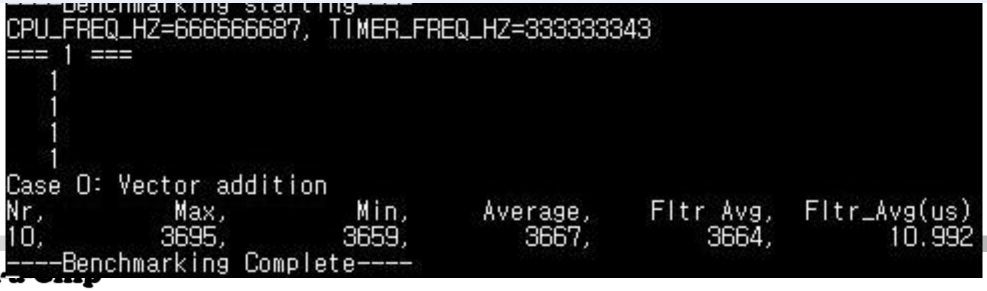
⇨



```
void add_int(int *pa, int *pb, unsigned int n, int x)
{
    unsigned int i;

    for (i = 0; i<(n&~3); i++)
    {
        pa[i] = pb[i] + x;
    }
}
```

- Click the ***‘Run As’*** icon to run the application
- Check the output on ***‘Tera Term’***.
  - ✓ Compare the outputs and check the performance gain.

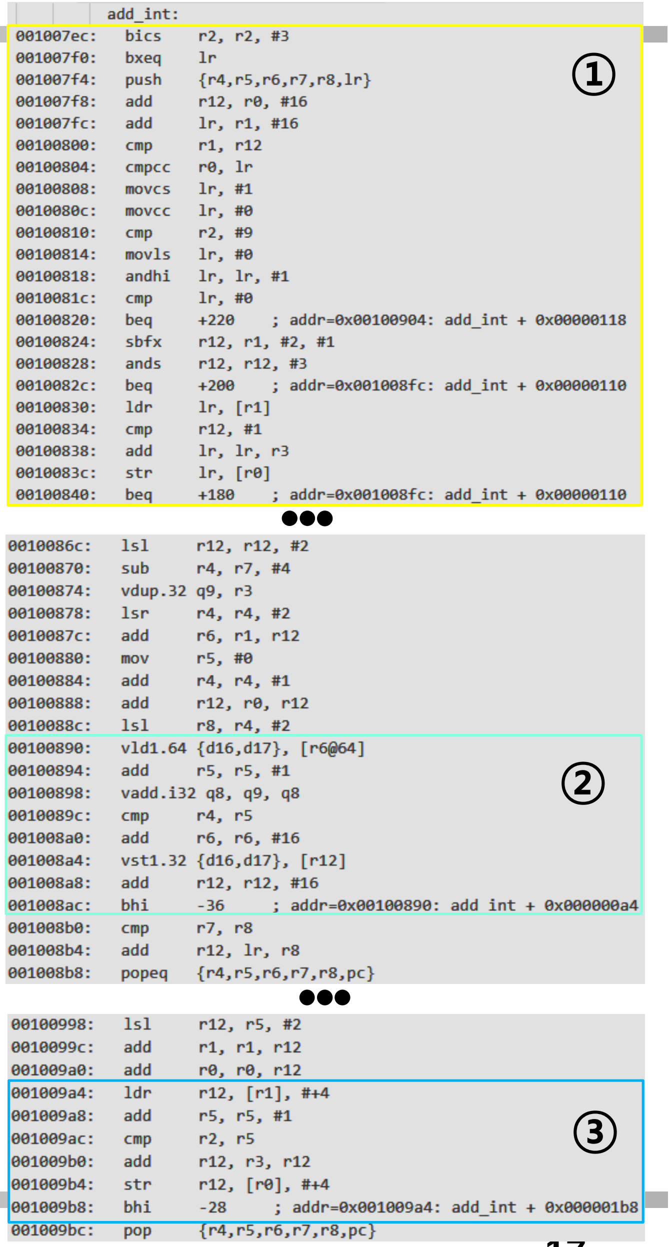




```
----Benchmarking starting----
CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343
=== 1 ===
     1
     1
     1
     1
Case 0: Vector addition
Nr,          Max,          Min,       Average,     Fltr Avg,  Fltr_Avg(us)
10,         3695,         3659,         3667,         3664,        10.992
----Benchmarking Complete----
```

Optimizing C Applications

- ❑ Review the disassembly again
  - ① Check on loop-carried dependency.
  - ② Four 32-bit additions per loop (no dependence)
  - ③ One 32-bit addition per loop
    - ✓ It never runs. Why?

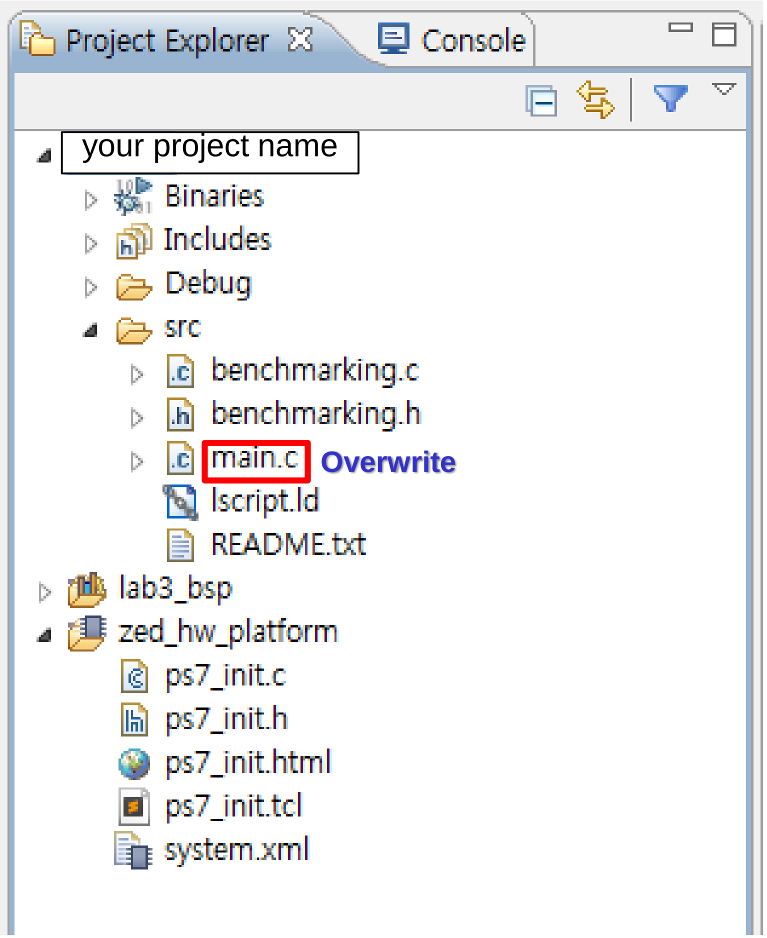


```
            add_int:
001007ec:   bics     r2, r2, #3
001007f0:   bxeq     lr
001007f4:   push     {r4,r5,r6,r7,r8,lr}
001007f8:   add      r12, r0, #16
001007fc:   add      lr, r1, #16
00100800:   cmp      r1, r12
00100804:   cmpcc    r0, lr
00100808:   movcs    lr, #1
0010080c:   movcc    lr, #0
00100810:   cmp      r2, #9
00100814:   movls    lr, #0
00100818:   andhi    lr, lr, #1
0010081c:   cmp      lr, #0
00100820:   beq      +220     ; addr=0x00100904: add_int + 0x00000118
00100824:   sbfx     r12, r1, #2, #1
00100828:   ands     r12, r12, #3
0010082c:   beq      +200     ; addr=0x001008fc: add_int + 0x00000110
00100830:   ldr      lr, [r1]
00100834:   cmp      r12, #1
00100838:   add      lr, lr, r3
0010083c:   str      lr, [r0]
00100840:   beq      +180     ; addr=0x001008fc: add_int + 0x00000110
                          ●●●
0010086c:   lsl      r12, r12, #2
00100870:   sub      r4, r7, #4
00100874:   vdup.32 q9, r3
00100878:   lsr      r4, r4, #2
0010087c:   add      r6, r1, r12
00100880:   mov      r5, #0
00100884:   add      r4, r4, #1
00100888:   add      r12, r0, r12
0010088c:   lsl      r8, r4, #2
00100890:   vld1.64 {d16,d17}, [r6@64]
00100894:   add      r5, r5, #1
00100898:   vadd.i32 q8, q9, q8
0010089c:   cmp      r4, r5
001008a0:   add      r6, r6, #16
001008a4:   vst1.32 {d16,d17}, [r12]
001008a8:   add      r12, r12, #16
001008ac:   bhi      -36      ; addr=0x00100890: add int + 0x000000a4
001008b0:   cmp      r7, r8
001008b4:   add      r12, lr, r8
001008b8:   popeq    {r4,r5,r6,r7,r8,pc}
                          ●●●
00100998:   lsl      r12, r5, #2
0010099c:   add      r1, r1, r12
001009a0:   add      r0, r0, r12
001009a4:   ldr      r12, [r1], #+4
001009a8:   add      r5, r5, #1
001009ac:   cmp      r2, r5
001009b0:   add      r12, r3, r12
001009b4:   str      r12, [r0], #+4
001009b8:   bhi      -28      ; addr=0x001009a4: add_int + 0x000001b8
001009bc:   pop      {r4,r5,r6,r7,r8,pc}
```

Optimizing C Applications

- ❑ Overwrite the following file (***'main.c'***) into the ***'src'*** folder.

main.c.2



Project Explorer   Console

```
your project name
    Binaries
    Includes
    Debug
    src
        benchmarking.c
        benchmarking.h
        main.c   Overwrite
        lscript.ld
        README.txt
lab3_bsp
zed_hw_platform
    ps7_init.c
    ps7_init.h
    ps7_init.html
    ps7_init.tcl
    system.xml
```

Optimizing C Applications

❑ Review the source code: ‘***main***’

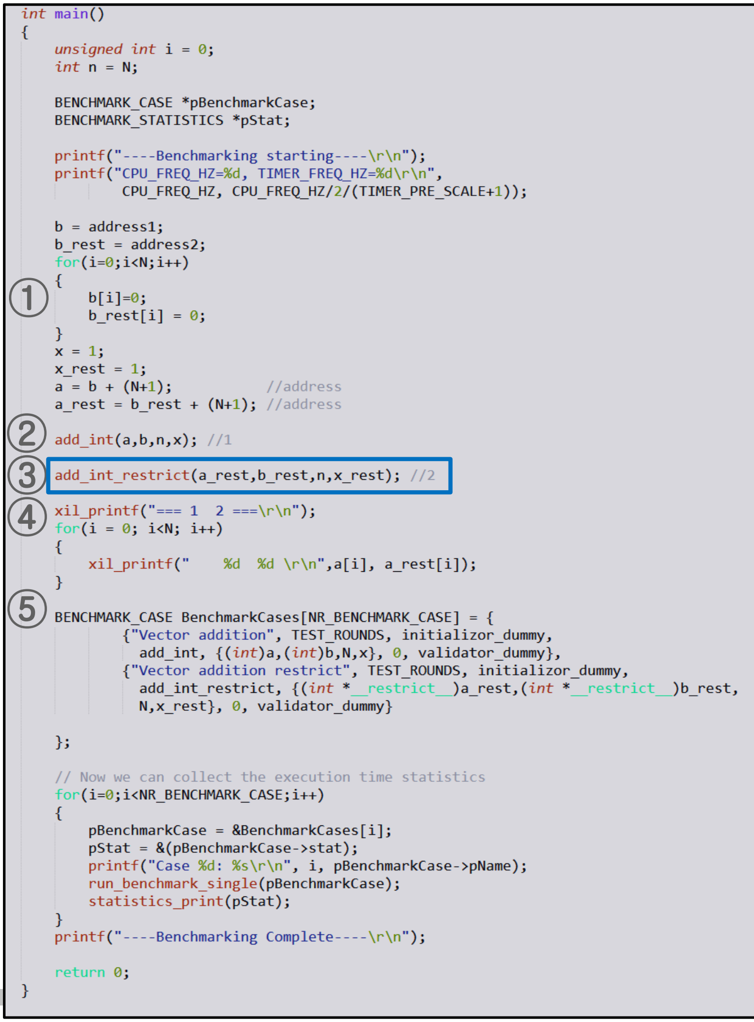
① Input a sequence

② Call ‘***add_int()***’

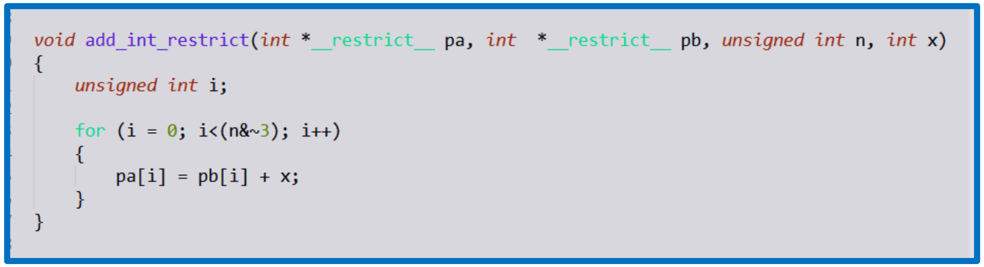
③ Call ‘***add_int_restrict()***’

④ Compare the outputs

⑤ Measure the execution times



```
int main()
{
    unsigned int i = 0;
    int n = N;

    BENCHMARK_CASE *pBenchmarkCase;
    BENCHMARK_STATISTICS *pStat;

    printf("----Benchmarking starting----\r\n");
    printf("CPU_FREQ_HZ=%d, TIMER_FREQ_HZ=%d\r\n",
            CPU_FREQ_HZ, CPU_FREQ_HZ/2/(TIMER_PRE_SCALE+1));

    b = address1;
    b_rest = address2;
    for(i=0;i<N;i++)
    {
①       b[i]=0;
        b_rest[i] = 0;
    }
    x = 1;
    x_rest = 1;
    a = b + (N+1);            //address
    a_rest = b_rest + (N+1); //address

②   add_int(a,b,n,x); //1

③   add_int_restrict(a_rest,b_rest,n,x_rest); //2

④   xil_printf("=== 1  2 ===\r\n");
    for(i = 0; i<N; i++)
    {
        xil_printf("    %d  %d \r\n",a[i], a_rest[i]);
    }
⑤
    BENCHMARK_CASE BenchmarkCases[NR_BENCHMARK_CASE] = {
            {"Vector addition", TEST_ROUNDS, initializor_dummy,
              add_int, {(int)a,(int)b,N,x}, 0, validator_dummy},
            {"Vector addition restrict", TEST_ROUNDS, initializor_dummy,
              add_int_restrict, {(int *__restrict__)a_rest,(int *__restrict__)b_rest,
              N,x_rest}, 0, validator_dummy}

    };

    // Now we can collect the execution time statistics
    for(i=0;i<NR_BENCHMARK_CASE;i++)
    {
        pBenchmarkCase = &BenchmarkCases[i];
        pStat = &(pBenchmarkCase->stat);
        printf("Case %d: %s\r\n", i, pBenchmarkCase->pName);
        run_benchmark_single(pBenchmarkCase);
        statistics_print(pStat);
    }
    printf("----Benchmarking Complete----\r\n");

    return 0;
}
```

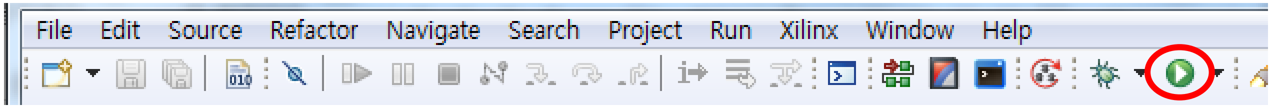


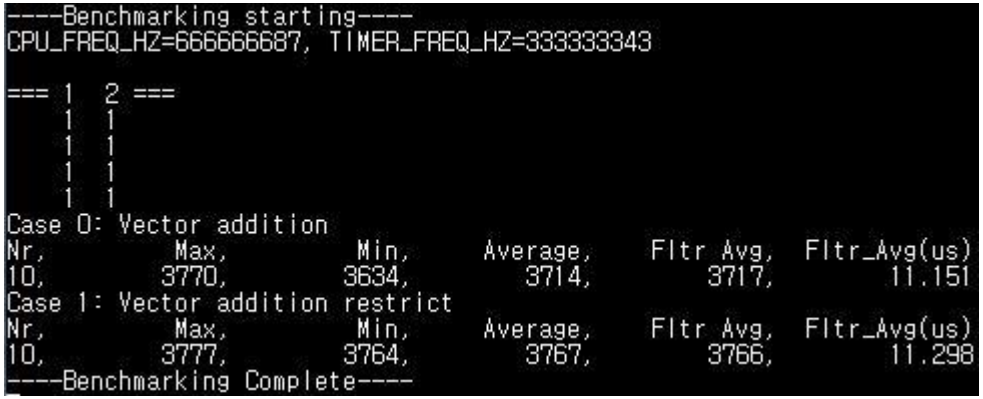
```
void add_int_restrict(int *__restrict__ pa, int  *__restrict__ pb, unsigned int n, int x)
{
    unsigned int i;

    for (i = 0; i<(n&~3); i++)
    {
        pa[i] = pb[i] + x;
    }
}
```

Optimizing C Applications

❑ Run the application

- Click the ***'Run As'*** icon to run the application again
- Check the output on ***'Tera Term'***.
  - ✓ Compare the outputs and check the performance gain.

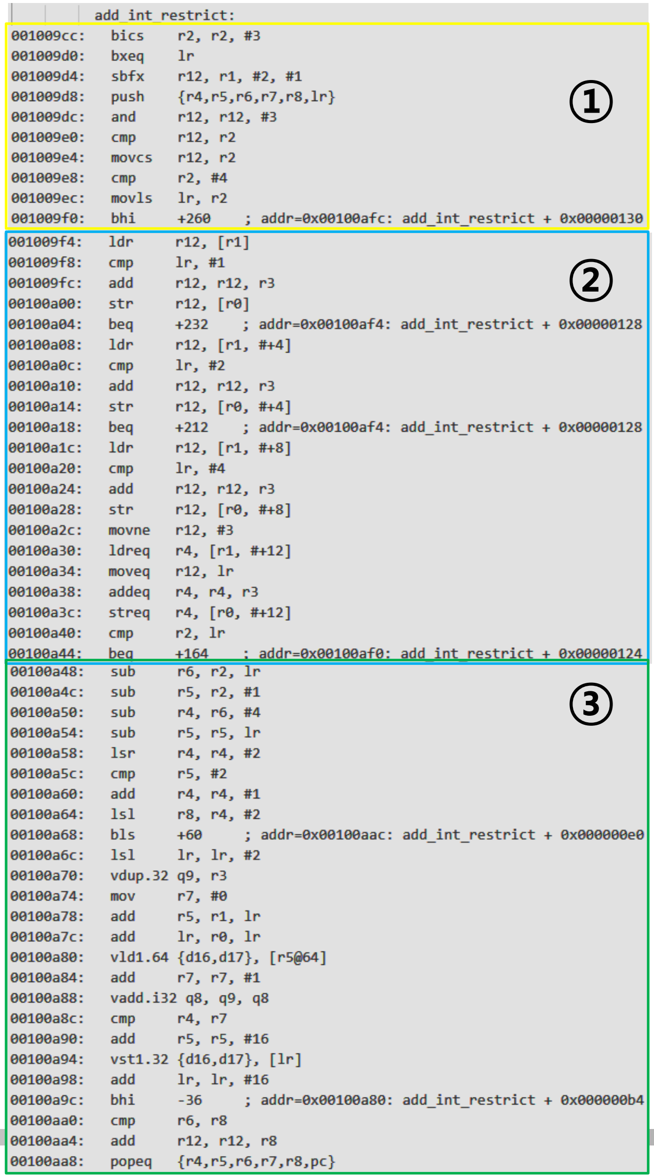




```
----Benchmarking starting----
CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343

=== 1  2 ===
    1  1
    1  1
    1  1
    1  1
Case 0: Vector addition
Nr,          Max,         Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,         3770,        3634,         3714,         3717,        11.151
Case 1: Vector addition restrict
Nr,          Max,         Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,         3777,        3764,         3767,         3766,        11.298
----Benchmarking Complete----
```

Optimizing C Applications

- ❑ Review the disassembly
  - ① Check on double word boundary.
  - ② One ~ Four 32-bit addition
  - ③ Four 32-bit additions per loop

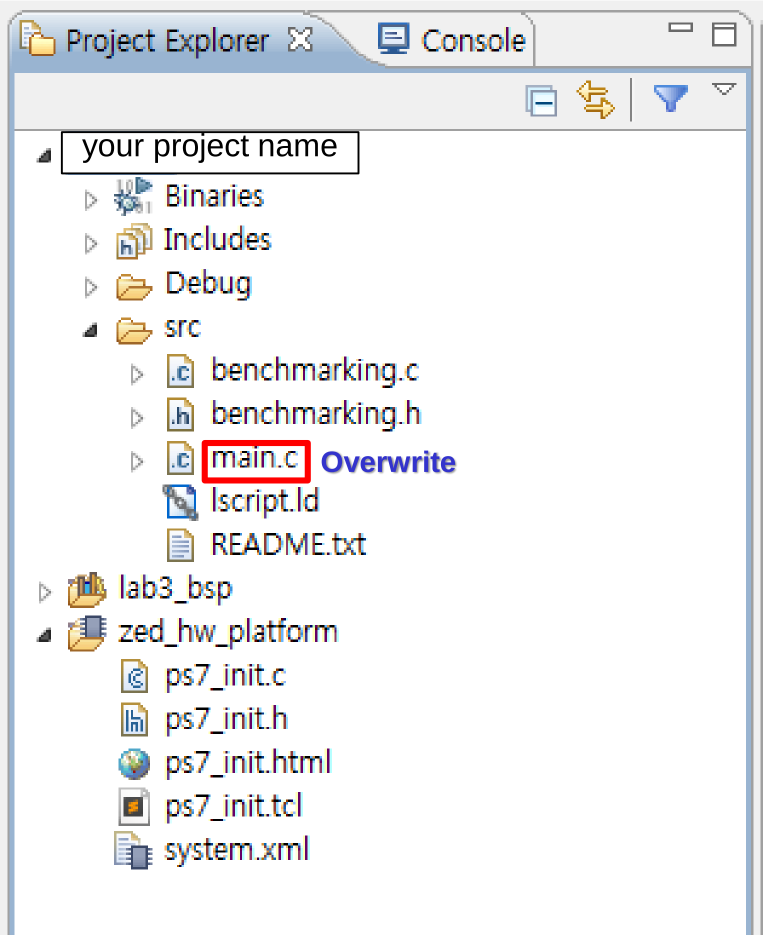


```
            add_int_restrict:
001009cc:   bics    r2, r2, #3
001009d0:   bxeq    lr
001009d4:   sbfx    r12, r1, #2, #1
001009d8:   push    {r4,r5,r6,r7,r8,lr}
001009dc:   and     r12, r12, #3
001009e0:   cmp     r12, r2
001009e4:   movcs   r12, r2
001009e8:   cmp     r2, #4
001009ec:   movls   lr, r2
001009f0:   bhi     +260     ; addr=0x00100afc: add_int_restrict + 0x00000130
001009f4:   ldr     r12, [r1]
001009f8:   cmp     lr, #1
001009fc:   add     r12, r12, r3
00100a00:   str     r12, [r0]
00100a04:   beq     +232     ; addr=0x00100af4: add_int_restrict + 0x00000128
00100a08:   ldr     r12, [r1, #+4]
00100a0c:   cmp     lr, #2
00100a10:   add     r12, r12, r3
00100a14:   str     r12, [r0, #+4]
00100a18:   beq     +212     ; addr=0x00100af4: add_int_restrict + 0x00000128
00100a1c:   ldr     r12, [r1, #+8]
00100a20:   cmp     lr, #4
00100a24:   add     r12, r12, r3
00100a28:   str     r12, [r0, #+8]
00100a2c:   movne   r12, #3
00100a30:   ldreq   r4, [r1, #+12]
00100a34:   moveq   r12, lr
00100a38:   addeq   r4, r4, r3
00100a3c:   streq   r4, [r0, #+12]
00100a40:   cmp     r2, lr
00100a44:   beq     +164     ; addr=0x00100af0: add_int_restrict + 0x00000124
00100a48:   sub     r6, r2, lr
00100a4c:   sub     r5, r2, #1
00100a50:   sub     r4, r6, #4
00100a54:   sub     r5, r5, lr
00100a58:   lsr     r4, r4, #2
00100a5c:   cmp     r5, #2
00100a60:   add     r4, r4, #1
00100a64:   lsl     r8, r4, #2
00100a68:   bls     +60      ; addr=0x00100aac: add_int_restrict + 0x000000e0
00100a6c:   lsl     lr, lr, #2
00100a70:   vdup.32 q9, r3
00100a74:   mov     r7, #0
00100a78:   add     r5, r1, lr
00100a7c:   add     lr, r0, lr
00100a80:   vld1.64 {d16,d17}, [r5@64]
00100a84:   add     r7, r7, #1
00100a88:   vadd.i32 q8, q9, q8
00100a8c:   cmp     r4, r7
00100a90:   add     r5, r5, #16
00100a94:   vst1.32 {d16,d17}, [lr]
00100a98:   add     lr, lr, #16
00100a9c:   bhi     -36      ; addr=0x00100a80: add_int_restrict + 0x000000b4
00100aa0:   cmp     r6, r8
00100aa4:   add     r12, r12, r8
00100aa8:   popeq   {r4,r5,r6,r7,r8,pc}
```

Programming Assembly Codes

- ❑ Overwrite the following file (***'main.c'***) into the ***'src'*** folder.

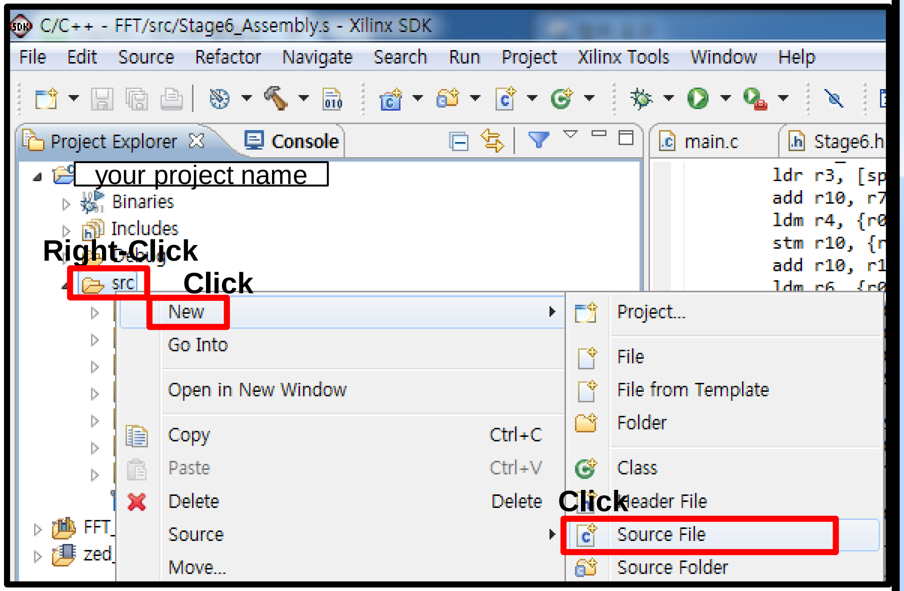
main.c.3



Programming Assembly Codes

❑ Add an assembly source code

- Click ***'src'*** > ***'New' > 'Source File'***.
- Type ***'vector_add_assembly.s'*** (using a file extension '***.s***') and then click ***'OK'***

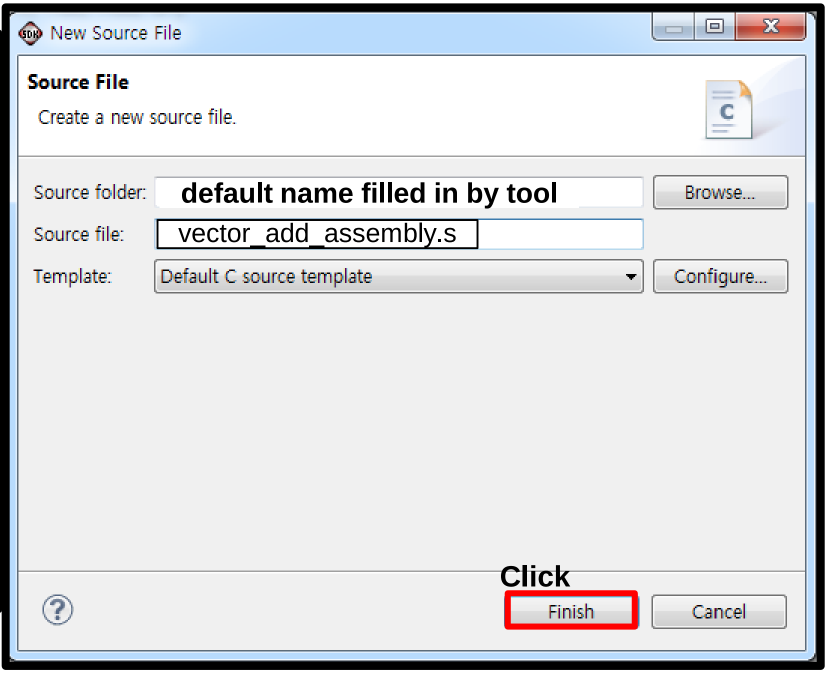


```
ldr r3, [sp
add r10, r7
ldm r4, {r0
stm r10, {r
add r10, r1
ldm r6, {r0
```

Right-Click

Click

Click



Source folder: default name filled in by tool

Source file: vector_add_assembly.s

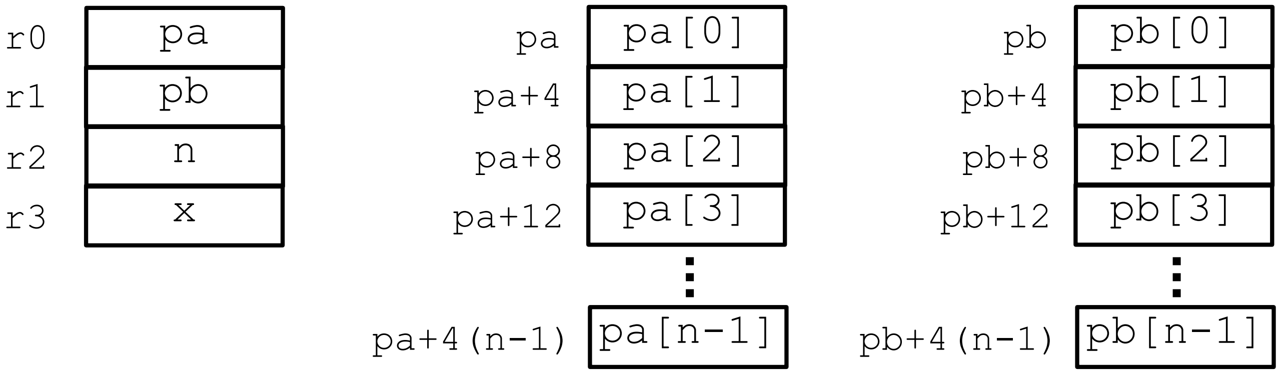
Template: Default C source template

Click

Programming Assembly Codes

❑ Add an assembly source code (cont'd)

- Register/memory setting



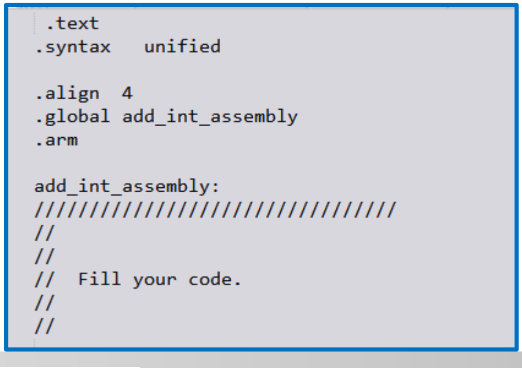
r0	pa
r1	pb
r2	n
r3	x

pa	pa[0]
pa+4	pa[1]
pa+8	pa[2]
pa+12	pa[3]
	⋮
pa+4(n-1)	pa[n-1]

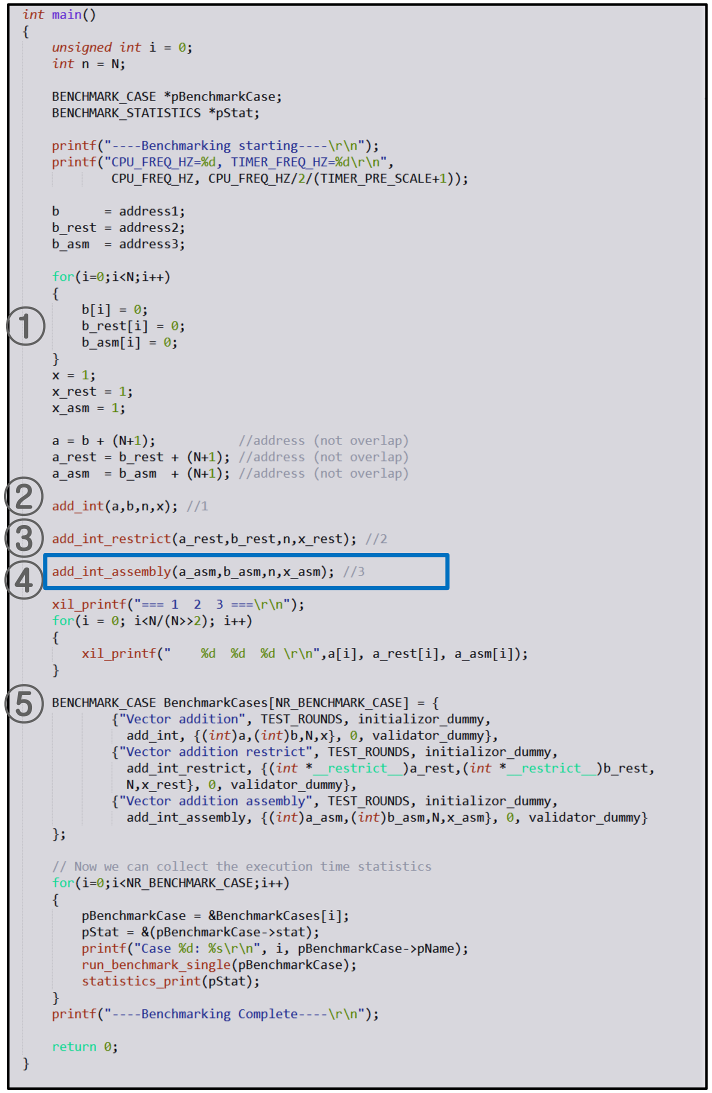
pb	pb[0]
pb+4	pb[1]
pb+8	pb[2]
pb+12	pb[3]
	⋮
pb+4(n-1)	pb[n-1]

Programming Assembly Codes

- ❑ Review the source code: ‘*main()*’
  - ① Input a sequence
  - ② Call ‘***add_int()***’
  - ③ Call ‘***add_int_restrict()***’
  - ④ Call ‘***add_int_assembly()***’
  - ⑤ Compare the outputs
  - ⑥ Measure the execution times



```
 .text
.syntax   unified

.align  4
.global add_int_assembly
.arm

add_int_assembly:
////////////////////////////////
//
//
//  Fill your code.
//
//
```

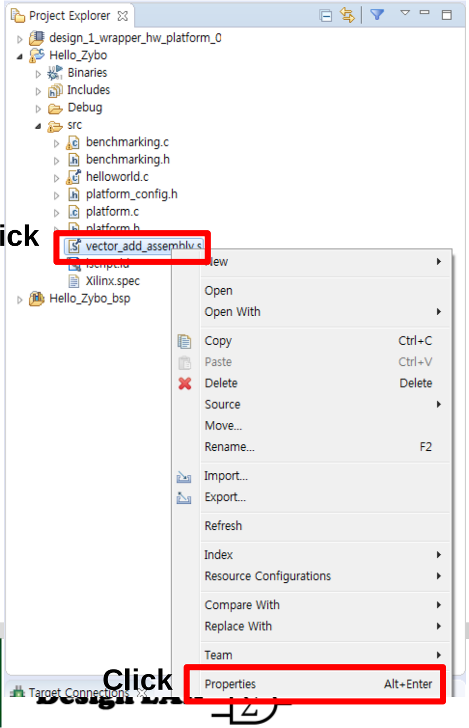


```
int main()
{
    unsigned int i = 0;
    int n = N;

    BENCHMARK_CASE *pBenchmarkCase;
    BENCHMARK_STATISTICS *pStat;

    printf("----Benchmarking starting----\r\n");
    printf("CPU_FREQ_HZ=%d, TIMER_FREQ_HZ=%d\r\n",
            CPU_FREQ_HZ, CPU_FREQ_HZ/2/(TIMER_PRE_SCALE+1));

    b      = address1;
    b_rest = address2;
    b_asm  = address3;

    for(i=0;i<N;i++)
    {
        b[i] = 0;
        b_rest[i] = 0;
        b_asm[i] = 0;
    }
    x = 1;
    x_rest = 1;
    x_asm = 1;

    a = b + (N+1);           //address (not overlap)
    a_rest = b_rest + (N+1); //address (not overlap)
    a_asm  = b_asm  + (N+1); //address (not overlap)

    add_int(a,b,n,x); //1

    add_int_restrict(a_rest,b_rest,n,x_rest); //2

    add_int_assembly(a_asm,b_asm,n,x_asm); //3

    xil_printf("=== 1  2  3 ===\r\n");
    for(i = 0; i<N/(N>>2); i++)
    {
        xil_printf("    %d  %d  %d \r\n",a[i], a_rest[i], a_asm[i]);
    }

    BENCHMARK_CASE BenchmarkCases[NR_BENCHMARK_CASE] = {
            {"Vector addition", TEST_ROUNDS, initializor_dummy,
             add_int, {(int)a,(int)b,N,x}, 0, validator_dummy},
            {"Vector addition restrict", TEST_ROUNDS, initializor_dummy,
             add_int_restrict, {(int *__restrict__)a_rest,(int *__restrict__)b_rest,
             N,x_rest}, 0, validator_dummy},
            {"Vector addition assembly", TEST_ROUNDS, initializor_dummy,
             add_int_assembly, {(int)a_asm,(int)b_asm,N,x_asm}, 0, validator_dummy}
    };

    // Now we can collect the execution time statistics
    for(i=0;i<NR_BENCHMARK_CASE;i++)
    {
        pBenchmarkCase = &BenchmarkCases[i];
        pStat = &(pBenchmarkCase->stat);
        printf("Case %d: %s\r\n", i, pBenchmarkCase->pName);
        run_benchmark_single(pBenchmarkCase);
        statistics_print(pStat);
    }
    printf("----Benchmarking Complete----\r\n");

    return 0;
}
```

Programming Assembly Codes

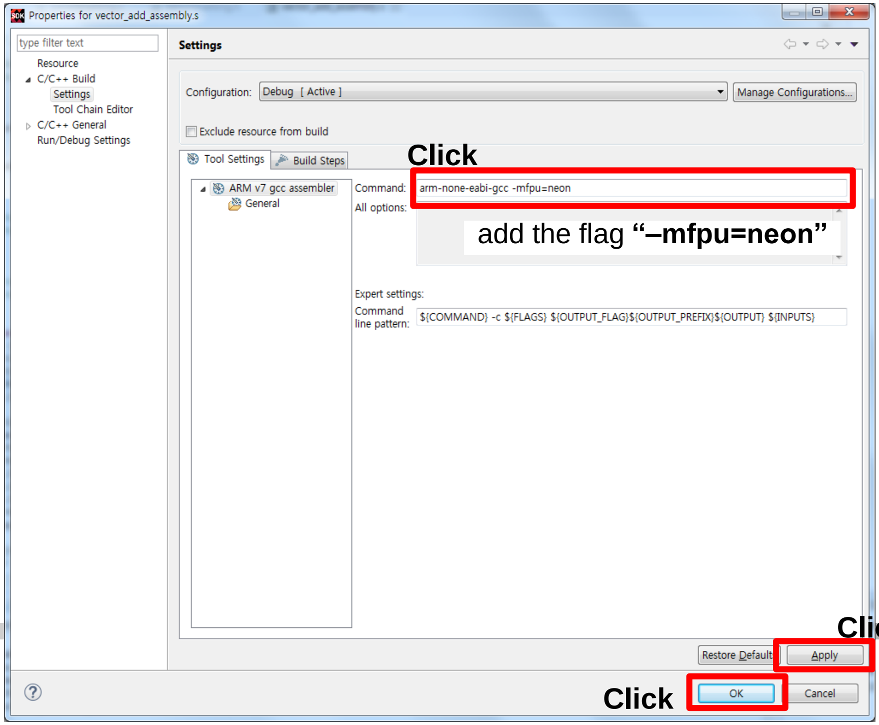
❑ Set the FPU to NEON for assembly file (*.s)

- Right-click ***'vector_add_assembly.c' > 'Properties'***
- Select **'Settings' > '*ARM v7 gcc assembler*'** and then modify the ***'–mfpu'*** flag
- Click ***'Apply' > 'OK'***



Right-Click

Click



Configuration: Debug [Active]

□ Exclude resource from build

Command: `arm-none-eabi-gcc -mfpu=neon`

All options:

Expert settings:

Command line pattern: `${COMMAND} -c ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

Click

add the flag **"–mfpu=neon"**

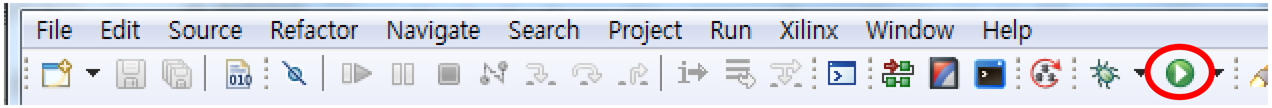
Click

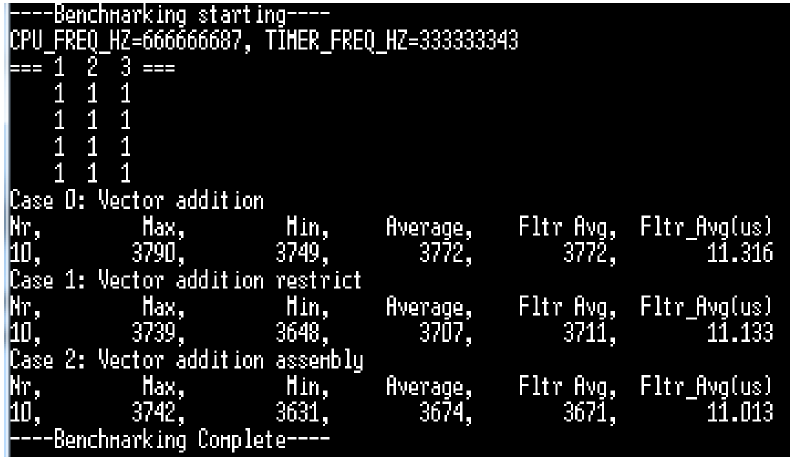
Click

Programming Assembly Codes

❑ Run the application

- Click the ***‘Run As’*** icon to run the application again
- Check the output on ***‘Tera Term’***
  - ✓ Compare the outputs and check the performance gain.



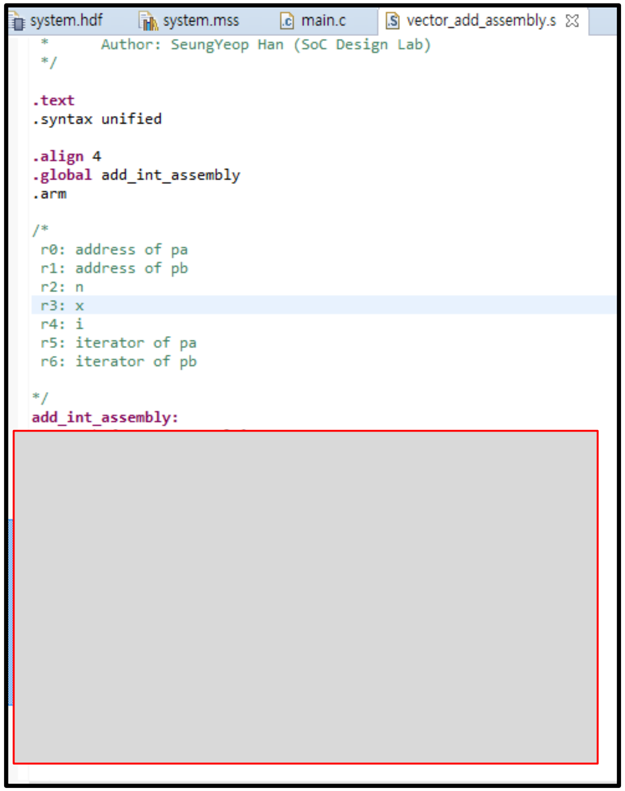


```
----Benchmarking starting----
CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343
=== 1  2  3 ===
    1  1  1
    1  1  1
    1  1  1
    1  1  1
Case 0: Vector addition
Nr,          Max,         Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,         3790,        3749,         3772,         3772,        11.316
Case 1: Vector addition restrict
Nr,          Max,         Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,         3739,        3648,         3707,         3711,        11.133
Case 2: Vector addition assembly
Nr,          Max,         Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,         3742,        3631,         3674,         3671,        11.013
----Benchmarking Complete----
```

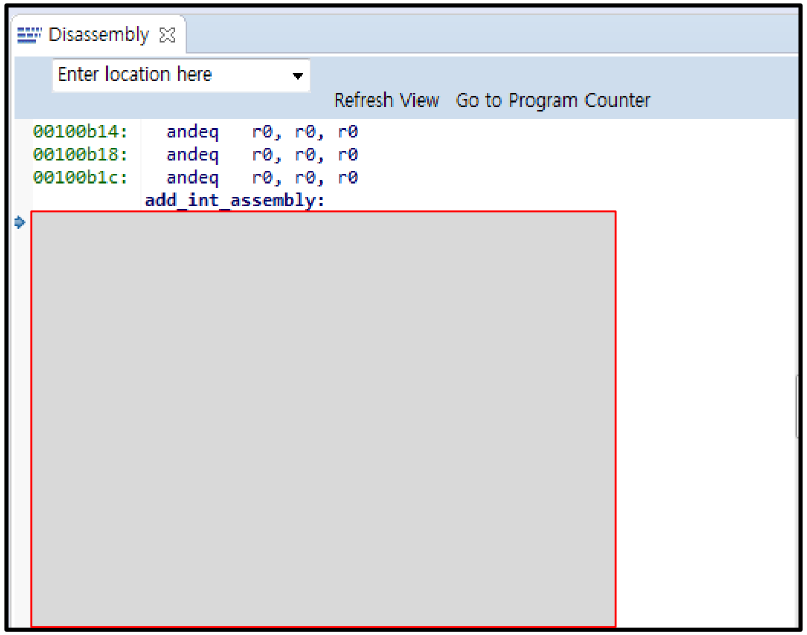
Programming Assembly Codes

❑ Review the disassembly

- Check the difference from the hand-coded assembly code



```
 *      Author: SeungYeop Han (SoC Design Lab)
 */

.text
.syntax unified

.align 4
.global add_int_assembly
.arm

/*
 r0: address of pa
 r1: address of pb
 r2: n
 r3: x
 r4: i
 r5: iterator of pa
 r6: iterator of pb

*/
add_int_assembly:
```

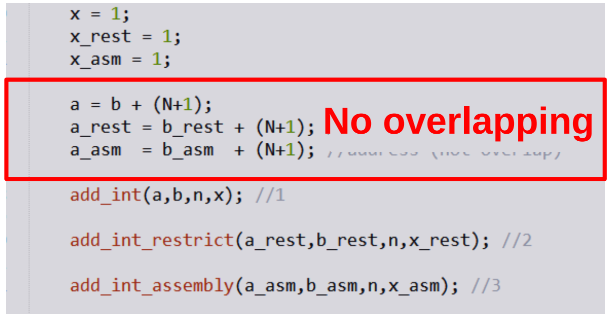


```
00100b14:   andeq   r0, r0, r0
00100b18:   andeq   r0, r0, r0
00100b1c:   andeq   r0, r0, r0
          add_int_assembly:
```

Programming Assembly Codes

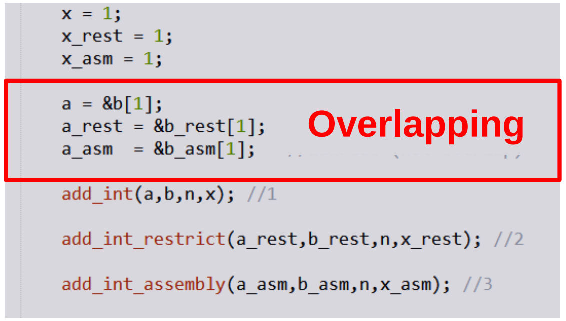
❑ Modify the C application

- Such that the memory regions overlap with each other



```
x = 1;
x_rest = 1;
x_asm = 1;

a = b + (N+1);
a_rest = b_rest + (N+1);
a_asm  = b_asm  + (N+1); //address (not overlap)

add_int(a,b,n,x); //1

add_int_restrict(a_rest,b_rest,n,x_rest); //2

add_int_assembly(a_asm,b_asm,n,x_asm); //3
```

No overlapping



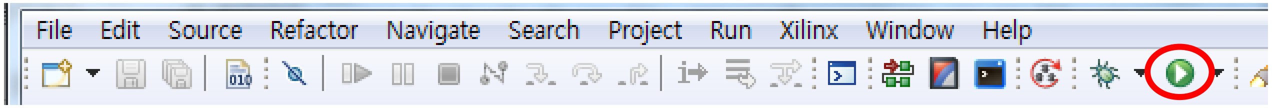
```
x = 1;
x_rest = 1;
x_asm = 1;

a = &b[1];
a_rest = &b_rest[1];
a_asm  = &b_asm[1];

add_int(a,b,n,x); //1

add_int_restrict(a_rest,b_rest,n,x_rest); //2

add_int_assembly(a_asm,b_asm,n,x_asm); //3
```

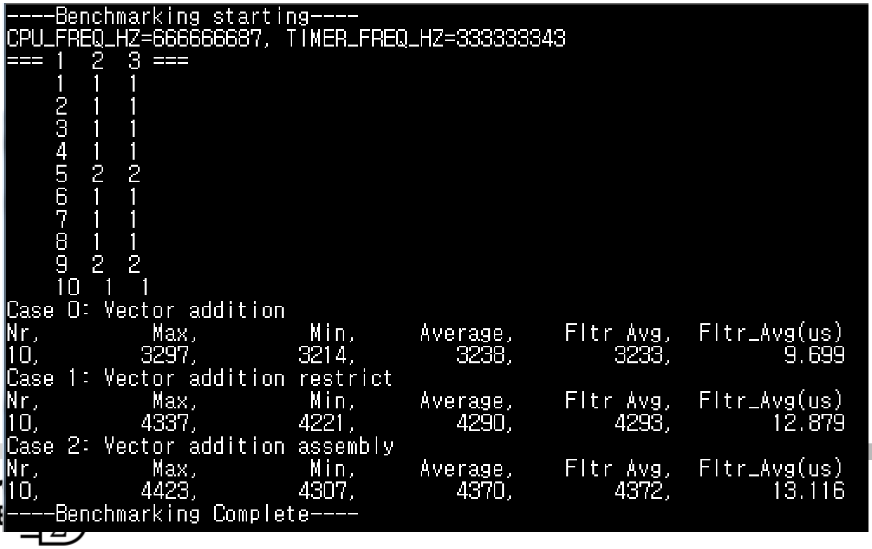
Overlapping

Programming Assembly Codes

❑ Run the application

- Click the ***'Run As'*** icon to run the application again
- Check the output on ***'Tera Term'***
  - ✓ Compare the outputs and figure out why they differ from one another

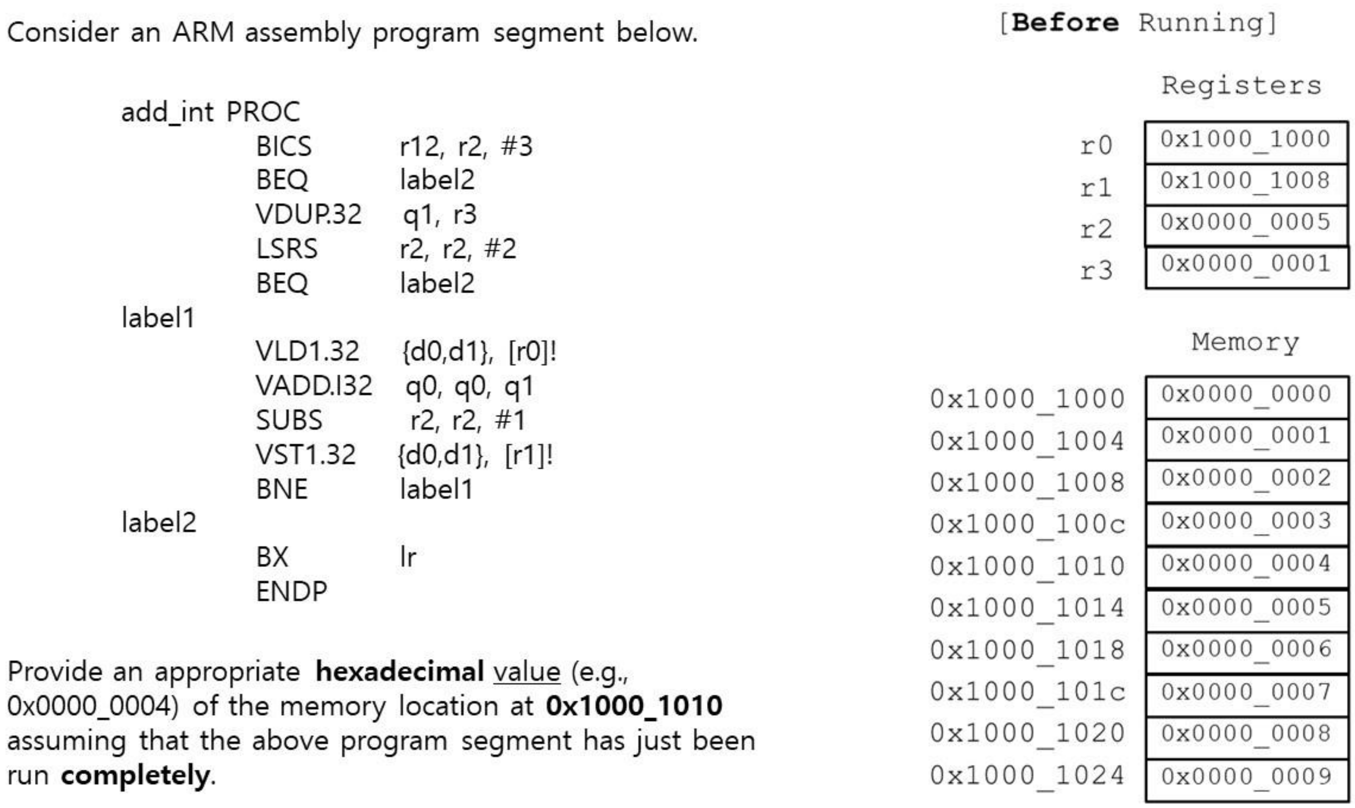




```
----Benchmarking starting----
CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343
=== 1  2  3 ===
    1  1  1
    2  1  1
    3  1  1
    4  1  1
    5  2  2
    6  1  1
    7  1  1
    8  1  1
    9  2  2
    10  1  1
Case 0: Vector addition
Nr,         Max,         Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,        3297,        3214,         3238,         3233,         9.699
Case 1: Vector addition restrict
Nr,         Max,         Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,        4337,        4221,         4290,         4293,        12.879
Case 2: Vector addition assembly
Nr,         Max,         Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,        4423,        4307,         4370,         4372,        13.116
----Benchmarking Complete----
```

Demo

- ❑ Modify the C application and run it to check the answer to the following question.



Consider an ARM assembly program segment below.

```
add_int PROC
            BICS        r12, r2, #3
            BEQ         label2
            VDUP.32      q1, r3
            LSRS        r2, r2, #2
            BEQ         label2
label1
            VLD1.32      {d0,d1}, [r0]!
            VADD.I32     q0, q0, q1
            SUBS         r2, r2, #1
            VST1.32     {d0,d1}, [r1]!
            BNE         label1
label2
            BX          lr
            ENDP
```

Provide an appropriate **hexadecimal** value (e.g., 0x0000_0004) of the memory location at **0x1000_1010** assuming that the above program segment has just been run **completely**.

[**Before** Running]

Registers

r0	0x1000_1000
r1	0x1000_1008
r2	0x0000_0005
r3	0x0000_0001

Memory

0x1000_1000	0x0000_0000
0x1000_1004	0x0000_0001
0x1000_1008	0x0000_0002
0x1000_100c	0x0000_0003
0x1000_1010	0x0000_0004
0x1000_1014	0x0000_0005
0x1000_1018	0x0000_0006
0x1000_101c	0x0000_0007
0x1000_1020	0x0000_0008
0x1000_1024	0x0000_0009