Design Assignment 2:

Assembly Language Programming

Embedded Systems

ELC 411

Matthew Strickland and Jacob Levine

Submission: 10/11/17

A. Previous assembly code

```
//Jacob Levine
inner_prod_asm

//take values r0 gets pointer h, r1 gets pointer x, r2 gets int n

//push r4, r5, etc
push {r4, r5}
r3 gets pointer h
//this is so we can deposit results in r0

//compute mult and addit via loop from n-1 to 0
subtract 1 from r2 to get n-1 for looping

LOOP:
LDRH r4, [r1], #2
//load from r1, decrement for next iteration #2 store in r4
LDRH r5, [r3], #2
//load from r3, decrement for next iteration #2 store in r5

mla instruction  on r4 and r5 depositing to r0

sub #1 from r2
cbnz LOOP:

ASR r0 4 bytes
//pop back to r4, r5, etc
pop {r4, r5}
//return r0
```

//Matthew Strickland

```
inner_prod_asm:
        MOV    r3, r0          // Place *h[] in r3 so r0 can be used for 'sum'
        MOV    r0, #0          // Set r0 to 0, so it can be used for sum

loop:
        LDRSH  r4, [r1], #2    // Contents of address stored in R1 (*x[]) half
word placed in R4, R1 incremented by 16 (2 bytes)
        LDRSH  r5, [r3], #2    // Contents of address stored in R3 (*h[]) half
word placed in R5, R3 incremented by 16 (1 halfword)
        MLA    r0, r4, r5, r0  // r0 = r0 + (r4 * r5); sum = sum + (h[n] * x[n])
        SUB    r2, r2, #1      // r2 = n elements of array, decrement r2 by one
each iteration
        CMP    r2, #0          // iff r2 (nth iteration) = 0, continue to next
line otherwise GoTo loop
        BGT    loop
        ASRS   r0, #16         // Shift the concatenated value of r0 to the
lower 16 bits of R0 (keep sign bit)
        BX     lr        // branch to PC address stored in link register when
executed (exit loop)

    .endfunc
    .end
```

B. Debugged assembly code and watch window

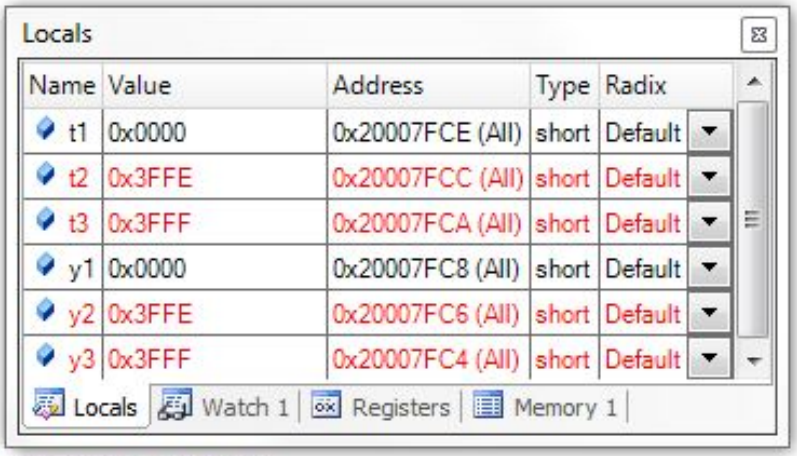
```
.syntax unified
    .text
    .global inner_prod_asm
    .func inner_prod_asm, inner_prod_asm
    .thumb_func

inner_prod_asm:
        PUSH   {r4, r5}        // Push contents of R4 and R5 so content can be
restored when function finished
        MOV    r3, r0          // Place *h[] in r3 so r0 can be used for 'sum'
        MOV    r0, #0          // Set r0 to 0, so it can be used for sum

loop:
        LDRSH  r4, [r1], #2    // Contents of address stored in R1 (*x[]) half
word placed in R4, R1 incremented by 16 (2 bytes)
        LDRSH  r5, [r3], #2    // Contents of address stored in R3 (*h[]) half
word placed in R5, R3 incremented by 16 (1 halfword)
        MLA    r0, r4, r5, r0  // r0 = r0 + (r4 * r5); sum = sum + (h[n] * x[n])
        SUB    r2, r2, #1      // r2 = n elements of array, decrement r2 by one
each iteration
        CMP    r2, #0          // iff r2 (nth iteration) = 0, continue to next
line otherwise GoTo loop
        BGT    loop
        ASRS   r0, #16         // Shift the concatenated value of r0 to the
lower 16 bits of R0 (keep sign bit)
        POP    {r4, r5}        // Restore original contents of R4 and R5 before
returning
      BX     lr          // branch to PC address stored in link register when
executed (exit loop)


    .endfunc


    .end
```



C. Table of execution time, C version (speed and none) and assembly version

Implementation	Time
Unoptimized C	150.67 µs
Speed optimized C	526.98 ns
Assembly	45.496 µs

D. Scope traces with captions

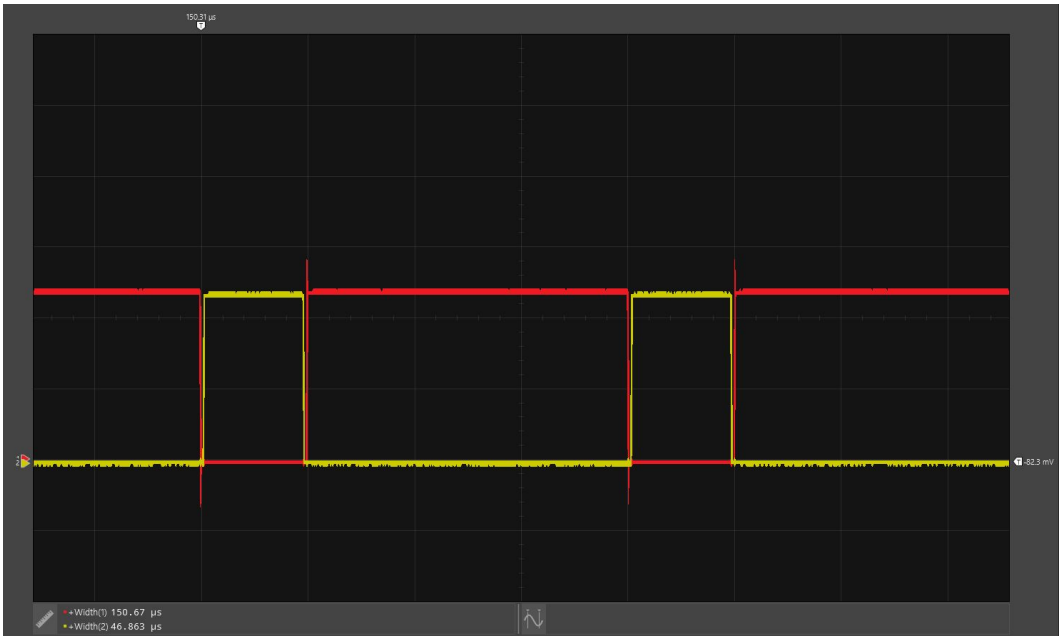


Figure: Unoptimized C (red) vs assembly (yellow)

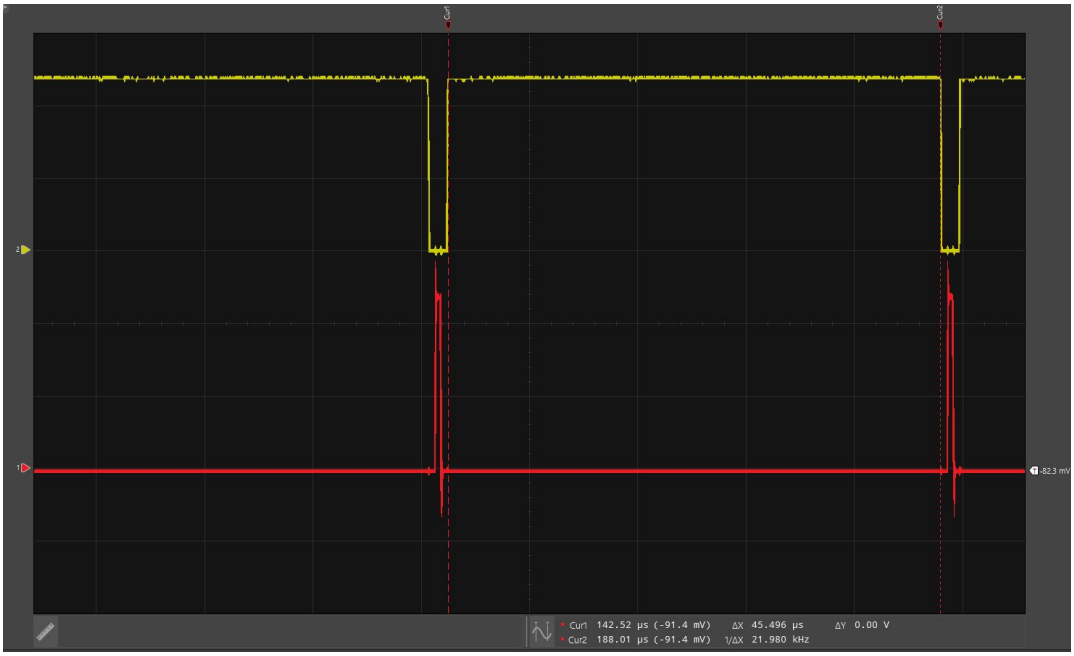


Figure: Assembly execution time (yellow)

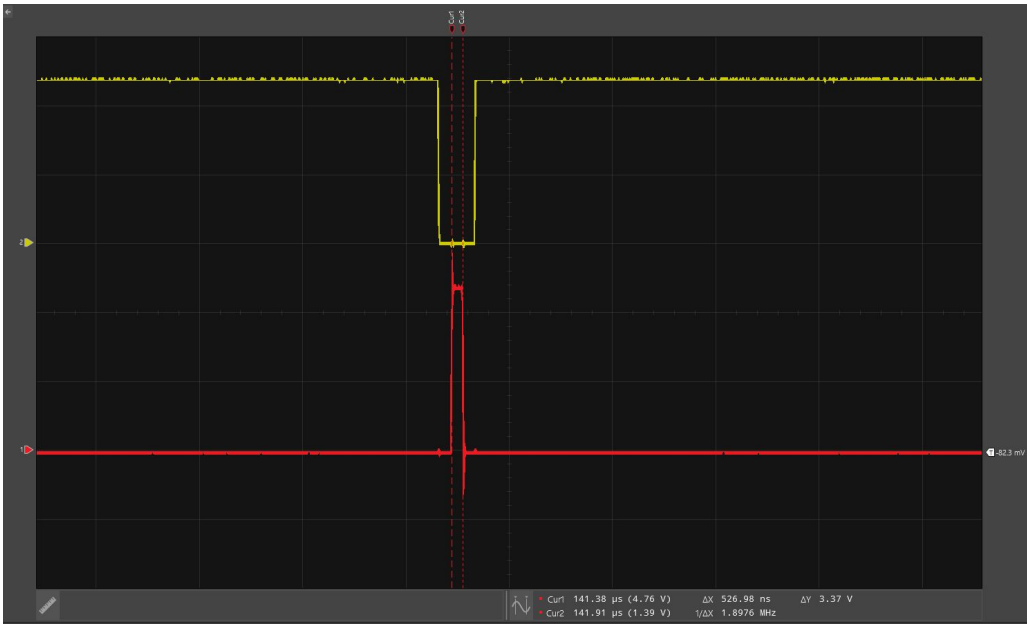


Figure: Optimized C code (red)

E. Inner_prod_gcc.s code

```
0x00000084 <inner_prod>:
  28: // Inputs:      h - pointer to array of int16_t values, length n
  29: //              x - pointer to array of int16_t values, length n
  30: // Returns:     [x (dot) h] >> 16, as an int16_t value
  31: int16_t inner_prod( int16_t *h, int16_t *x, int n )
  32: {
0x00000084 push    {r7}            //push r7 to memory, update stack pointer
with address
0x00000086 sub     sp, #1c         //Subtract 28 address from stack pointer
0x00000088 add     r7, sp, #0      //use r7 as frame pointer
0x0000008A str     r0, [r7, #c]    //store *h into r7 with offset 12
0x0000008C str     r1, [r7, #8]    //store *x into r7 with offset 8
0x0000008E str     r2, [r7, #4]    //store n into r7 with offset 4
  33:     int i;
  34:     int32_t sum = 0;
0x00000090 movs    r3, #0          //set R3 to 0, update N, Z, C flags
0x00000092 str     r3, [r7, #10]   //store [r7 + 10] into r3
  35:
  36:     for (i = 0; i < n; ++i)
0x00000094 movs    r3, #0          //set r3 to 0, update N, Z, C flags
0x00000096 str     r3, [r7, #14]   //store r3 into address [r7 + 14 bytes]
0x00000098 b.n     c4 <CYDEV_PICU_SIZE+0x14>
  37:     {
  38:         sum += (h[i] * x[i]);
0x0000009A ldr     r3, [r7, #14]       //load r3 with [r7 + 14] address
0x0000009C lsls    r3, r3, #1          //left shift r3 by 1 and update flags
0x0000009E ldr     r2, [r7, #c]        //load into r3 from r7 with offset 12
0x000000A0 add     r3, r2              //r3=r3+r2
0x000000A2 ldrsh.w r3, [r3]    //load signed halfword into r3 from [r3]
0x000000A6 mov     r1, r3              //move value of r3 into r1
0x000000A8 ldr     r3, [r7, #14]       //load r3 with [r7 + 14] address
0x000000AA lsls    r3, r3, #1          //left shift r3 by 1 and update flags
0x000000AC ldr     r2, [r7, #8]        //load r2 with [r7+8]
0x000000AE add     r3, r2              //r3=r3+r2
0x000000B0 ldrsh.w r3, [r3]    //load signed halfword into r3 from address [r3]
0x000000B4 mul.w   r3, r3, r1  //multiply 32 bit r3*r1 and store into r3
0x000000B8 ldr     r2, [r7, #10]       //load into r2 from r7 with offset 10
0x000000BA add     r3, r2              //r3=r3+r2
0x000000BC str     r3, [r7, #10]       //store r3 into r7 with offset 10
  31: int16_t inner_prod( int16_t *h, int16_t *x, int n )
  32: {
  33:     int i;
  34:     int32_t sum = 0;
  35:
  36:     for (i = 0; i < n; ++i)
```

```
0x000000BE ldr     r3, [r7, #14]    //load r3 with [r7 + 14] address
0x000000C0 adds    r3, #1           //r3=r3+1 with flags updated
0x000000C2 str     r3, [r7, #14]    //set r3 at M[r7+14]
0x000000C4 ldr     r2, [r7, #14]    //load r3 with contents of [r7 + 14] address
0x000000C6 ldr     r3, [r7, #4]     //load r3 with contents of [r7 + 4] address
0x000000C8 cmp     r2, r3           //compare r2 and r3
0x000000CA blt.n   9a <inner_prod+0x16>    //branch to .n if r2 is less than r3
  37:     {
  38:         sum += (h[i] * x[i]);
  39:     }
  40:     sum = sum >> 16;
0x000000CC ldr     r3, [r7, #10]    //set r3 to the content of M[r7+10]
0x000000CE asrs    r3, r3, #10      //arithmetic right shift r3 by 10 flags
updated
0x000000D0 str     r3, [r7, #10]    //store r3 into r7 with offset 10
  41:
  42:     return (int16_t) sum;       // return value truncated to int16_t
range
0x000000D2 ldr     r3, [r7, #10]     //loads r7 to r3 with immediate offset
0x000000D4 sxth    r3, r3            //sign extends r3 halfword
  43: }
0x000000D6 mov     r0, r3            //move r3 to r0
0x000000D8 adds    r7, #1c           //adds immediate to r7 and updates flags
0x000000DA mov     sp, r7            //change stack pointer to value in r7
0x000000DC pop     {r7}              // pops r7 back to initial value before
calling inner_prod
0x000000DE bx      lr                //branch
```