Lab 7 Efficient Load and Store in Assembly

10/29/21

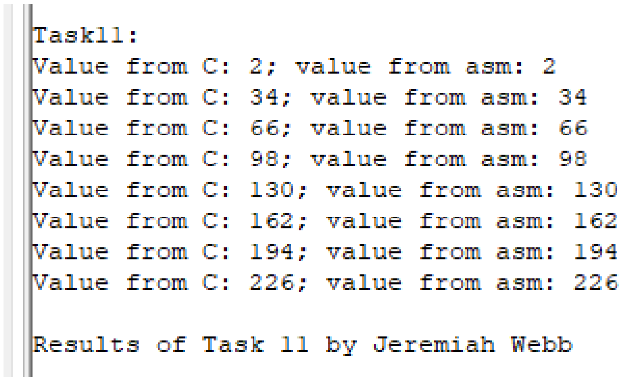
11/5/21

Jeremiah Webb

Introduction:

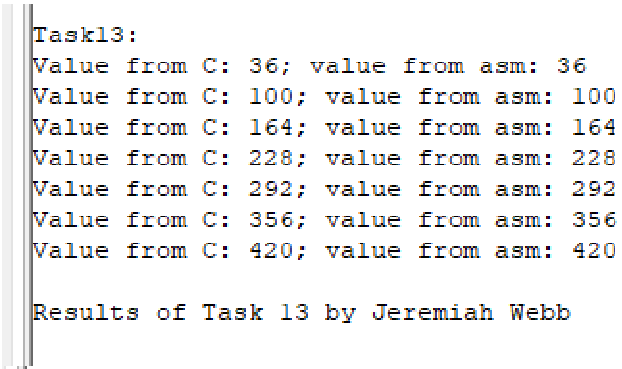
Being able to understand how loading and storing works in assembly is essential to efficient and safe programs. This lab helps specifically understanding how arguments in C are loaded and stored in assembly code, and how to effectively convert C code to assembly code.

Screenshots:



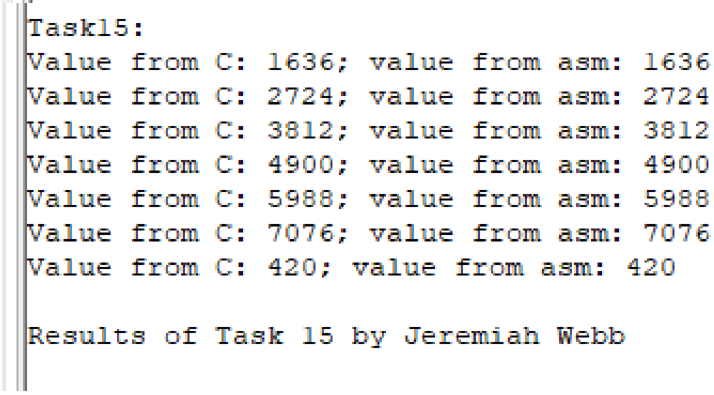
```
Task11:
Value from C: 2; value from asm: 2
Value from C: 34; value from asm: 34
Value from C: 66; value from asm: 66
Value from C: 98; value from asm: 98
Value from C: 130; value from asm: 130
Value from C: 162; value from asm: 162
Value from C: 194; value from asm: 194
Value from C: 226; value from asm: 226

Results of Task 11 by Jeremiah Webb
```



```
Task13:
Value from C: 36; value from asm: 36
Value from C: 100; value from asm: 100
Value from C: 164; value from asm: 164
Value from C: 228; value from asm: 228
Value from C: 292; value from asm: 292
Value from C: 356; value from asm: 356
Value from C: 420; value from asm: 420

Results of Task 13 by Jeremiah Webb
```



```
Task15:
Value from C: 1636; value from asm: 1636
Value from C: 2724; value from asm: 2724
Value from C: 3812; value from asm: 3812
Value from C: 4900; value from asm: 4900
Value from C: 5988; value from asm: 5988
Value from C: 7076; value from asm: 7076
Value from C: 420; value from asm: 420

Results of Task 15 by Jeremiah Webb
```

Code Snippet:

//Code for Task 11

```
task11  PROC
     MOV  r2, #0
task11_loop
        CMP  r2, r1                 ; test = r2 - r1
        BGE  task11_end             ; if test >= 0, then branch
to task11_end
        MOV  r3, r2, LSL #5         ; r3 <- r2 * 32
        ADD  r3, #2                ; r3 <- r3 + 2
        STRB r3, [r0, r2]           ; r3 -> mem[r0 + r2] or r3 -
> mem[r0 + i]
        ADD  r2, #1                 ; r2 <- r2 + 1
        B    task11_loop            ; branch to task11_loop
task11_end
          BX   lr
          ENDP
```

//Code for Task 13

```
task13  PROC
          PUSH {r4-r5, lr}
          ; r0 = gPtrArray11a
        ; r1 = gPtrArray13a
        ; r2 = gVar1
          SUB r2, #1
          MOV r3, #0
task13_loop
          CMP r3, r2
          BGE task13_end
          LDRB r4, [r0]
          LDRB r5, [r0, #1]
          ADD r4, r5
          STRH r4, [r1, r3, LSL #1]
          ADD r3, #1
          ADD r0, #1
          B task13_loop
task13_end
          POP  {r4-r5, pc}
          ENDP
```

//Code for Task 15

```
task15  PROC
        PUSH {r4-r5, lr}
        ; r0 = gPtrArray13a
        ; r1 = gPtrArray15a
        ; r2 = gVar1
        SUB r2, #1
        MOV  r3, #0; i = 0
task15_loop
        CMP  r3, r2
        BGE  task15_end
        LDRH  r4, [r0], #2; load gPtr13a to temp
        LDRH  r5, [r0]; load gPtr11a + 1
        ADD r5, r4, r5, LSL # 4; = temp + 16 * (*gPtr13a)
        STRH r5, [r1, r3, LSL #2]
        ADD  r3, #1; increment i
        B    task15_loop
task15_end
        POP  {r4-r5, pc}
        ENDP
        END
```

Discussion:

The major thing I had to deal with for the program to work was reanalyzing how the arguments in C were transferred into assembly. First grasping how those functions use the arguments in C and then being able to use the LDR and push and pop instructions assisted with the overall construction of the tasks. One major insight was how to use LDRH and how that works with differently sized variables.

Result:

I learned specifically how registers are loaded with data and how the registers can be used to simulate C functions. Furthermore, with understanding how these registers are used, I can create more efficient code than were I to use regular C. Even doing basic functions will be more efficient in assembly. Passing back and forth data from C and assembly can now be easily done as shown by this lab.