1. RISC program for ANSWER = A \* B + C \* D

Load R2, A Put A into R2

Load R3, B Put B into R3

Multiply R4, R2, R3 Get the first product and put in R4

Load R5, C Put C into R5

Load R6, D Put B into R6

Multiply R7, R5, R6 Get the second product and put in R7

Clear R8 Empty where I am going to put the Answer

Add R8, R4, R7 Adds two products, in R4 and R7 and puts in R8

Store R8, ANSWER Store the answer in memory

1. Processor Stack contents and contents of stack pointer with Fig 2.18

Elements are pushed to lower memory locations

* 1. The second Store instruction in the subroutine

976 Level 3(SP)

980

984 R3

988 R2

992 n Level 2

996 NUM1

1000 Level 1

* 1. The last Load instruction in the subroutine

976 R5 Level 3

980 R4

984 R3

988 R2

992 n Level 2 (SP)

996 R3

1000 Level 1

* 1. The last Store instruction in the calling program

976 R5 Level 3

980 R4

984 R3

988 R2

992 R2(SUM) Level 2 (SP)

996 R3

1000 Level 1

1. In Figure 3.9, the interrupt-enable bit in the PS is set last in the START section of the Main program. Why? Does the order matter for earlier operations in START? Why or why not?

The PS bit is enabled at the end of the START section because the program needs to save everything that it deems needed and prepare for possible interruptions before it can accept interruptions. The program does this to maintain integrity of the soon-to-be interrupted program, so it can be resumed after the interruption is done. If the program was allowed to be interrupted before initializing other interrupts and prepping the beginning of a program, data essential to resuming could be lost. Without the organization of enabling interrupts and then setting the interrupt-enable bit, the system could loop or only accept interrupts from one source instead of having “equal” share to interrupt (display could not get attention for a while if it was enabled after the PS). Therefore, order does matter.

1. Write a RISC-style program to read a 32-bit value V, and display it in the form 10100010111101011010000101101101 on the display device describe in Figure 3.3 (and in class).

Move R2, #V

Read: MoveByte R3, $

LoadByte R4, KBD\_STATUS

And R4, R4, #2

Branch\_if\_[R4]=0 READ

LoadByte R5, KBD\_DATA

StoreByte R5, (R2)

Add R2, R2, #1

Echo: LoadByte R4, DISP\_STATUS

And R4,R4, #4

Branch\_if\_[R4] = 0 ECHO

StoreByte R5, DISP\_DATA

Branch\_if\_[R5]!=[R3] READ