The University of Alabama in Huntsville
Electrical & Computer Engineering Department
CPE/EE 42/521 01
Sample Homework Solutions

Chapter 2

1. a. [M(0)] = 12
   b. [M([M[(7)])] = M[(6)] = M4
   c. [M([M[([M(0)])])] = [M([M(12)])] = [M(6)] = 4
   d. [M(3 + 4)] = [M(7)] = 6
   e. [M([M(9)] + 4)] = [M(5 + 4)] = [M(9)] = 5
   f. [M([M(9)] + [M(2)])] = [M(5 + 7)] = [M(12)] = 6
   g. [M(2)] * [M(13)] = 7 * 3 = 21
   h. [M(0)] * 3 + [M(1)] * 4 = 12 * 3 + 17 * 4 = 36 + 68 = 104
   i. [M([M(5)] + [M(13)] + 2 * [M(14)])] = [M(4 + 3 + 2 * 2)] = [M(7 + 4)] = [M(11)] = 7

14. a. Copy the address in A0 to A3. Therefore, A3 is loaded with 4.
    b. Copy the address –2 +[A0] to A3. Therefore A3 is loaded with –2 + 4 = 2.
    c. Move the value pointed at by [PC] + 4 into D2. [PC] + 4 = 10 + 4 = 14. [M(14)] = 2 is copied into D2.
    d. Copy the byte pointed at by 2+[PC]+[A1] into D7. The effective address is 2 + 10 + 2 = 14, and the contents of location 14 = 2.
    e. Copy the address 10+[A0]+[D0] into A3. This value is 10+4+0=14 which is loaded into A3.
    f. Add the contents of location 12 to D0. D0 contains 0 initially, the contents of memory location 12 are 6, therefore the final value in D0 is 6.
    g. The effective address is –1 + [A0] + [A1] = -1 + 4 + 2 = 5. The contents of location 5 (4) are copied into D4.
    h. The effective address is 4 + 4 + 2 = 10. To the contents of this location (12) are added the contents of D0(0)
    i. This instruction copies the contents of location 8 (0) to D0.

15. a. Should use a `MOVEA.L` instruction to load an address register. However, note that many assemblers will accept this instruction.
    b. Cannot use a destination register as a destination address with `LEA`.
    c. The source operand for an `EOR` instruction must be a data register; for example, `EOR Di, <ea>`.
    d. The destination for a `MOVEA` must be an address register.
    e. Literal data must be in range 1 to 8 for `ADDQ`.
    f. Bye operations are not allowed on the contents of address registers.
    g. The only permitted addressing modes for `MOVEP` are `MOVEP Di,d(Aj)` and `MOVEP d(Aj),Di`.
    h. `SWAP` operates only on data registers.
    i. PC relative addressing is not allowed for destination operands.
    j. Byte operations not permitted on the contents of an address register. Also, the literal operand range for an `ADDQ` instruction is 1 to 8.
    k. Wrong if “FC” is assumed to be a hexadecimal literal (it should be $FC). Ok if FC is a symbolic name.
    l. The `EXG` instruction exchanges the entire 32 bit contents of two registers. Byte exchanges are not permitted.
    m. `LEA` does not permit an autoincrementing source address.

n. `UNLK` requires an address register as an operand (e.g., `UNLK A6`).
o. `ANDI` means `AND` immediate and the source must be a literal; for example `ANDI #4, D5`.
p. `NOT` takes only a single operand.
q. The immediate operand range for shift instructions is only 1 to 8.
r. `RTS` has no extension.
s. A `.B` extension has no meaning with a `BRA` instruction. `BRA.S` indicates a short branch.
t. `MOVEQ` operates on a longword destination.
u. The `DIVU` instruction may not specify an address register as either a destination or a source operand.
v. `CMPM` permits only the `(Ai)+, (Ay)+` addressing modes.
w. `CL`:R cannot be applied to an address register.
x. `CMPM` permits only the `(Ai)+, (Ay)+` addressing modes.
y. `LEA` is a longword operation only. Note also that the source operand does not take a "#".
z. `DIVU` does not permit a byte operand. Also, the destination operand should be a data register.

23.

Pseudocode version

```
Set destination pointer to $2000
REPEAT
    Get byte
    Store it at pointer
    Increment pointer
UNTIL byte = 0
Set source pointer to $2000
Get byte at pointer
Increment pointer
WHILE (byte is not 0)
    IF (byte mod 2) = 0 THEN Print it
    Get byte at pointer
    Increment pointer
END WHILE
```

6800 Assembly version

```
      ORG      $400
      LEA      $2000, A0      A0 is the pointer to the text
Next  BSR      IN_CHAR        Get byte
      MOVE.B   D0,(A0)+       Store byte, increment pointer
      BNE      Next           Repeat until byte = 0
      LEA      $2000,A0       Reset pointer in A0
      MOVE.B   (A0)+,D0       Get a byte, increment pointer
Next1 BEQ      Exit           If zero THEN exit (end of list)
      BTST.B   #0,D0          WHILE: Test bit 0 of byte
      BNE      Next2          If not 0, don't print
      BSR      OUT_CHAR       If zero, byte even; print
Next2 MOVE.B   (A0)+,D0       Get next byte; increment pointer
      BRA      Next1
Exit
```

3.
```
        ORG     $400
        LEA     $1500,A7        Set up the stack pointer
        PEA     A               Push address of variable A
        LEA     -2(A7),A7       Make room for the result
        BSR     ADDABC          Call the subroutine
        LEA     4(A7),A7        Clean up the stack
        STOP    #$2700
ADDABC  MOVEM.L A0/D0, -(A7)    Save working registers A0 and D0
        MOVE.L  14(A7),A0       Get address of A
        MOVE.W  (A0),D0         Get value of A
        ADD.W   2(A0),D0        Add B
        MOVE.W  D0,12(A7)       Put result on the stack
        MOVEM.L (A7)+, A0/D0    Restore working registers
        RTS
        ORG     $1000           Data area
A       DC.W    9               Dummy A
B       DC.W    5               Dummy B
        END $400
```

11.
```
        ORG     $400
        LEA     $2000,A7            Set up initial stack pointer
        LEA     $A6A6A6A6, A6       Set up dummy initial A6 to help tracing
        MOVE.W  X,-(A7)             Push the value of x on the stack
        PEA     Y                   Push address of y on the stack
        BSR     Calc                Call subroutine
        LEA     (6,A7),A7           Clean up stack
        STOP    #$2700
Calc    LINK    A6, #-8             Create a stack frame for two longwords
        MOVEM.L A0/D0, -(A7)        Save working registers A0 and D0
        CLR.L   D0                  Clear D0 to use .W and .L operations
        MOVE.W  (12,A6),D0          Copy value of x to D0
        MOVE.L  D0,(-4,A6)          Save x in stack frame
        MULU    D0,D0               Calculate x^2
        MOVE.L  D0,(-8,A6)          Save x^2 in stack frame
        MULU    D0,D0               Calculate x^4
        ADD.L   (-8,A6),D0          Calculate x^4 + x^2
        ADD.L   (-4,A6),D0          Calculate x^4 + x^2 +x
        MOVEA.L (8,A6),A0           Pull address of result off stack
        MOVE.L  D0,(A0)             Pass result to calling program
        MOVEM.L (A7)+, A0/D0        Restore working registers
        UNLK    A6                  Collapse the stack frame
        RTS
        ORG     $1000               Put the data here
X       DC.W    4                   Provide a dummy value for x
Y       DS.L    1                   Reserve a long word for the result
        END     $400
```