EECS 388: Computer Systems and Assembly Language

Homework 2 - Solution

Justify your answers!
Figure 1 shows a part of the memory (both contents and locations).

Contents	Locations
$20	$4000
$50	$4001
$01	$4002
:	:
:	:
$B5	$5000
$CD	$5001

Figure 1.

Problem 1 (10 Points):

Consider the memory shown in Figure 1. What is in accumulator A and the N, Z, C, V bits in CCR after an LDD #$4001 instruction is executed?

Accumulator A _____$40____ N __0__ Z ___0__ C __??(unchanged)_ V __0__

LDD #$4001 →
This instruction will load register D (A:B) with the immediate value of $4001

D = (A:B) = $4001, therefore…
A = $40
B = $01

According to pg. 450 of the textbook the LDD instruction has an effect on the N and Z bits, it will zero out the V bit, and it has no effect on the C bit.

The N-bit is set when a number is negative in 2's complement representation. This means that this bit will be set if the most significant bit of D is set. The most significant nybble (4-bits) of D is 4, or 0100 in binary, so the most significant bit is 0, and thus the N-bit is also zero.

The Z-bit is set when the result of an operation is zero. In this case, Z will be zero due to the fact that the number being loaded into register D is not a zero.

Problem 2 (15 Points):

Write a program segment to reverse the bit order of a 6-bit number. Assume this number is stored in $6000. Store the reversed number to $6001 (i.e., if the original number is 0 0 b_5 b_4 b_3 b_2 b_1 b_0, after this program, the number in $6001 will be 0 0 b_0 b_1 b_2 b_3 b_4 b_5).

Reversing a number can be easily accomplished by rotating the number 1-bit at a time, until the desired rotation is accomplished. First, start the process by rotating the LSB out of the original number and into the carry register. Then rotate this number into the LSB of another register, and repeat this process. Each subsequent rotation will "push" the last rotation forward until all bits are in their correct places. This can be accomplished with a series of shifts, or with a loop.

```
        ORG $4000          ; Set program start address

        LDAA $6000         ; Load original number in register A
        LDAB #$00          ; Initialize register B

        LSRA               ; Shift LSB out of original number and into carry register
        ROLB               ; Rotate bit in carry register into LSB of register B

              ; Now register B = 0000000b0

        LSRA               ; Shift LSB out of original number and into carry register
        ROLB               ; Rotate bit in carry register into LSB of register B

              ; Now register B = 000000b0b1

        LSRA               ; Shift LSB out of original number and into carry register
        ROLB               ; Rotate bit in carry register into LSB of register B

              ; Now register B = 00000b0b1b2

        LSRA               ; Shift LSB out of original number and into carry register
        ROLB               ; Rotate bit in carry register into LSB of register B

              ; Now register B = 0000b0b1b2b3

        LSRA               ; Shift LSB out of original number and into carry register
        ROLB               ; Rotate bit in carry register into LSB of register B

              ; Now register B = 000b0b1b2b3b4

        LSRA               ; Shift LSB out of original number and into carry register
        ROLB               ; Rotate bit in carry register into LSB of register B

              ; Now register B = 00b0b1b2b3b4b5

        STAB $6001         ; Store the results and stop the program
        SWI
```

Problem 3 (20 points):

Write a program segment to multiply a 16-bit number in the D register by 10 using arithmetic left shift instead of multiplication instructions.

This problem can be solved using simple algebra techniques. Let the 16-bit number be called X:

10*X
= (8 + 2)*X
= (8*X) + (2*X)
= (arithmetic shift left X 3 times) + (arithmetic shift left X once)

```
ORG $4000              ; Set program start address

; Assume number is already loaded into D

; Calculate 2*X (one shift)
ASLD
STD $6000     ; Save 2*X

; Calculate 8*X (two more shifts shift)
ASLD
ASLD

; Now location $6000 contains 2*X and register D holds 8*X
; Just need to add the two numbers together

ADDD $6000   ; D = D + MEM[$6000]

SWI    ; Stop the program
```

Problem 4 (20 points):

Write a program to subtract two 24-bit numbers and store the result to memory locations starting at $6000. The two 24-bit numbers are stored in memory locations starting at $5000 and $5010, respectively.

The process of subtraction requires one to perform carrying if needed. Keeping track of carry bits is difficult, so an easier solution is to simply negate one of the numbers and add b/c algebraic rules tell us that X + Y is the same as X + (-Y).

The process of negating a number (in 2's complement) requires one to negate all of it's bits and add one. The HC12 does not have a simple negation operator (it does have a NEG operation which converts an 8-bit number to negative 2's complement, but this can't be used for 24-bit numbers!!) so instead we can use a trick. The trick is that a bit can be negated by using an exclusiveOR with the binary value ‘1’. Therefore a full 8-bit negation can be accomplished by exclusiveOR with $FF.

Not (X) = ExclusiveOR(X,$FF)

```
NEGATE_MASK         EQU     $FF     ; Create an equate for the negation mask
BASE1                       EQU     $5000   ; Base of 1st number
BASE2                       EQU     $5010   ; Base of 2nd number
NEGATED_BASE        EQU     $5020   ; Base of negated version of 2nd number
RESULT_BASE                 EQU     $6000   ; Base of result

        ORG $4000               ; Set program start address

        ; Convert the second number to negative in 2's complement

        ; First Negate the 2nd number 1 byte at a time
        LDAA BASE2+0                    ; Read in byte
        EORA #NEGATE_MASK+0             ; Negate it
        STAA NEGATED_BASE+0             ; Store the result

        LDAA BASE2+1                    ; Do this for the remaining 2 bytes
        EORA #NEGATE_MASK
        STAA NEGATED_BASE+1

        LDAA BASE2+2
        EORA #NEGATE_MASK
        STAA NEGATED_BASE+2

        ; Now perform 2's complement addition
        ; NUM1 - NUM2 = NUM1 + (Negated NUM2) + 1

        ; Pre-load the carry bit with 1
        LDAA #$01       ; Load a 1 into A, and shift the 1 into the carry reg
        LSRA

        ; Now begin the addition starting with the least-signficant byte

        LDAA BASE1+2                    ; Load in byte of NUM1
        ADCA NEGATED_BASE+2             ; Add negated byte with carry
        STAA RESULT_BASE+2              ; Use carry (with carry pre-set to 1)

        LDAA BASE1+1                    ; Continue for remaining bytes
        ADCA NEGATED_BASE+1
        STAA RESULT_BASE+1              ; Use carry again

        LDAA BASE1+0
        ADCA NEGATED_BASE+0
        STAA RESULT_BASE+0              ; Use carry again

        SWI ; Stop the program
```

Problem 5 (20 Points):

If A contains $56, what is the result of each of the following instructions? Assume that A is restored to its original value before each instruction.

a) ANDA #$33

A = $56 = 0101_0110
Mask = $33 = 0011_0011
Result(AND) = 0001_0010 = $12

A = $12

b) ORAA #$33

A = $56 = 0101_0110
Mask = $33 = 0011_0011
Result(OR) = 0111_0111 = $77

A = $77

c) EORA #$33

A = $56 = 0101_0110
Mask = $33 = 0011_0011
Result(EOR) = 0110_0101 = $65

A = $65

d) BITA #$80

The BITA operation performs arithmetic, but does not alter the contents of register A. However, it does alter the contents of the CC register. Typically this instruction will be used for the sake of comparison before a branch instruction, so that CCR bits can be purposefully set w/o having to alter any of the values in registers.

Problem 6 (15 Points):

Consider the following program:

```
LDD #$F00D
STD $8100
BSET $8100, $44
BCLR $8101, $11
```

What numbers are in $8100 and $8101 at the end?

LDD #$F00D
This instruction loads register D (A:B) with the immediate value $F00D.

D = $F00D $8100 = ?? $8101 = ??

STD $8100
This instruction writes the contents of D, which are $F00D, to memory location $8100

D = $F00D $8100 = $F0 $8101 = $0D

BSET $8100, $44
This instruction alters memory location $8100 by setting bits that are 1 in the mask.

$8100 = $FO = 1111_0000
Mask = $44 = 0100_0100
Result = 1111_0100 = $F4

D = $F00D $8100 = $F4 $8101 = $0D

BCLR $8101, $11
This instruction alters memory location $8101 by clearing bits that are 1 in the mask.

$8101 = $0D = 0000_1101
Mask = $11 = 0001_0001
Result = 0000_1100 = $0C

D = $F00D $8100 = $F4 $8101 = $0C