EE 308 – Homework 4

Due Feb. 16, 2009

1. Find the values of the N, Z, C, and V bits of the CCR register after execution of each of the following instructions, given that (A) = $AC and the condition flags are N=0, C=1, Z=0, and V=0. (Assume these are the values before each instruction starts e.g., do not use the flag state resulting from the instruction in part (a) as the initial state for part (b).)

   (a) `ADDA #$7A`
   (b) `ADCA #$F2`
   (c) `LSRA`
   (d) `ASRA`
   (e) `CMPA #$60`
   (f) `SUBA #$AC`

2. Suppose you started with the following register contents:
   `PC=2007 Y=7892 X=FF00 A=44 B=70 SP=1F7F`
   Wht address is in the stack pointer and registers, and exactly what is in the stack after the following instructions sequence is executed:

```
PSHA
PSHY
PSHB
PULX
JSR   $2007
```

3. Below are some data in the HC12 memory:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1000	D6	05	35	CF	E0	00	FE	08	20	A6	00	47	6A	05	08	53
1010	26	F7	34	C6	C8	CD	9C	40	03	26	FD	53	26	F7	3D	3F
1020	07	C2	3A	68	F3	09	C2	67	9A	0F	AA	55	08	40	CD	CF

Indicate the values in the registers after the HC12 executes the following instructions. Also write down the number of cycles needed to execute each instruction. Show what will be in the registers (in hex) after each of the instructions. If the instruction does not change a register, you may leave that entry blank. Note that the first instruction is located at address `0x2000`.

Instruction	D		X	Y	SP	N	Z	V	C	Addressing Mode	Effective Address
	A	B									
	0A	BB	1010	1020	0A00	1	0	1	0		
`lds #$1018`											
`cpd $1009`											
`puly`											
`asrb`											
`staa $1013`											
`adda 4,+x`											

4. Problem E4.1 from the text:

   Suppose that we have the following instruction sequence to be executed by the HCS12, what will be the contents of the topmost four bytes of the stack after the execution of these instructions?

```
lds     #$1500
ldaa    #$56
staa    1,-SP
ldab    #22
stab    1,-SP
ldy     #0
sty     2,-SP
```

5. Problem E4.10 from the text (modified):

   Draw the stack frame (the memory where the stack is located) and enter the value of each stack slot (if it is known) at the end of the following instruction sequence. Also, indicate the value of the stack pointer after execution of each instruction.

```
          org     $2000
          lds     #$2000
          leas    -2,sp     ; reserve 2 bytes for local variables
          clrb
          ldaa    #20
          psha
          ldaa    #$E0
          psha
          ldx     #$7000
          pshx
          jsr     sub_abc

...

sub_abc:  pshd
          leas      -4,sp  ; reserve 4 bytes for local variables
```

6. Write some assembly code to convert an eight-bit binary number to its Gray code equivalent. The procedure to do this is as follows: Exclusive OR the number with the same number shifted right by one bit. The number to convert is stored in a variable called `gray_code`. You should use a logical shift rather than an arithmetic shift.

7. Write some assembly code to generate the next pattern in the sequence for an eight-bit Johnson counter. The procedure to do this is as follows: Shift the present pattern to the right by one bit. The most significant bit of the next pattern is the inverse of the least significant bit of the present pattern. The present pattern is stored in a variable called `johnson_pattern`. After determining the next pattern, store this into the `johnson_pattern` variable.

8. Write some code to take the next entry out of a table, write it to `PORTB`, and update the index into the table. Here is an example of what the table might look like:

```
table_len: equ (table_end-table)

org data

table: dc.b    $00, $01, $02, $04, $08, $10, $20, $40, $80
table_end:
```

   Use a variable called `table_index` to hold the index of the pattern to write to PORTB. Your code should write the table entry corresponding to the index to PORTB. (For example, if `table_index` were 5, you would write the fifth element of the table, $10, to PORTB.) Then update `table_index` to point to the next element of the table. Make sure that `table_index` stays between 0 and `table_len - 1`.