Exam I Review February 2017

Binary Number Representations

Conversion of binary to hexadecimal and decimal.

Examples:

Convert binary number 1000 1101 to hexadecimal:

> Make groups of 4 bits to convert to hexadecimal, so binary number 1000 1101 can be represented as 8D in hexadecimal

Convert binary number 1000 1101 to unsigned decimal:

$$V = 1 \times 2^7 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 141$$

Convert binary number 1000 1101 to signed decimal:

> First, note that we are dealing with a negative number (MSB is 1), so take 2's complement of number, convert result to decimal, and then add the "-" sign.
>
> The 2's complement is 0111 0011. Now convert to decimal and add the sign, and the final result is -115.

Convert the signed decimal number -115 to binary:

> This is a negative so we should expect to have a "1" in the MSB. First convert the magnitude of the number to binary and then take 2's complement.
>
> 1. To convert a decimal number to binary we need to divide repeatedly by 2, and the reminder will be the binary number, and we get 0111 0011.
>
> 2. Take the 2's complement, and we get 1000 1101.

Convert the unsigned decimal number 115 to binary:

> Since it is unsigned, we do not have to worry about the sign being "0" or "1". Only divide by 2 consecutively to get the binary number, and do not take the 2's complement, and we will get 0111 0011.

Binary Number Addition and Subtraction (Using 2's complement)

Examples:

To add binary numbers, add bit-by-bit starting from the LSB:

```
   0001 1110
+  1001 1101
-----------------
   1011 1011
```

To add hexadecimal numbers, add digit-by-digit:

```
   1 E
+  9 D
----------------
   B B
```

Status of CCR when adding hex numbers:

C bit set when result does not fit in word
V bit set when P + P = N or N + N = P
N bit set when MSB of result is 1
Z bit set when result is 0

Example:

```
   1 E
+  9 D
----------------
   B B
```

C = 0, V = 0, N = 0, Z = 0

To subtract binary numbers, find the 2's complement of subtrahend and then add them:

```
   0001 1110
-  1001 1101
-----------------

   0001 1110
+  0110 0011
-----------------
   1000 0001
```

To subtract hex numbers, also find the 2's complement of subtrahend and then add them:

```
  1 E
-  9 D
----------------

   1 E
+  6 3
----------------
   8 1
```

Status of the CCR when subtracting hex numbers:

C bit set on borrow (when the magnitude of the subtrahend is greater than the minuend
V bit set when N - P = P or P - N = N
N bit set when MSB is 1
Z bit set when result is 0

Example:

```
   1 E
-  9 D
----------------
   8 1
```

C = 1, V = 1, N = 1, Z = 0

Disassembly of an HCS12 Program

For example, consider the hex code:

1000 C6 05 CE 20 00 18 06 3F

To determine the instructions use Table A-2 - A-6

1. $C6 corresponds to LDAB using IMM addressing mode.

    LDAB #$05

2. $CE corresponds to LDX using IMM addressing mode.

    LDX $#2000

3. $18 is not in Sheet 1 of Table A-2, and we need to go to Sheet 2 of Table A-2, to look up $06 code in that sheet.

    ABA

   The ABA operation uses INH addressing mode, so there is no operand for this command.

4. $3F corresponds to the SWI instruction to terminate the program.

Computation of Time of Execution

To compute how long a program takes to execute we need to count the total number of cycles it takes to execute a segment of code. We can also find this information in Table A-2 – A-6.

Example:

Cycles	Instructions
[1]	LDAB #$05
[2]	LDX $#2000
[2]	ABA
[9]	$3F

---

14 CPU cycles

We want to know how fast this program executes. We also know that a CPU cycle takes 41.7 ns to execute, therefore:

14 cycles × 41.7 ns/cycle = 583.8 ns

Addressing Modes

We are able to tell the effective address by knowing the addressing mode.

INH Inherent
IMM Immediate
DIR Direct
EXT Extended
IDX Indexed (won't study indirect indexed mode)
REL Relative (used only with branch instructions)

The Inherent (INH) addressing mode is used by instructions which work only with registers inside ALU

Examples:

```
ABA    ; Add B to A
CLRA ; clear A
```

The HC12 does not access memory

The Extended (EXT) addressing mode is used by instructions which give the 16−bit address to be accessed

Examples:

```
LDAA $1000 ; Load accumulator A with contents of address $1000
             Effective address $1000

LDX $1001    ; Load index register X with contents of address $1001:$1002
             Effective address $1001
```

Effective address is specified by the two bytes following op code

The Direct (DIR) addressing mode is used by instructions which give 8 LSB of address

Examples:

LDAA $20 ; Load accumulator A with content from address $0020
Effective address $0020

STX $21 ; Load index register with content of address $0021:$0022
Effective Address: $0021

Effective address is specified by byte following op code

The Immediate (IMM) addressing mode is used as part of instruction

Examples:

LDAA #$17 ; Move $17 into accumulator A
Effective Address: PC + 1

ADDA #10 ; Add to accumulator A a 10
Effective Address: PC + 1

Effective address is the address following the op code

The Indexed (IDX, IDX1, IDX2) addressing mode is used by instructions in which the effective address is obtained from X or Y register (or SP or PC):

Examples:

LDAA 0,X ; Use (X) as address to get value to put in A
Effective address: contents of X

ADDA 5,Y ; Use (Y) + 5 as address to get value to add to
Effective address: contents of Y + 5

The Stack and the Stack Pointer

The stack is a region of memory can be used for temporary storage by the microcontroller and/or the user.

To save information onto the stack use a push (the SP decrements).

To retrieve information from the stack use a pull (the SP increments).

The very first instruction in a program that uses the stack should be an LDS.

Example:

```
        org $2000     ; program starts at $2000
        lds #$2000    ; initialize stack at $2000
        ldaa #$2E     ; loads acca with $2E
        ldx #$1254    ; loads reg X with $1234
        psha          ; decrements SP by 1, pushes A into the stack
        pshx          ; decrements SP by 2, pushes X into the stack
        jsr sub       ; decrements SP by 2, saves return address into
                      ; stack, jumps to subroutine
        pulx          ; pulls reg X from stack, increments SP by 2
        pula          ; pulls reg A from stack, increments SP by 1
        swi           ; terminates program
sub:    clra          ; clears A (A=$00)
        ldx #$FFF     ; loads reg X with $FFFF
        rts           ;  return address is pulled off the stack, and SP increments by 2
```

When the program finishes executing (executes the SWI instruction), we will have the following in the registers:

A = $2E

X = $1254

SP = $2000