Microcomputers I – CE 320

Mohammad Ghamari, Ph.D.
Electrical and Computer Engineering
Kettering University

Lecture10: Basic Arithmetic Instructions

Announcement

- HW-Exercises 3 is already uploaded on blackboard.

- You are going to have your quiz no.3 on Thursday, Nov 9.
    - Focus is on all lecture materials as well as **homework exercise 4**.

Today’s Topics

- Review Addition and Subtraction
- Use Multiple Precision arithmetic to add and subtract large numbers.
- Practice writing assembly programs.

Programs to do Arithmetic

Example1: Add 3 memory location bytes. Store result in memory.

- Write a program to add the numbers stored at memory locations $800, $801, and $802, and store the sum at memory location $900.

Programs to do Arithmetic

Example1: Add 3 memory location bytes. Store result in memory.

- Write a program to add the numbers stored at memory locations $800, $801, and $802, and store the sum at memory location $900.

Ans:

```
org   $1000    ; starting address of the program
ldaa  $800     ; place the contents of the memory location $800 into A
adda  $801     ; add the contents of the memory location $801 into A
adda  $802     ; add the contents of the memory location $802 into A
staa  $900     ; store the sum at the memory location $900
end
```

Programs to do Arithmetic

Example2: Add 2 memory locations and subtract a third. Store result in memory.

- Write a program to subtract the contents of the memory location at $805 from the sum of the memory locations at $800 and $802, and store the result at the memory location $900.

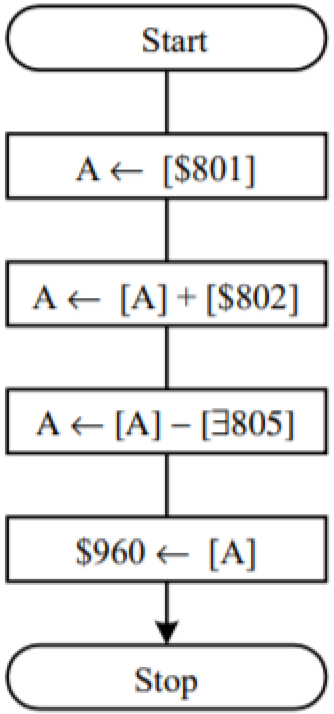
Programs to do Arithmetic

Example2: Add 2 memory locations and subtract a third. Store result in memory.

- Write a program to subtract the contents of the memory location at $805 from the sum of the memory locations at $800 and $802, and store the result at the memory location $900.

Ans:

```
org     $1000    ; starting address of the program
ldaa    $800     ; copy the contents of the memory location at $800 to A
adda    $802     ; add the contents of memory location at $802 to A
suba    $805     ; subtract the contents of memory location at $805 from A
staa    $900     ; store the contents of accumulator A to $805
end
```



Programs to do Arithmetic

Example3: Subtract a constant from memory locations.

- Write a program to subtract 5 from four memory locations at $800, $801, $802, and $803.

Programs to do Arithmetic

Example3: Subtract a constant from memory locations.

- Write a program to subtract 5 from four memory locations at $800, $801, $802, and $803.

Ans:

```
org     $1000
ldaa    $800            ; copy the contents of memory location $800 to A
suba    #5              ; subtract 5 from A
staa    $800            ; store the result back to memory location $800
ldaa    $801
suba    #5
staa    $801
ldaa    $802
suba    #5
staa    $802
ldaa    $803
suba    #5
staa    $803
end
```

Programs to do Arithmetic

Example4: Add 2 words in memory. Store result in memory.

Write a program to add two 16-bit numbers that are stored at \$800-\$801 and \$802-\$803, and store the sum at \$900-\$901.

Programs to do Arithmetic

Example4: Add 2 words in memory. Store result in memory.

Write a program to add two 16-bit numbers that are stored at $800-$801 and $802-$803, and store the sum at $900-$901.

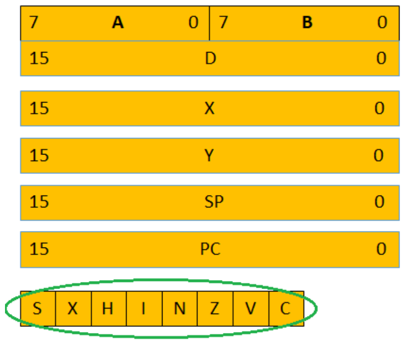
Ans:

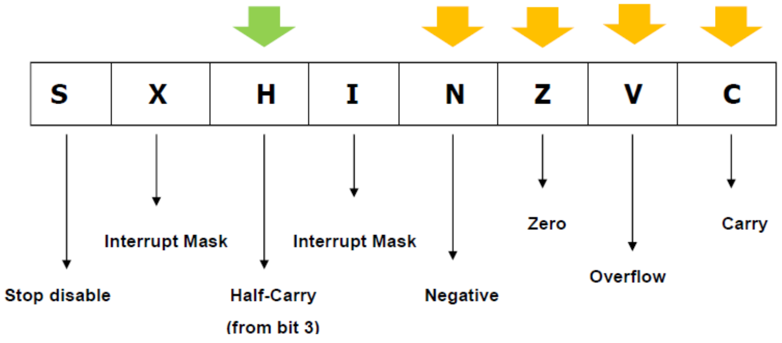
```
org     $1000
ldd     $800        ; place the 16-bit number at $800~$801 in D
addd    $802        ; add the 16-bit number at $802~$803 to D
std     $900        ; save the sum at $900~$901
end
```

Multi-Precision Arithmetic

- Programs can also be written to add numbers larger than 16 bits.
- **Multi-precision arithmetic:** Arithmetic performed in a 16-bit microprocessor on numbers that are larger than 16 bits is called multi-precision arithmetic.
    - Makes use of the carry flag (C flag) of the condition code register (CCR).
    - Bit 0 of the CCR register is the C flag. It can be thought of as a temporary 9th bit that is appended to any 8-bit register or 17th bit that is appended to any 16-bit register.

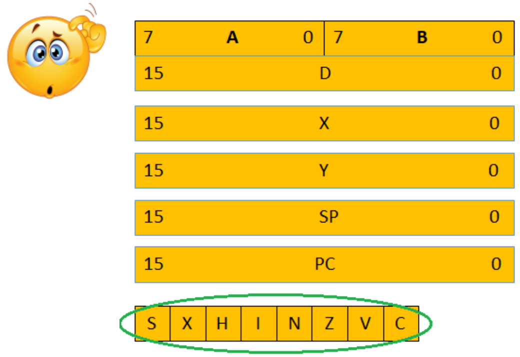
HCS12 CPU Registers





Multi-Precision Arithmetic

Example:



- Consider the following two instructions:

ldd	**#$8645**
addd	**#$9978**

These two instructions add the numbers $8645 and $9978.

SUM

$\$8645$
$+\$9978$
Carry $\$11FBD$

- The result is $11FBD, **a 17-bit number**, which is too large to fit into the 16-bit double accumulator D.
- When the HCS12 executes these two instructions:
  - The lower sixteen bits of the answer, **$1FBD**, are placed in double accumulator D. This part of the answer is called the **sum**.
  - The leftmost bit is called a **carry**.
    - A carry of 1 following an addition instruction sets the C flag of the CCR register to 1.
    - A carry of 0 following an addition clears the C flag to 0.

Addition and Subtraction
From Lecture 8

- 8 bit addition
  - ABA: (A) + (B) → A; Note that there is no AAB instruction!
  - ADDA: (A) + (M) → A
    - ADDA $1000
  - ADDB: (B) + (M) → B
    - ADDB #10
  - ADCA: (A) + (M) + C → A
  - ADCB: (B) + (M) + C → B
- 8 bit subtraction
  - SBA: (A) – (B) → A; Subtract B from A (Note: not SAB instruction!)
  - SUBA: (A) – (M) → A; Subtract M from A
  - SUBB: (B) – (M) → B
  - SBCA: (A) – (M) – C → A
  - SBCB: (B) – (M) – C → B
- 16 bit addition and subtraction
  - ADDD: (A:B) + (M:M+1) → A:B
  - SUBD: (A:B) – (M:M+1) → A:B
  - ABX: (B) + (X) → X
  - ABY: (B) + (Y) → Y

We will use ADCA(B) and SBCA(B) to do multi-precision addition or subtraction.

There is a pattern that make you be easy to remember the instructions!!!

1. The last letter in these instructions is the destination!
2. Also it comes to the first in the operation

Precision?

- The term **precision** is often used to refer to **the size of a unit of data** manipulated by the processor.

- **Single-precision** refers to instructions that manipulate **one byte** at a time.
  - ADDA, ADDB, ABA, SUBA, SUBB, SBA

- **Double-precision** refers to **two-byte** operation.
  - ADDD, SUBD
  - ABX: (B) + (X) → X, ABY: (B) + (Y) → Y

- **Multi-precision**
  - Adding and subtracting **numbers longer than single precision** introduce **an issue**.
  - **Carries** and **borrows** need to **propagate** through a number.

Multi-Precision Addition

Since the sum of the most significant digit has a sum greater than 16, it generates a carry that must be added to the next more significant digit, causing the C flag to be set to 1.

Example1:

```
ldaa    #$1A
adca    #$76
staa    $800
```

```
ldaa    #$59
adca    #$54
staa    $801
```

```
   1   1  1
  $1A 59 8183
+ $76 54 8290
  $90 AE 0413
```

```
ldd     #$8183
addd    #$8290
std     $802
```

- Multi-precision addition is performed one byte at a time, beginning with the least significant byte.
- The HCS12 does allow us to add **16-bit numbers** at a time because it has the **addd** instruction.
  - Two instructions can be used to add the **least significant 16-bit numbers together**:

    ldd #$8183

    addd #$8290
  - Then, the contents of double accumulator D must be saved before the higher bytes are added:

    std $802
  - When the **second-to-most significant bytes** are added, the carry from the lower byte must be added in order to obtain the correct sum.
    - Thus, we need an "add with carry" instruction (**ADCA** instruction for accumulator A).
    - The instructions for adding the **second-to-most-significant bytes** are:

      ldaa #$59

      adca #$54
    - We also need to save the **second-to-most-significant byte** of the result: **staa $801**
    - **Most significant bytes** can be added using similar instructions as the **second-to-most-significant byte.**

Multi-Precision Addition

Since the sum of the most significant digit has a sum greater than 16, it generates a carry that must be added to the next more significant digit, causing the C flag to be set to 1.

Example1:

```
ldaa    #$1A
adca    #$76
staa    $800
```

```
ldaa    #$59
adca    #$54
staa    $801
```

```
  1   1  1
  $1A59 8183
+ $7654 8290
  ----------
  $90AE 0413
```

```
ldd     #$8183
addd    #$8290
std     $802
```

```
ldd     #$8183      ; place the lowest two bytes of the first number in D
addd    #$8290      ; add the lowest two bytes of the second number to D
std     $802        ; store the lowest two bytes of the sum at $802-$803
ldaa    #$59        ; place the second-to-most significant byte of the first number in A
adca    #$54        ; add the second-to-most-significant byte of the second number and carry to A
staa    $801        ; store the second-to-most-significant byte of the sum at $801
ldaa    #$1A        ; place the most-significant byte of the first number in A
adca    #$76        ; add the most-significant byte of the second number and carry to A
staa    $800        ; store the most significant byte of the sum end
```

Multi-Precision Addition

Example2: Write a program to add two 4-byte numbers that are stored at \$1000-\$1003 and \$1004-\$1007, and store the sum at \$1010-\$1013.

The addition starts from the LSB and proceeds toward MSB.

```
org   $1500

; Add and save the least significant two bytes
ldd   $1002       ; D ← [$1002, $1003]
addd  $1006       ; D ← [D] + [$1006, $1007]
std   $1012       ; m[$1012, $1013] ← [D]

; Add and save the second most significant bytes
ldaa  $1001       ; A ← [$1001]
adca  $1005       ; A ← [A] + [$1005] + C
staa  $1011       ; $1011 ← [A]

; Add and save the most significant bytes
ldaa  $1000       ; A ← [$1000]
adca  $1004       ; A ← [A] + [$1004] +C
staa  $1010       ; $1010 ← [A]
```

C

std and ldaa do not change the carry so C is the carry resulted from addd \$1006

Notice there is no instruction for addition with carry for 16 bits.

Multi-Precision Subtraction

Example1:

```
ldaa    #$98
sbca    #$16
staa    $800
```

```
ldaa    #$76
sbca    #$75
staa    $801
```

16+5

5

$$
\begin{array}{r}
\$98765432 \\
-\$16757284 \\
\hline
\$8200E1AE
\end{array}
$$

Since a larger number is subtracted from a smaller one, there is a need to borrow from the higher byte, causing the C flag to be set to 1.

```
ldd     #$5432
subd    #$7284
std     $802
```

- Multi-precision subtraction also is performed one byte at a time, beginning with the least significant byte.
- The HCS12 does allow us to add **16-bit numbers** at a time because it has the **SUBD** instruction.
  - Two instructions can be used to subtract the **least significant two bytes of the subtrahend from the minuend**:

    ldd #$5432

    subd #$7284
  - Then, the contents of double accumulator D must be saved before the higher bytes are subtracted:

    std $802
  - When the **second-to-most significant bytes** are subtracted, the borrow 1 has to be subtracted from **second-to-most significant byte** of the result.
    - Thus, we need an "subtract with borrow" instruction (**SBCA** instruction for accumulator A).
    - The instructions to subtract the **second-to-most-significant bytes** are:

      ldaa #$76

      sbca #$75
    - We also need to save the **second-to-most-significant byte** of the result: **staa $801**
    - **Most significant bytes** can be subtracted using similar instructions as the **second-to-most-significant byte.**

Multi-Precision Subtraction

Since a larger number is subtracted from a smaller one, there is a need to borrow from the higher byte, causing the C flag to be set to 1.

Example1:

```
ldaa    #$98
sbca    #$16
staa    $800
```

```
ldaa    #$76
sbca    #$75
staa    $801
```

16+5

5

$$
\begin{array}{r}
\$98765432 \\
-\$16757284 \\
\hline
\$8200E1AE
\end{array}
$$

```
ldd     #$5432
subd    #$7284
std     $802
```

```
org     $1000       ; starting address of the program
ldd     #$5432      ; place the lower two bytes of the minuend in D
subd    #$7284      ; subtract the lower bytes of the subtrahend from D
std     $802        ; save the lower two bytes of the difference
ldaa    #$76        ; place the second-to-most-significant byte of the minuend in A
sbca    #$75        ; subtract the second-to-most-significant byte of the ; subtrahend and the borrow from A
staa    $801        ; save the second-to-most-significant byte of the difference
ldaa    #$98        ; put the most-significant-byte of the minuend in A
sbca    #$16        ; subtract the most-significant-byte of the ; subtrahend and the borrow from A
staa    $800        ; save the most-significant-byte of the difference end
```

Multi-Precision Subtraction

Example2: Write a program to subtract the 4-byte number stored at $1004-$1007 from the number stored at $1000-$1003, and save the difference at $1010-$1013.

The subtraction Addition starts from the LSB and proceeds toward MSB.

```
org   $1500
; Subtract and save the least significant two bytes
ldd   $1002        ; D ← [$1002, $1003]
subd  $1006        ; D ← [D] - [$1006, $1007]
std   $1012        ; m[$1012, $1013] ← [D]

; Subtract and save the second most significant bytes
ldaa  $1001        ; A ← [$1001]
sbca  $1005        ; A ← [A] - [$1005] - C
staa  $1011        ; $1001 ← [A]

; Add and save the most significant bytes
ldaa  $1000        ; A ← [$1000]
sbca $1004         ; A ← [A] - [$1004] - C
staa  $1010        ; $1010 ← [A]
```

Only these instructions have changed comparing to last slide’s example.

There is no instruction for subtraction with borrow for 16 bits.

Multi-Precision Addition

Example

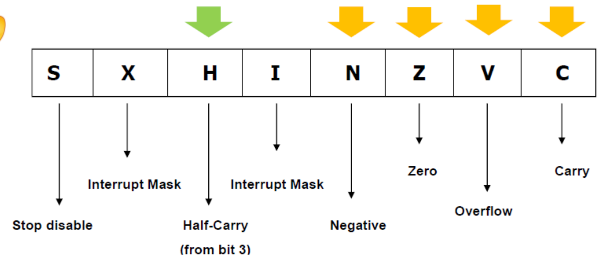
- Adding two quadruple-precision numbers.
- Multi-precision addition is performed one byte at a time, beginning with the least significant byte.
    - $92FF45B7_{16}$
    - $6D325D88_{16}$

```
 1    1         1
    92   FF   45   B7
    6D   32   5D   88
+   ------------------
    00   31   A3   3F
```

```
        ORG     $1200
num1    DC.B    $92, $FF, $45, $B7
num2    DC.B    $6D, $32, $5D, $88
ans     DS.B    4

        ORG     $2000
        LDAA    num1+3      ; 1
        ADDA    num2+3      ; 2
        STAA    ans+3       ; 3
        LDAA    num1+2      ; 4
        ADCA    num2+2      ; 5
        STAA    ans+2       ; 6
        LDAA    num1+1      ; 7
        ADCA    num2+1      ; 8
        STAA    ans+1       ; 9
        LDAA    num1        ; 10
        ADCA    num2        ; 11
        STAA    ans         ; 12
        SWI                 ; 13
```

Program Trace




```
        ORG     $1200
num1    DC.B    $92,$FF,$45,$B7
num2    DC.B    $6D,$32,$5D,$88
ans     DS.B    4

        ORG     $2000
        LDAA    num1+3    ; 1
        ADDA    num2+3    ; 2
        STAA    ans+3     ; 3
        LDAA    num1+2    ; 4
        ADCA    num2+2    ; 5
        STAA    ans+2     ; 6
        LDAA    num1+1    ; 7
        ADCA    num2+1    ; 8
        STAA    ans+1     ; 9
        LDAA    num1      ; 10
        ADCA    num2      ; 11
        STAA    ans       ; 12
        SWI               ; 13
```

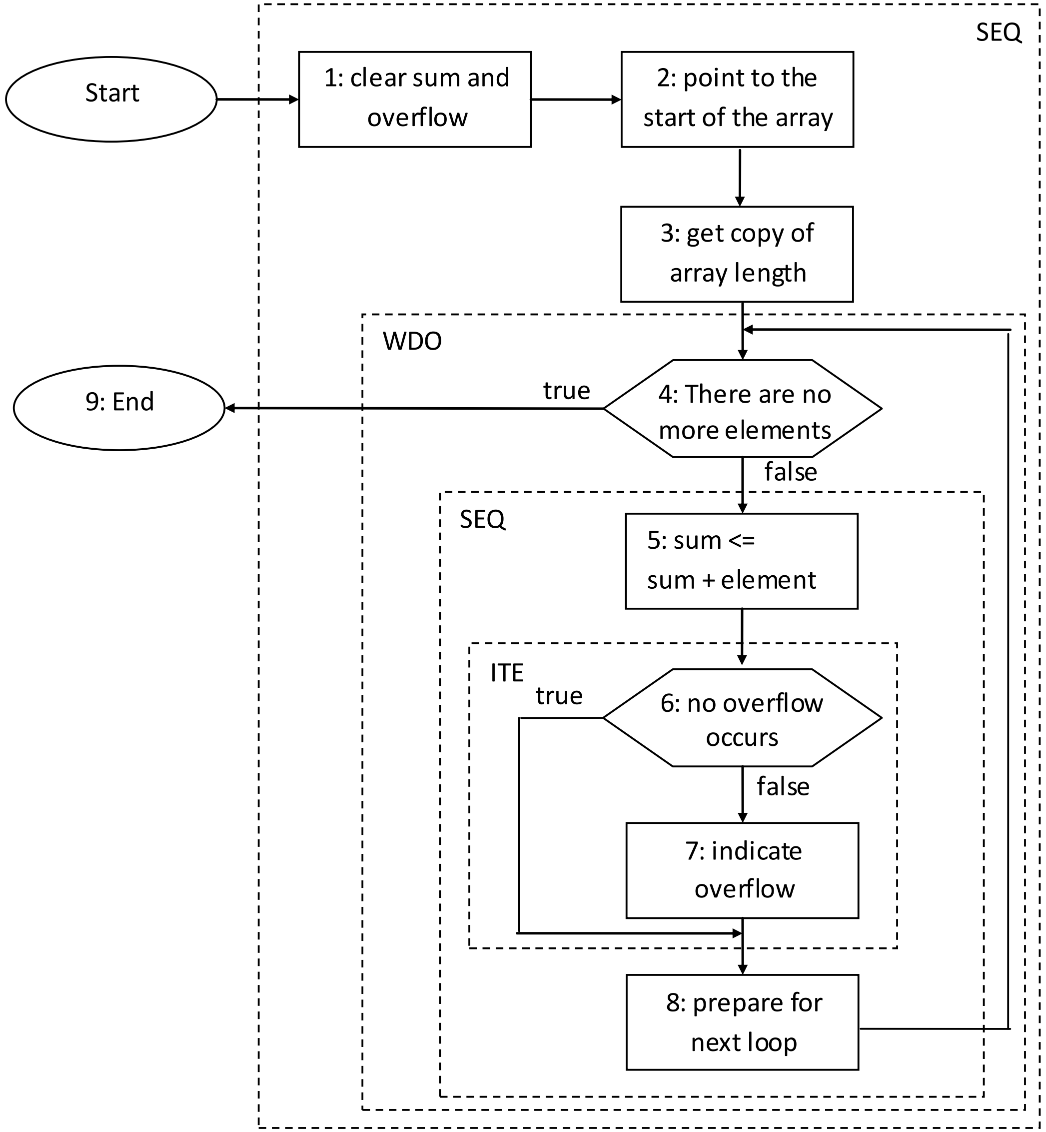
Trace	Line	PC	A	N	Z	V	C
1	1	2003	B7	1	0	0	-
2	2	2006	3F	0	0	1	1
3	3	2009	3F	0	0	0	1
4	4	200C	45	0	0	0	1
5	5	200F	A3	1	0	1	0
6	6	2012	A3	1	0	0	0
7	7	2015	FF	1	0	0	0
8	8	2018	31	0	0	0	0
9	9	201B	31	0	0	0	1
10	10	201E	92	1	0	0	1
11	11	2021	00	0	1	0	1
12	12	2022	00	0	1	0	1
13	13	2024	-	-	-	-	-

Homework Example

- Calculate a **two-byte** sum of an array of one-byte **unsigned** numbers.

- Requirements
  - Variable ***ovflow*** should be $00 if the sum is valid. Otherwise, $ff.
  - The address of the array of one-byte unsigned integers is supplied at $1030.
  - The length of the array is a one-byte value supplied in $1032.
  - ***Ovflow*** must be assigned to address $1040.
  - The sum is returned in locations $1041 and $1042.

Homework Solution: Flowchart



```
;--------------------------------------------------------------------
; variable/data section
                org     $1030
array   ds.w    1       ; address of the array
length  ds.b    1       ; length of the array

                org     $1040
ovflow  ds.b    1       ; overflow flag. $00 = valid, $ff = invalid
sum             ds.w    1       ; 2-byte sim of unsigned numbers in the array

;--------------------------------------------------------------------
; code section
                org     $2000
                movw #0,sum             ; 1. clear sum
                movb    #0,ovflow       ;    clear ovflow
                ldd     #0                      ; clear A and B
                ldx     array           ; 2. point to the start of the array
                ldab    length
                tfr     D,Y                     ; 3. get copy of array length
loop            beq     done                    ; 4. no more elements?
                clra
                ldab    0,X                     ; load an element to B
                addd    sum                     ; 5. sum = sum + element
                std     sum                     ; store D to sum
                bcc     sum_ok          ; 6. no overflow?
                movb    #$ff,ovflow   ; 7. indicate overflow
sum_ok  inx                     ; 8. prepare for next loop
                dey                             ;    "
                bra     loop                    ; go to "loop"
done    swi
```

Homework: Changes for Two-Byte Length

- How likely is unsigned overflow in the original program?
  - Cannot happen. The largest possible sum is $FE01 ($FF * $FF).

- What modifications are needed to handle two-byte length?
  - Replace **DS.B 1** with **DS.W 1**
  - Replace **LDAB, TFR** with **LDY**

```
;-----------------------------------------------------------------------
; variable/data section
                        org         $1030
array       ds.w        1           ; address of the array
length      ds.w        1           ; length of the array

                        org         $1040
ovflow      ds.b        1           ; overflow flag. $00 = valid, $ff = invalid
sum                     ds.w        1           ; 2-byte sim of unsigned numbers in the array

;-----------------------------------------------------------------------
; code section
                        org         $2000
                        movw #0,sum                 ; 1. clear sum
                        movb        #0,ovflow       ;    clear ovflow
                        ldd         #0                          ; clear A and B
                        ldx         array           ; 2. point to the start of the array
;                       ldab        length
;                       tfr         D,Y                         ; 3. get copy of array length
                        ldy         length          ; 3. get copy of array length
loop                    beq         done                        ; 4. no more elements?
                        clra
                        ldab        0,X                         ; load an element to B
                        addd        sum                         ; 5. sum = sum + element
                        std         sum                         ; store D to sum
                        bcc         sum_ok          ; 6. no overflow?
                        movb        #$ff,ovflow ; 7. indicate overflow
sum_ok      inx                                 ; 8. prepare for next loop
                        dey                                     ;    "
                        bra         loop                        ; go to "loop"
done        swi
```

Homework: Changes for Signed Numbers

```
;-----------------------------------------------------------------------
; program
                    org      $2000
                    movw #0,sum                 ; 1. clear sum
                    movb     #0,ovflow          ;    clear ovflow
                    ldd      #0                          ; clear A and B
                    ldx      array              ; 2. point to the start of the array
                    ldab     length
                    tfr      D,Y                         ; 3. get copy of array length
loop                beq      done                        ; 4. no more elements?
                    clra
                    ldab     0,X                         ; load an element to B
                    bpl      skip                        ; check if B is positive
                    ldaa     #$ff                        ; extend the sign bit if B is negative
skip
                    addd     sum                         ; 5. sum = sum + element
                    bvc      sum_ok             ; 6. no overflow?
                    movb     #$ff,ovflow ; 7. indicate overflow
sum_ok
                    ; std clears the v bit so std is moved to here
                    std      sum                         ; store D to sum
                    inx                                  ; 8. prepare for next loop
                    dey                                  ;    "
                    bra      loop                        ; go to "loop"
done        swi
```

Questions?

Wrap-up

What we’ve learned

- **Multiple-precision** arithmetic to add and subtract large numbers.

- More practice writing programs in assembly

What to Come

- Advanced arithmetic instructions

- Boolean logic instructions