Lecture 10: Basic Arithmetic Instructions
Today’s Goals

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

Adding two quadruple-precision numbers

- Multi-precision addition is performed one byte at a time, beginning with the least significant byte.

<table>
<thead>
<tr>
<th>92FF45B7_16</th>
<th>6D325D88_16</th>
</tr>
</thead>
</table>

1 1 1 | 92FF45B7 | 6D325D88 |

+ | --------------- |

00 31 A3 3F

ORG $1200
num1 DC.B $92, $FF, $45, $B7
num2 DC.B $6D, $32, $5D, $88
ans DS.B4

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.B4

<table>
<thead>
<tr>
<th>Trace</th>
<th>Line</th>
<th>PC</th>
<th>A</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2003</td>
<td>B7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2006</td>
<td>3F</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2009</td>
<td>3F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>200C</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>200F</td>
<td>A3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2012</td>
<td>A3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>2015</td>
<td>FF</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2018</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>201B</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>201E</td>
<td>92</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>2021</td>
<td>00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>2022</td>
<td>00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>2024</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Another Example

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

• Requirements
  - Variable **overflow** 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 $1032$.
  - **Overflow** must be assigned to address $1040$.
  - The sum is returned in locations $1041$ and $1042$.
Start

1: clear sum and overflow

2: point to the start of the array

3: get copy of array length

4: There are no more elements

5: sum <= sum + element

6: no overflow occurs

7: indicate overflow

8: prepare for next loop

9: End

false

true

WDO

SEQ

SEQ

ITE
; 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 sum of unsigned numbers in the array

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

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
inxx ; 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