Lecture 8:
Assembly Language

Today's Topics

- Review the concept of memories and registers (accumulators)
- Generating machine code manually.
  - You are expected to convert assembly code lines to machine codes.
- Files and processes associated with converting assembly source code to machine code
- To learn assembler directives

A Short Story

- World is converging again.
  - No pure EE, CE, and CS.
- Fundamentals are always important.
- A short story about hiring interviews.
  - Very simple but fundamental questions are asked even to Senior engineers.
- Grade or GPA is important but…
- Last but not least, you are expected to study outside classroom. Lectures cannot cover all topics.

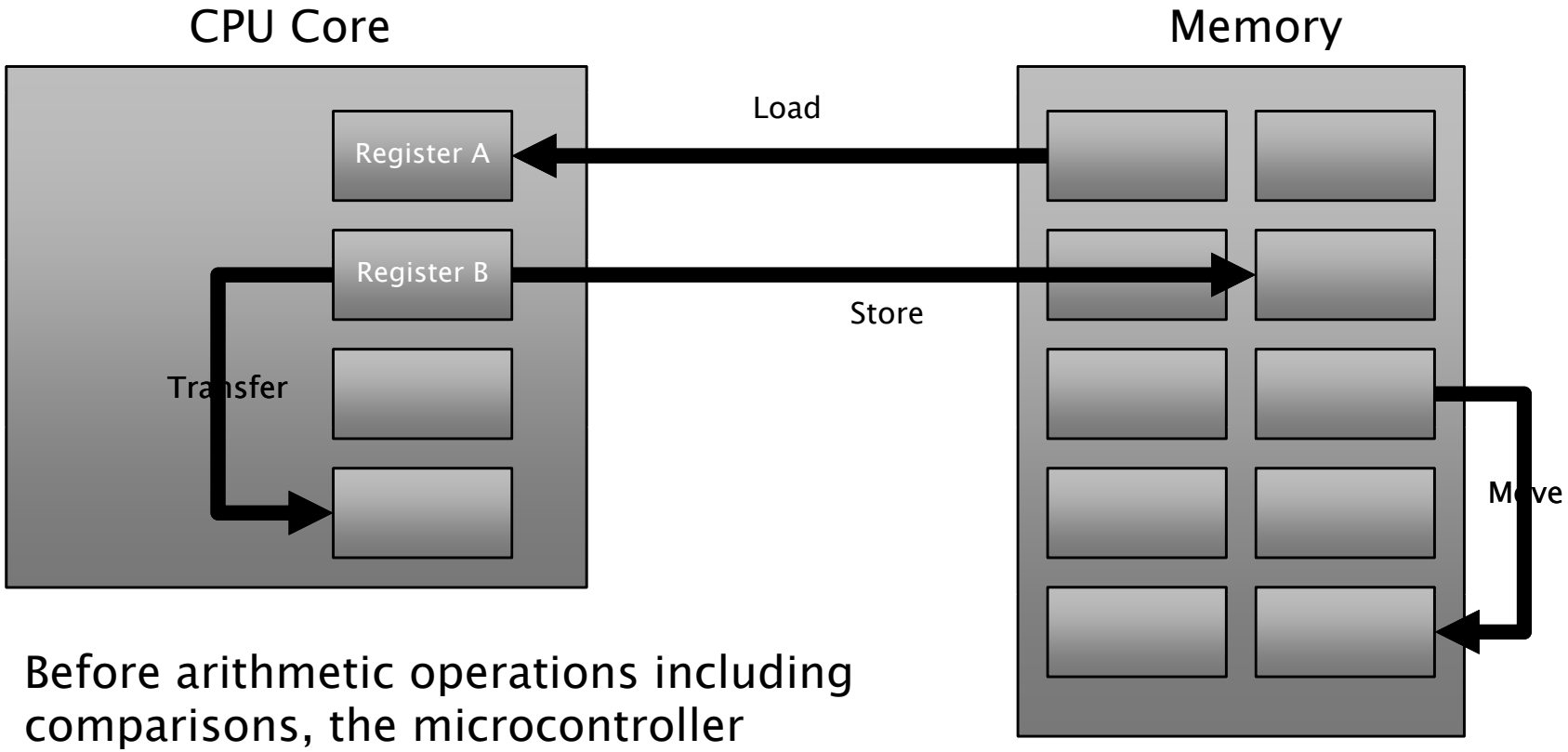
Microcontrollers (or Microcomputers)

Basic ideas

- Microcontroller
  - CPU core + I/O ports + Memories (RAM and ROM) + …
- Memories
  - We only use RAM area to learn assembly language and test programs.
  - No need to worry about burning your program into ROM.
- Registers (accumulators) vs. memories
  - Registers are small read/write memory cells inside CPU core.
  - Memories are located outside CPU.
  - To get a value from a location in a memory, the value should travel through data bus. (Remember memory modules are separated from CPU core)
    - This takes time (much longer than getting from/setting to Registers)

Microcontrollers (or Microcomputers)

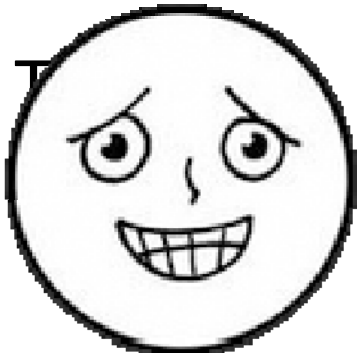
Load and Store / Move / Transfer

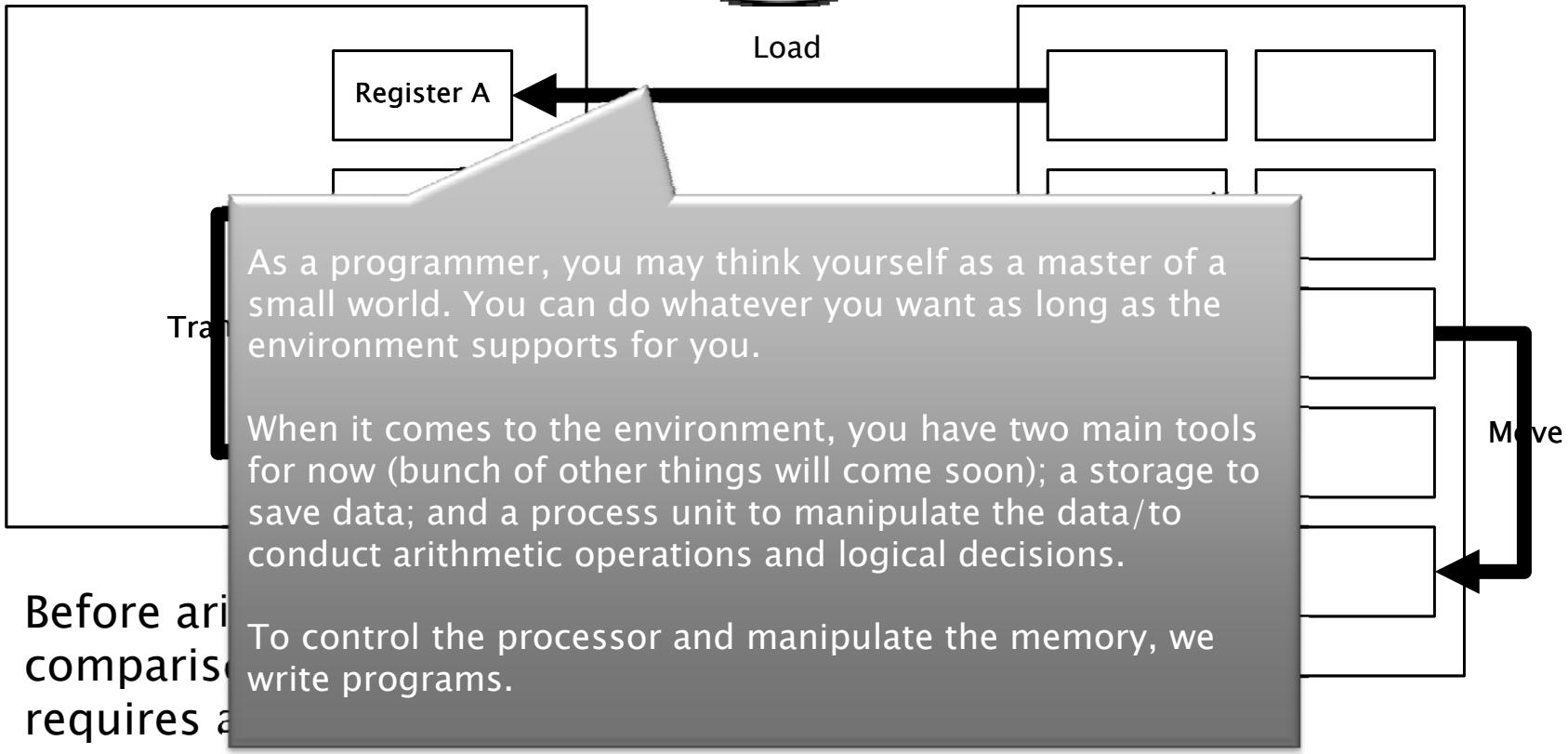


Before arithmetic operations including comparisons, the microcontroller requires a value on a register to do the operations.

Microcontrollers (or Microcomputers)

Load and Store / Move / T





As a programmer, you may think yourself as a master of a small world. You can do whatever you want as long as the environment supports for you.

When it comes to the environment, you have two main tools for now (bunch of other things will come soon); a storage to save data; and a process unit to manipulate the data/to conduct arithmetic operations and logical decisions.

To control the processor and manipulate the memory, we write programs.

Before ari
comparis
requires a
operations.

Code Line and Program Counter

- When we say **Code Line** in Lab assignments, quizzes, and exams, the instruction line is completed (executed). So registers are supposed to be affected by the execution.
- **Program Counter** always points the NEXT instruction!!
  - Caution on Branches. PC depends on whether the branch is taken or not.
- Example:

```
1:    1500    CE 2000          LDX #$2000
2:    1503    180B FF 1000     MOVB #$FF,$1000
3:    1508    C6 02            LDAB #2
4:    150A    27 0E            BEQ 14
5:    150C    A6 00            LDAA 0,X
6:    150E    B1 1000          CMPA $1000
7:    1511    24 03            BHS 3
8:    1513    7A 1000          STAA $1000
9:    1516    08               INX
10:   1517    53               DECB
11:   1518    20 F0            BRA -16
12:   151A    3F               SWI
```

Trace	Line	PC	A	B	X	N	Z	V	C
1	1	1503	-	-	2000	0	0	0	-
2	2	1508	-	-	2000	0	0	0	-
3	3	150A	-	02	2000	0	0	0	-
4	4	150C	-	02	2000	0	0	0	-
5	5	150E	40	02	2000	0	0	0	-
6	6	1511	40	02	2000	0	0	0	1
7	7	**1513**	40	02	2000	0	0	0	1
8	8	1516	40	02	2000	0	0	0	1
9	9	1517	40	02	2001	0	0	0	1
10	10	1518	40	01	2001	0	0	0	1

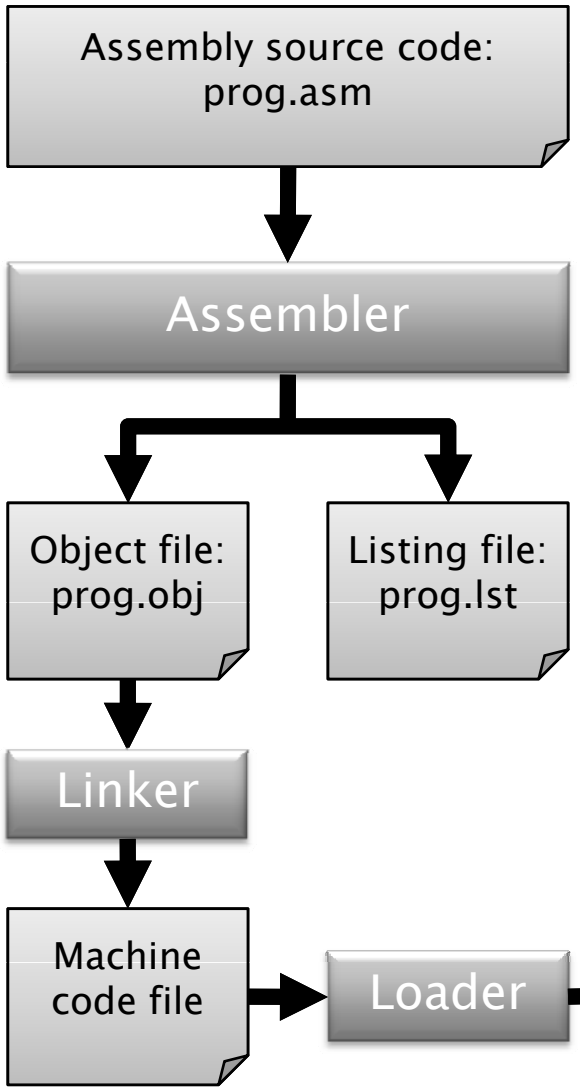
Machine Code

Manually generate machine code*

Address	Machine Code	Source Code
1500	CE 2000	LDX #$2000
1503	180B FF 1000	MOVB #$FF, $1000
1508	C6 02	LDAB #2
150A	27 0E	BEQ 14
150C	A6 00	LDAA 0,X
150E	B1 1000	CMPA $1000
1511	24 03	BHS 3
1513	7A 1000	STAA $1000
1516	08	INX
1517	53	DECB
1518	20 F0	BRA -16
151A	3F	SWI

Assembly Process

General case



ASCII text file
.asm extension

Cross-assembler:
Translate assembly code into object code.

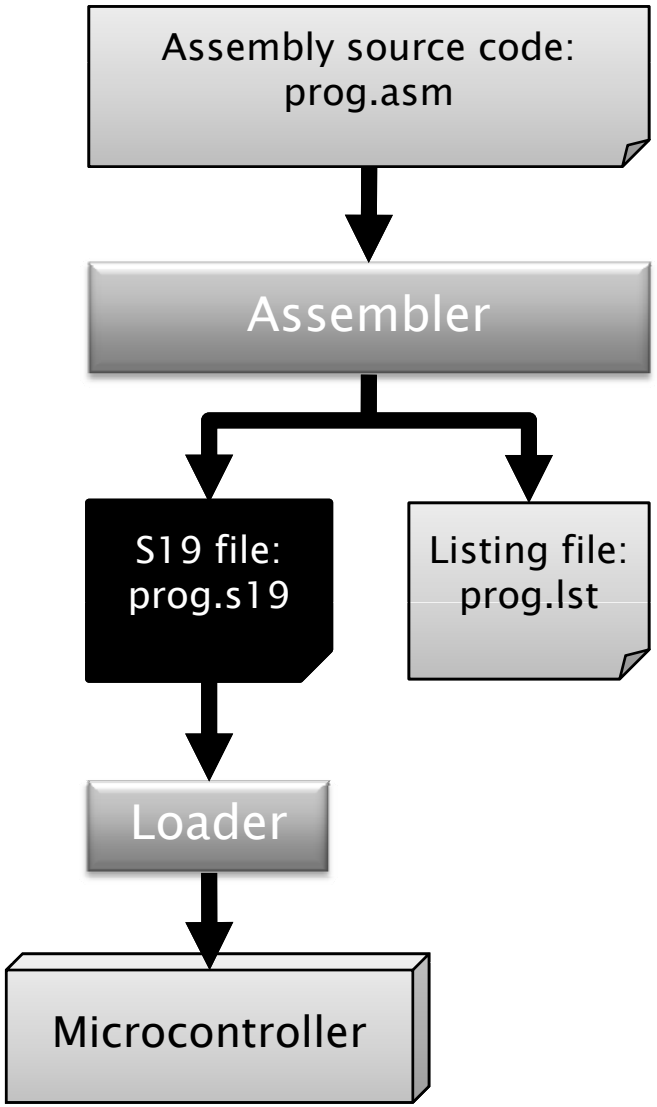
Object code:
Mix of machine code and additional information

Listing file:
Human readable log file containing the original assembly and the machine code.

Linker:
Combine multiple object files into a single piece of machine code

Assembly Process

In the lab



ASCII text file
.asm extension

Cross-assembler:
Translate assembly code into object code.

Listing file:
Human readable log file containing the original assembly and the machine code.

S19 file: Motorola S-record format
An ASCII encoding for binary data.
http://en.wikipedia.org/wiki/S19_(file_format)

Loader:
D-Bug12 is used to load .s19 file to the memory of the microcontroller. **Load** instruction is used.

Proper Assembly Code

- 1. Separate the source code into constant section, data and variable sections, and code section.
- 2. Do not use numbers within the code
  - Except for possibly 0 or 1 in obvious situation
- Always begin with a comment block stating
  - Purpose of the program
  - Inputs
  - Outputs
  - Programmer
  - Anything else useful
- Comment within the code
- Assume that a reader understands the processor's assembly code, so do not use comments to simply rephrase the assembly code.

Assembly Language Syntax

- The assembly language consists of lines of text in the form:
  - [label:] [command] [operand(s)] [;comment]

    or
  - ; comment

  - where ':' indicates the end of label and ';' defines the start of a comment. The *command* field may be an instruction, a directive or a macro call.

Assembler Directives

Only small fractions…

- Memory allocation
  - **org**: puts an address into the assembler location counter
    - org $1000
  - **ds.[size]** (Define Storage)
    ; allocate specified number of storage spaces
    ds.b (Define Storage Bytes)
    ds.w (Define Storage Words)
    - buffer ds.b 100
    - dbuffer ds.w 100

- Data formation
  - **dc[.size]** (Define Constant)
    ; allocate and initialize storage for variables
    dc.b (Define Constant Byte)
    dc.w (Define Constant Word)
    - array dc.b $11, $12,$22
    - Initialize a 3-byte constant with the data

- Note: need to know a difference between
  - buffer ds.b $10
    - allocate 16 byte memory space for ‘buffer’
  - buffer dc.b $10
    - allocate 1 byte for ‘buffer’ and the value of ‘buffer’ will be initialized to $10

Assembler Directives

- Symbol Definition
  - equ
    - TRUE equ $FF
- Numbers
  - $xx or xxh: hexadecimal
  - %xxxx.. : binary
  - Otherwise: decimal

Some More Instructions

Load Effective Address Instructions

- Load effective address instructions
  - LEAX: Load effective address into X
    - LEAX 10,X
  - LEAY: Load effective address into Y
    - LEAY B, Y
  - LEAS: Load effective address into SP
    - LEAS 0,PC
- Can you tell a difference between “LEAX 10,X” and “LDX 10,X”?
  - Assuming (X) = 1200, the content at 120A is 34h, and at 120B is 56h
    - “LEAX 10,X” makes X be 120A
    - “LDX 10,X” makes X be the content at memory (120A+120B) which is 3456.

Some More Instructions

Addition and Subtraction

- 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

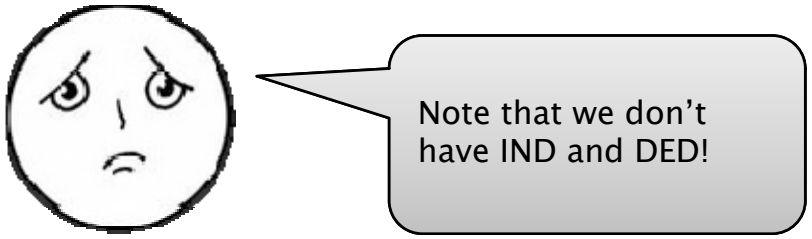
There is a pattern that make you be easy to remember the instructions!!!

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

Some More Instructions

Increments, Decrements, and Negate

- Increments
  - INC: (M) + 1 → M
  - INCA: (A) + 1 → A
  - INCB
  - INS
  - INX
  - INY



- Decrements
  - DEC
  - DECA
  - DECB
  - DES
  - DEX
  - DEY

- Negate
  - NEG: negate a memory byte
  - NEGA
  - NEGB

Assembly Code Example

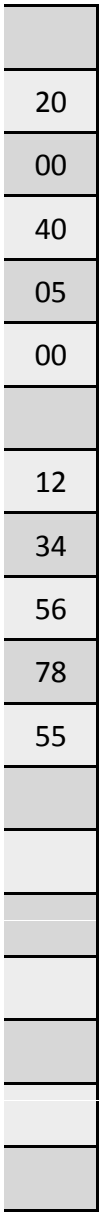
Requirements

- Write a program to copy a table of one-byte values.
- Our table will be defined by a starting address, supplied at $1000, and by a one-byte number of elements in the table, supplied at $1002.
- The table will be copied a given distance from the original table, and this two-byte offset will be supplied at address $1003.

Just one example

```
 1:          =00001000                          ORG    $1000
 2:       1000 +0002                 table       ds.w   1
 3:       1002 +0001                 length      ds.b   1
 4:       1003 +0002                 offset      ds.w   1
 5:
 6:          =00001800                          ORG    $1800
 7:       1800 FE 1000                          LDX    table
 8:       1803 B7 54                            TFR    X,D
 9:       1805 F3 1003                          ADDD   offset
10:       1808 B7 46                            TFR    D,Y
11:       180A F6 1002                          LDAB   length
12:       180D 27 09                 loop       BEQ    done
13:       180F 180A 00 40                       MOVB   0,X,0,Y
14:       1813 08                               INX
15:       1814 02                               INY
16:       1815 53                               DECB
17:       1816 20 F5                            BRA    loop
18:       1818 3F                     done      SWI

Symbols:
done                           *00001818
length                         *00001002
loop                                 *0000180d
offset                         *00001003
table                          *00001000
```

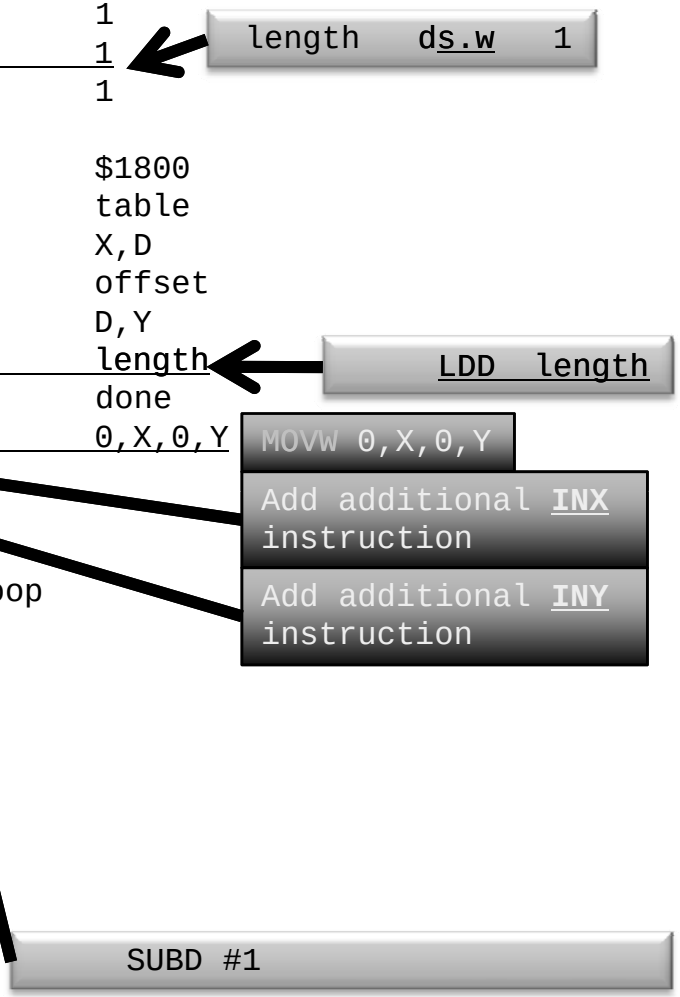


Modification of the Example

- What changes are required to handle a table of two-byte numbers?
  - Need to copy two bytes instead of one byte.
- What changes are required to handle a two-byte length?
  - The length should represent two-byte numbers!

```
 1:           =00001000                          ORG         $1000
 2:       1000 +0002                 table       ds.w        1
 3:       1002 +0001                 length      ds.b        1
 4:       1003 +0002                 offset      ds.w        1
 5:
 6:           =00001800                          ORG         $1800
 7:       1800 FE 1000                           LDX         table
 8:       1803 B7 54                             TFR         X,D
 9:       1805 F3 1003                           ADDD        offset
10:      1808 B7 46                             TFR         D,Y
11:      180A F6 1002                           LDAB        length
12:      180D 27 09                 loop        BEQ         done
13:      180F 180A 00 40                        MOVB        0,X,0,Y
14:      1813 08                                INX
15:      1814 02                                INY
16:      1815 53                                DECB
17:      1816 20 F5                             BRA         loop
18:      1818 3F                      done      SWI

Symbols:
done                            *00001818
length                          *00001002
loop                                  *0000180d
offset                          *00001003
table                           *00001000
```



Questions?

Wrap-up

What we've learned

- Registers and memories
- Generating machine code manually.
- Concept of assembly language
- Assembler directives

What to Come

- Flowcharts
- Some assembly programming examples