Assembler Syntax

- An assembler takes a source code file and builds a memory image (binary) executable file
  - Specifies the location in memory (either relative or absolute) of data and instructions
- In Coldfire assembler each line of the assembly program may be one of three possible options
  - Comment
  - Instruction
  - Assembler Directive

Comments

- In Codewarrior for CF an entire line can be marked as a comment by starting it with an asterisk (*) character or the C-style comments of // or /* */:
- Example:

```
* This line will be ignored by the assembler
// This line will be ignored too
/* As will this one */
        MOVE.L  D2,D3
        ADD.L   (A0),D3
        ...
```

Instructions

- In Codewarrior each instruction is written on a separate line and has the following syntax:

  (Label:) Instruc. Op. Operands Comment

- Example:

  START: MOVE.L D3,D2 ; Initialize D2

- Notes:
  - Label is optional and is a text identifier for the address where the instruction is placed in memory. (These are normally used to identify the target of a branch or jump instruction.)
  - Everything after the semicolon (;) will be treated as a comment

Labels

- Labels are text placeholders for a location in your code
- The optional label in front of an instruction or other directive evaluates to the address where the instruction or data starts in memory and can be used in other instructions
- Can be any alphanumeric string (or start with a '.' or '_') but should be terminated with a colon ':'

```
            .text
START:      MOVE.W      #3,D0
.L1:        MOVE.B      #0xFF,D3
            BRA         .L1
```

Assembly Source File

START =	0x1000	**MOVE.W**
	0x1002	**0003**
.L1 =	0x1004	**MOVE.B**
	0x1006	**00FF**
	0x1008	**BRA**

Assembler finds what address each instruction starts at…

Note: The BRA instruc. causes the program to branch (jump) to the instruction at the specified address

```
            .text
START:      MOVE.W      #3,D0
.L1:        MOVE.B      #0xFF,D3
            BRA         0x1004
```

…and replaces the labels with their corresponding address

Assembler Directives

- Direct the assembler in how and where to…
  - Assemble the actual instructions
  - Initialize memory before executing the first instruction of the program
- Similar to pre-processor statements and global variable declarations in C/C++/Java
  - #DEFINE, etc.
  - int x = 1;

Assembler Directive Overview

Directive	Description
.text, .data	Section directives for program code or data variables
.org	Indicates where to place the following code/variables in memory
.equ / .set	Defines a constant replacement value (like #define in C)
.long, .short, .byte	Defines memory storage for the given size variable and specifies the initial value(s)
.ascii, .asciz	Defines memory storage for ASCII text strings (.asciz places a null terminator/zero after the string)
.align	Starts the following instructions/data at an address that is a multiple of a certain power of 2 (i.e. 1, 2, 4, 8)
.space	Leaves space for the specified number of bytes
.extern	“Imports” the specified label from some other file
.global	“Exports” the specified label name making it available to other file to be imported
.include	Includes the specified assembly file into the current file

Section Directives

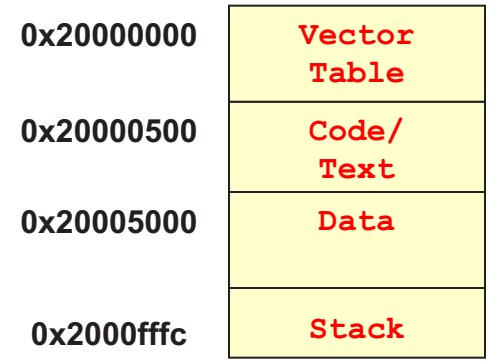
- An assembler file is broken into sections by using appropriate directives
- The assembler starts each section at different locations in memory (e.g. .text => address 0x20000500)

Directive	Meaning
.text	All instructions / code must be in a text section
.data	Space for global and other variables can be allocated in this section (this section of memory will be initialized to 0's by the OS/loader before the program starts

```
            .data
myvar:      .long   1,-1
x:          .short   1
            .text
.main       move.l   x,d0
            ...
            .data
mystr:      .asciz 'Hi World\r\n'
```

Note: Multiple .text / .data sections are fine

52259 RAM Address Space:



.org

- .org directive tells the assembler the following instructions or data values should be placed starting at an the given address offset from the section base address
- .org's can be used to space out data or function definitions but can be avoided if desired
- Format: .org *addr_offset*

Assume the .text section starts at 0x20000500

```
            .text
            .org        0x100
start:      MOVE.W      #3,D0
            ADD.W       D1,D0
            ...
```

start = 0x20000600	**MOVE.W**
0x602	**0003**
0x604	**ADD.W**

Assembler starts placing following lines at the specified address and continues sequentially until next .org

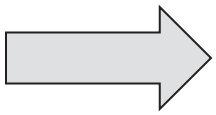
Equate (.equ) / Set (.set)

- Equates/sets a symbol to a value
  - Essentially a find and replace by the assembler (replaces any occurrence of symbol with value)
  - Similar to a #define in C/C++
  - .set's allow the symbol to be redefined/reset to another value later in the code while .equ symbols cannot be redefined
- Syntax:

  symbol .equ expression

  symbol .set expression
- Example:

```
ONE         .equ        1
NULL        .equ        0x00
MAX         .set        8
            .org        0x1000
            MOVE.L      #ONE,D0
            MOVE.B      #NULL,D3
            ...
            CMPI.L      #MAX,D0
MAX         .set        16
```



Symbolic names in .equ statements are replaced with their values when assembled.

```
ONE         .equ        1
NULL        .equ        0x00
MAX         .set        8
            .org        0x1000
            MOVE.L      #1,D0
            MOVE.B      #0x00,D3
            ...
            CMPI.L      #8,D0
MAX         .set        16
```

Data Directives

- Fills memory with specified size/type of data
  - Directives: .long, .short, .byte, .ascii, .asciz
  - .ascii places each character enclosed in double quotes in consecutive bytes in memory (.asciz terminates the string with a null (0) character)
- Format:

  (Label) .long val_0,val_1,…,val_n
- Each value in the comma separated list will be stored using the indicated size
  - Example: myarray: .short 0x1,2,3
    - Each value 1,2,3 is stored as a word/short
    - Label “myarray” evaluates to the start address of the first word (i.e. address of the 0x1 value)

.align

- Usually used with data directives to ensure following short/word and longword variables start at an address a multiple of the variable size
- Format:

  .align n

  – Next variable/instruction will start at an address a multiple of 2^n (see table below)

N	Alignment (address multiple)	Used when next variable is…
1	2	short / word
2	4	longword, FP single
3	8	FP double

Data Directives Example

```
        .data
C1:     .byte   0xFE,0x05
MSG:    .asciz  "SC\r\n"
        .align  1
DAT:    .short  1,2,-1
MSG2:   .ascii  "012"
        .align  2
VAR:    .long   0x12345678
```

.data + 0x00	**FE 05 53 43**
.data + 0x04	**0D 0A 00 00**
.data + 0x08	**00 01 00 02**
.data + 0x0c	**FF FF 30 31**
.data + 0x10	**32 00 00 00**
.data + 0x14	**12 34 56 78**

C1 = .data + 0x00
MSG = .data + 0x02
DAT = .data + 0x08
MSG2 = .data + 0x0e
VAR = .data + 0x14

- **Note:**
  - **Labels evaluate to the starting address of the first value in the list**
  - **\r = Carriage Return (CR) which will usually return cursor to first column of same line**
  - **\n = Line Feed (LF) which will usually move cursor to next line**
  - **The combination of \r \n moves cursor to beginning of next line.**
  - **Different OS'es use CR and LF differently: Linux = only LF, Windows = CR + LF**

.space

- Reserves the indicated number of bytes in memory and initialized values to 0's
- Format:

(Label:) .space num_bytes

```
        .data
const:  .long    -2
array:  .space   8
.L1:    .short   0xABCD
```

.data + 0x00	**FF FF FF FE**
.data + 0x04	**00 00 00 00**
.data + 0x08	**00 00 00 00**
.data + 0x0c	**AB CD 00 00**

const = .data + 0x00
array = .data + 0x04
.L1 = .data + 0x0c

.global & .extern

- .global defines the following label to be visible to other assembly files to import
- Format:

  .global label1 (, label2…)

```
        .filename "file1.s"
        .data
gvar:   .long   0x12345678
        .global gvar
        ...
        .text
main:   move.l  gvar,d0
        ...
```

- .extern tells the assembly the following label is defined in another file
- Format:

  .extern label1 (, label2…)

```
        .filename "file2.s"
        .data
        .extern gvar
        .text
main:   move.l  gvar,d1
        ...
```

.include

- Includes the specified file into the current file at the location of the .include directive
- Format

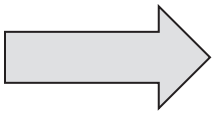
.include filename

```
            .data
            .align 2
            .long  0x1a, 0x2b
            .text
            .global f1
f1:         add ...
```

func_lib.s

```
            .data
            .long  5
            .include func_lib.s
            .text
            .global main
main:       move ...
```

myfile.s



```
            .data
            .long  5
            .data
            .align 2
            .long  0x1a, 0x2b
            .text
            .global f1
f1:         add ...
            .text
            .global main
main:       move ...
```

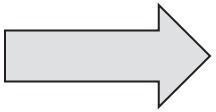
Seen by assembler

C/C++ and Data Directives

- Data directives are used to initialize or reserve space for global variables in C

```
short int count = 7;
char msg[15];
int table[8] = {0,1,2,3,4,5,6,7};

void main()
{
  ...
}
```

C/C++ style global declarations



```
        .data
count:  .short 7
msg:    .space 15
        .align 2
table:  .long  0,1,2,3,4,5,6,7

        .text
        .global main
main:   ...
```

Assembly equivalent

Another Directive Example

```
        .data   ( .data = 0x20003004)
C1:     .long    0x20003570,0x12345678
MSG:    .byte    -2,-4,-6
        .align   2
PTR:    .long    MSG-4
        .short    0xfe,0xc0
```

20003004	**20 00 35 70**
20003008	**12 34 56 78**
2000300c	**FE FC FA 00**
20003010	**20 00 30 08**
20003014	**00 FE 00 C0**
20003018	**00 00 00 00**

C1 = 0x20003004
MSG = 0x2000300c
PTR = 0x20003010

- **Note:**
  - **Labels evaluate to the starting address of the first value in the list**
  - **Values put into memory are always constants (can't read a value from another location)**
  - **Constants are 0 extended in assembler directives not sign extended**

An Assembly File Template

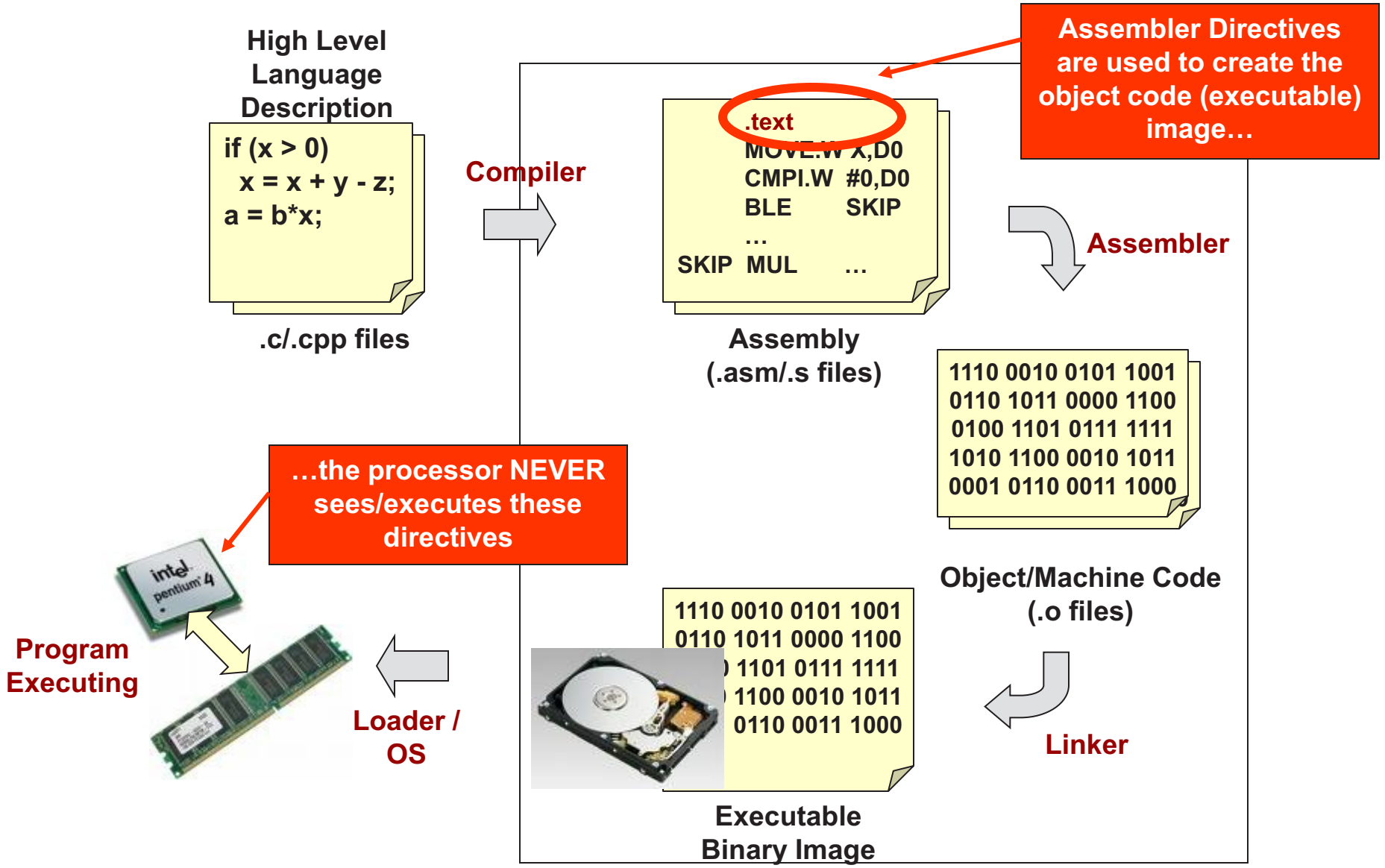
- Perform .set and .equ at the top of the file as well as any .extern declarations
- Start with a .data section to declare all static/global variables and constants
- Start a .text section with the main program
- Follow the main program with .text and .data sections for any subroutines

```
/* Internal and external definitions */
/*   such as .set, .equ, .extern, etc.
          .data
/* Variable allocation/declaration */

          .text
/* Program instructions */
main:     ...

          .data
/* Variables for subroutine 1 */
          .text
/* Subroutine 1 code */
          ...
```

Summary & Notes

- Assembler Directives:
  - Tell the assembler how to build the program memory image
    - Where instructions and data should be placed in memory when the program is loaded
    - How to initialize certain global variables
- Recall, a compiler/assembler simply outputs a memory **IMAGE** of the program. It must then be loaded into memory by the OS to be executed.
- Key: Directives are NOT instructions!
  - They are used by the assembler to create the memory image and then removed.
  - The Coldfire processor never sees these directives only actual instructions

Directives in the Software Flow



Assembly Process

- The assembly procedure of a file requires two passes over the code
  - 1st Pass: Build a symbol table by parsing through the file keeping track of where each data and instruction will be located in memory in an internal variable called the location counter (LC)
    - Symbol table maps:
      - Labels to corresponding addresses
      - Maps .equ/.set symbols to corresponding values
  - 2nd Pass: Substitute corresponding values for labels and symbols and then translate to machine code

Assembly Process Diagram