UMP OPEN COURSEWARE

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

Chapter 5

Program Development

Expected Outcomes

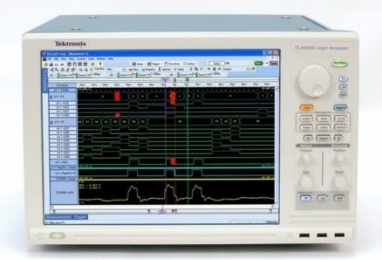
- Distinguish between various codes in the programming language
- Explain the role of assembler and compiler
- Distinguish between different data types
- Use directive assemblers for source code development

Development Tools

- **Software**
- **Translator**
  - Assemblers
  - Compiler
- **Linkers**
- **Debug Monitor**
- **Performance Analyzer**

- **Hardware**
- **In-Circuit Emulator**
  - Substitute for CPU chip
  - Allow to see 'inside' the CPU
- **Logic Analyzer**
- **Logic Probe**
- **Oscilloscope**
- **LED's**

Development Tools



Software Tools

- Programmer should know how to fully utilize instruction set to produce the efficient program
- **Assembler**
  - Translate source code program to object code program
- **Cross assembler**
  - Assemble the source code for a target system (CPU) using a computer with a different processor (CPU) than the target system
- **Compiler**
  - Program that converts a high level language source code into object code

Software Tools

- **Source code**
  - A computer program written using text information to indicate instructions
  - It is not executable and needs an assembler or compiler to translate it to produce machine code
- **Object code**
  - Sequence of binary numbers and other information that represents machine code
  - It is not executable form since the start address may be unknown and are linked to produce executable file

Source Code Compilation

- **Compilation procedure**
  - Write source program in text editor and save (`cuba.asm`)
  - Create binary file with assembler (`asm68k cuba.asm -l`)
    - -l creates list file (`cuba.lst`) with assembly and binary code
  - Linker is used to combine various object code files to produce a hex file
  - A hex file is not a machine code but represent the code using ASCII character
  - The most common are Intel hex and Motorola S-record
  - If using ASM68K assembler, it runs assembler and followed by a linker to produce a hex file directly (`cuba.s`)

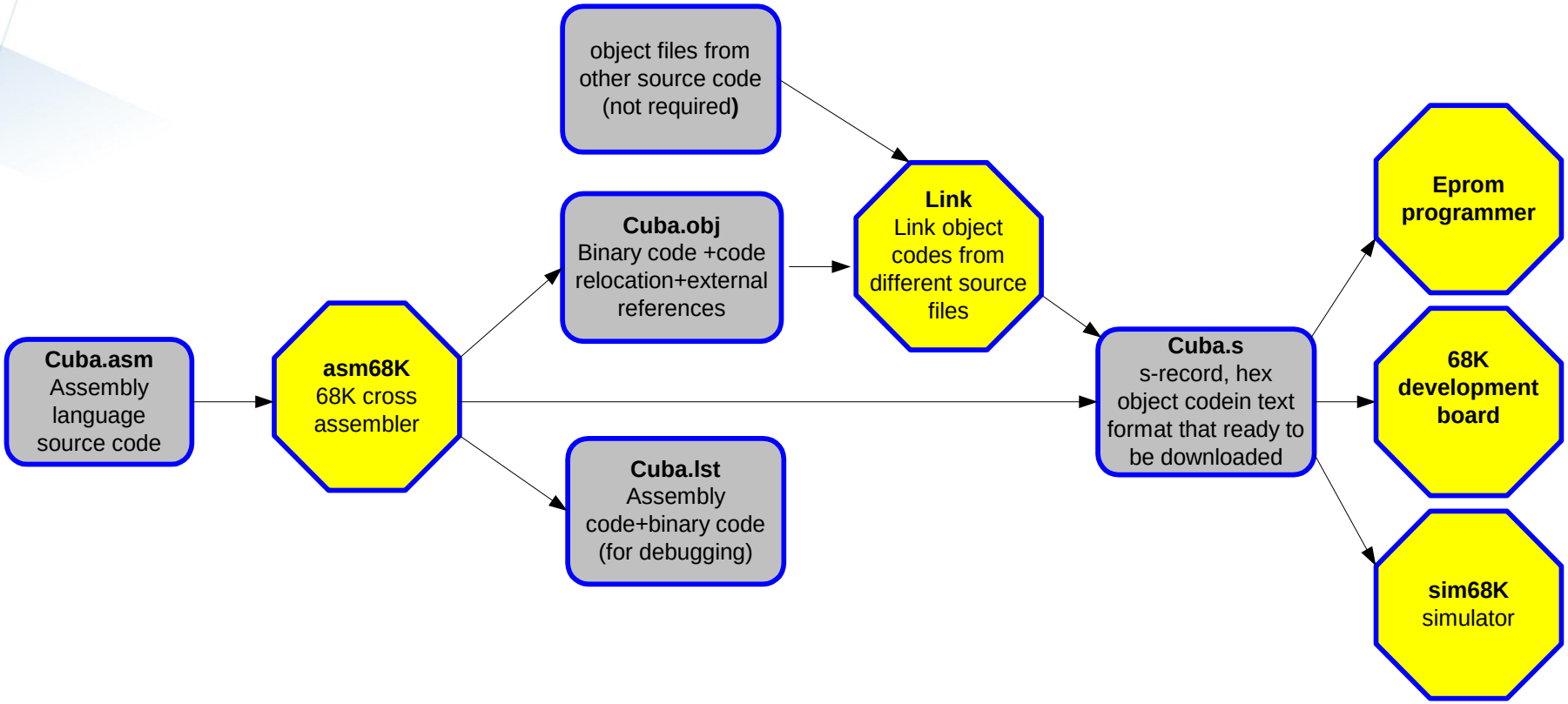
Source Code Compilation

- Execution procedure
  - Using simulator

    Use any simulator software and load the object file (`cuba.s`)
  - Using Eprom programmer

    Download the object file (`cuba.s`) into the Eprom

Source Code Compilation



Source Code

```
DRA         EQU     $E00000         ; Define the address of DRA, DDRA, CRA
DDRA        EQU     $E00000         ; DDRB, DRB, CRB and RAM
CRA         EQU     $E00002
DDRB        EQU     $E00004
DRB         EQU     $E00004
CRB         EQU     $E00006
RAM         EQU     $400000

            ORG     0
            DC.L    $404000         ; Define addressing for the RESET
            DC.L    START

            ORG     $400
START       MOVE.B  #0,CRA          ; Initialize  port A
            MOVE.B  #$FF,DDRA
            MOVE.B  #4,CRA
            MOVE.B  #0,CRB          ; Initialize port B
            MOVE.B  #$FF,DDRB
            MOVE.B  #4,CRB
            LEA     $2000,A0
LOOP        MOVE.B  (A0)+,DRA       ; Read port A
            MOVE.B  (A0)+,DRB       ; Read port B
            JSR     DELAY
            BRA     LOOP
DELAY        MOVE.L  #200000,D0     ; Delay subroutine
WAIT        SUB.L   #1,D0
            BNE     WAIT
            RTS
```

List File

```
MC68000 Cross Assembler
Copyright (C) Stephen Croll, 1991.  Author: Stephen Croll
Version 2.00 beta 1.02
00E00000                                1 DRA           EQU    $E00000   ; Define the address of DRA, DDRA, CRA
00E00000                                2 DDRA          EQU    $E00000  ; DDRB, DRB, CRB and RAM

00E00002                                3 CRA           EQU    $E00002
00E00004                                4 DDRB          EQU    $E00004
00E00004                                5 DRB           EQU    $E00004
00E00006                                6 CRB           EQU    $E00006
00400000                                7 RAM            EQU    $400000
00000000                                8               ORG    0
00000000  00404000                      9                DC.L   $404000 ; Define addressing for the RESET

00000004  00000400                      10              DC.L   START
                                        11
00000400                                12              ORG    $400
00000400  13FC 0000 00E00002            13 START        MOVE.B  #0,CRA
00000408  13FC 00FF 00E00000            14              MOVE.B  #$FF,DDRA               ; Initialize Port A
00000410  13FC 0004 00E00002            15              MOVE.B  #4,CRA
00000418  13FC 0000 00E00006            16              MOVE.B  #0,CRB
00000420  13FC 00FF 00E00004            17              MOVE.B  #$FF,DDRB               ; Initialize Port B
00000428  13FC 0004 00E00006            18              MOVE.B  #4,CRB
00000430  41F8 2000                     19              LEA     $2000,A0
00000434  13D8 00E00000                 20 LOOP         MOVE.B  (A0)+,DRA               ; Read Port A
0000043A  13D8 00E00004                 21              MOVE.B  (A0)+,DRB               ; Read Port B
00000440  4EB9 00000448                 22              JSR     DELAY
00000446  60EC                          23              BRA     LOOP
00000448  203C 00030D40                 24 DELAY        MOVE.L  #200000,D0
0000044E  0480 00000001                 25 WAIT         SUB.L   #1,D0
00000454  66F8                          26              BNE     WAIT
00000456  4E75                          27              RTS
                                        28
00002000                                29              ORG    $2000
00002000  87C78888C8F999B0              30              DC.B   $87,$C7,$88,$88,$C8,$F9,$99,$B0
00002008  868886F98EC886A1              31              DC.B   $86,$88,$86,$F9,$8E,$C8,$86,$A1
00002010  F988AF89                      32              DC.B   $F9,$88,$AF,$89
00002014                                33      END
No errors detected.
```

Object File

- Sequence of binary numbers and other information that represents machine code
- It is not executable form since the start address may be unknown and are linked to produce executable file

```
S0030000FC
S20C0000000404000000004006F
S21400040013FC000000E0000213FC00FF00E0000008
S21400041013FC000400E0000213FC000000E00006ED
S21400042013FC00FF00E0000413FC000400E00006DC
S21400043041F8200013D800E0000013D800E00004C4
S2140004404EB90000044860EC203C00030D400480D8
S20C000450000000166F84E757D
S21400200087C78888C8F999B0868886F98EC886A159
S208002010F988AF890E
S804000000FB
```

Assembly Language Format

```
<Label>   Opcode   <Operand>   <;comment>
```

- **Label** – optional and is used for branch and jump target purposes
  - First character should be alphabet (A-Z)
  - Internal register such as D0, A0 etc can't be a label
  - Label should not be more than 8 character

- **Opcode** – machine instruction or directive assembler
  - If size is not stated, word is considered

Assembly Language Format

- **Operand** – depends on type of operation
  - It may consists of constant, variable or special symbol
  - Separated from opcode with at least 1 space
  - A comma is needed to separate two operands

- **Comment** – optional and used as documentation purposes
  - Begins with ; or *

Constant

- Two types
  - **Integer** - prefix is required to represent the base
  - **String constant/ASCII**

Symbol	Base	Examples
%	2	%100011
$	16	$FADC
@	8	@2345
[none]	10	1234
'	ASCII	'C'
string		'WELCOME'

Directive Assemblers

- There are two type of statements
  - **Machine instruction**
  - **Directive assemblers (pseudo-op)**
- Directive assemblers are instructions to the assembler program
- They are not instruction to microprocessor, thus they do not have machine code representation
- They cover a number of functions including
  - Define symbols & assign them values
  - Controlling the flow of execution of the assembler
  - Setting format & content of the object and listing files

Directive Assemblers

Directive	Operation	Syntax
ORG	Set the program origin	ORG value
EQU	Equate value to symbol	LABEL EQU value
END	End of source program	END
DC	Define data constant	LABEL DC.s value
DS	Define RAM storage	LABEL DS.s value
EXTERN	Define external symbol	EXTERN LABEL
PUBLIC	Define public symbol	PUBLIC LABEL

ORG directive

- **Function**
  - Set the starting address in memory for instruction or data constant
  - May have more than one `ORG` in a program
- **Example**

```
00001000                          1      ORG $1000
00001000     4200                 2      CLR.B D0
00001002     203C 0000 0001       3      MOVE.L #1,D0
```

DC directive

- **Function**
  - Store data in memory
  - Can be any sizes (B, W, L) and if W and L, the initial location must begin at even address
- **General format**

```
[Label]  DC.s lists
```

DC directive

```
00002000                1          ORG  $2000
00002000  01020304    2    BEE   DC.B 1,2,3,4
00002004  00230050    3    BET   DC.W $23,$50
00002008  0000000A    4    BEP   DC.L 10

00003000                5          ORG  $3000
00003000  00001000    6    DEE   DC.L $1000
00003004  31323334    7    DET   DC.B '1234'
00003008  00410061    8    BEN   DC.W 'A',a'
```

DS directive

- **Function**
  - Allocate/reserve location in memory
  - Can be any size (B, W, L) and if W and L, the initial location must begin at even address

- **General format**

```
[Label]  DS.s  sum
```

DS directive

Example

```
00005000                        1               ORG     $5000
00005000                        2       BEC     DS.B    2
00005002                        3       BEI     DS.W    4
0000500A                        4       BEP     DS.L    2
00005012        6162636465      5       DTE     DC.B    'abcde'
```

- *The memory location reserve are not initialized, thus they will contain random data*

EQU directive

- **Function -** Assign certain value to a symbol
- **General format**

```
Symbol  EQU  value
```

- **Example**

```
00004000                    1          ORG    $4000
00000010                    2      NO  EQU    $10
00007000                    3          ORG    $7000
00007000    7010            4          MOVEQ  #NO,D0
```

END directive

- **Function**
  - Used to indicate the end of source program
  - Statements following the `END` directive are not processed by assembler
  - The label of the `END` directive represents the entry point for the program
- **Example**

```
00007000                3               ORG    $7000
00007000    7E01        4               MOVEQ  #1,D7
                                        END
```

* directive

- **Function**
  - Contain the address of instruction that is being assembled
- **Example**

```
JMP  *+8
```

means jump 8 bytes forward

```
BRA  *
```

means branch to this address

More Examples..

Memory	Machine Code	Line	Assembly Code
00002000		1	ORG $2000
00002000	203C 00000000	2	MOVE.L #0,D0
00002006	4287	3	CLR.L D7
00002008	1239 00002500	4	MOVE.B TABLE,D1
0000200E	D200	5	ADD.B D0,D1
00002010	13C1 00002501	6	MOVE.B D1,SUM
00002016		7	
		8	ORG $2500
00002500	50	9	TABLE DC.B $50
00002501		10	SUM DS.B 1
00002502		11	END

Self-Test

Exercise

Examine the following code and obtain the memory map for this code. What is the value for symbolic label BEE and its content

```
     ORG  $1000
DEE  DS.B 4
BEC  DS.W 3
BEA  DC.W DEE
BEP  DC.L 1
BEE  DC.W DEE+BEA
```

Self-Test

Exercise

Examine the following code and obtain the memory map for this code. What is the content of D0, D1 and D2 ?

```
        ORG     $2000
FKEE    DC.L    23454
FKASA   DC.W    233
FKM     DC.B    '123459'
START   MOVE.L  FKASA,D0
        MOVE.L  #FKM,D1
        MOVE.L  FKM-2,D2
```