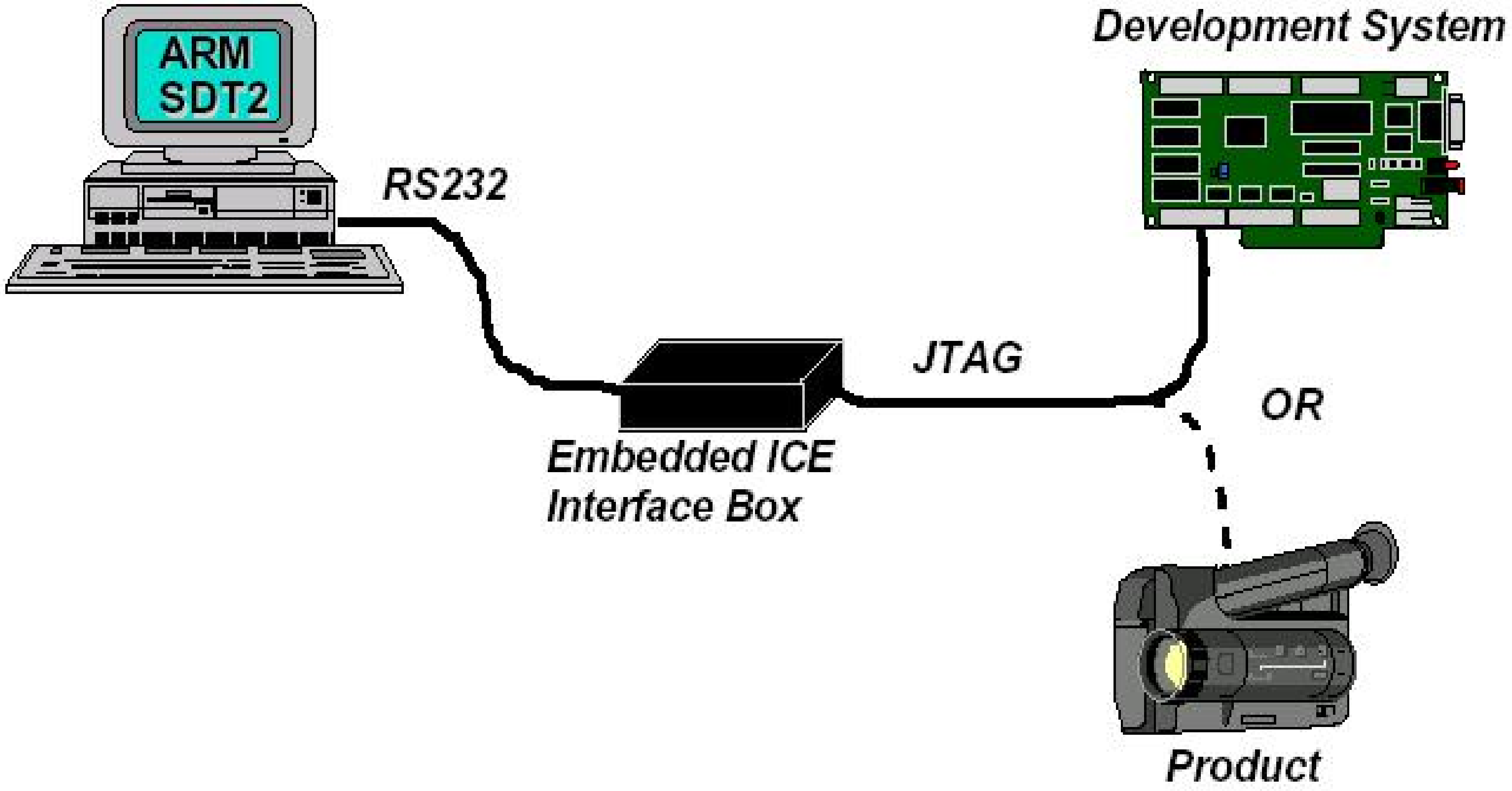
Make:
Sensors
Projects and Experiments to Measure the World with Arduino and Raspberry Pi
Tero Karvinen, Kimmo Karvinen & Ville Valtokari

4. Development Tools

Content

- Host and Target Machines
- Linker/Locators for Embedded Software
- Getting Embedded Software into the Target System

Host and Target Machines



Host and Target Machines

- Host computer
  - The standard platform being used to develop the software and link to the target system for debugging
- Target system
  - The embedded system under development

Host and Target Machines

- Cross-development
  - Using host-based tools to create a code image that will execute on a different instruction set architecture

- Example:
  - Write a C program on your PC
  - Compile it to run on a PowerPC 603 using a Cross-compiler
  - Create a runtime image for execution in the target system

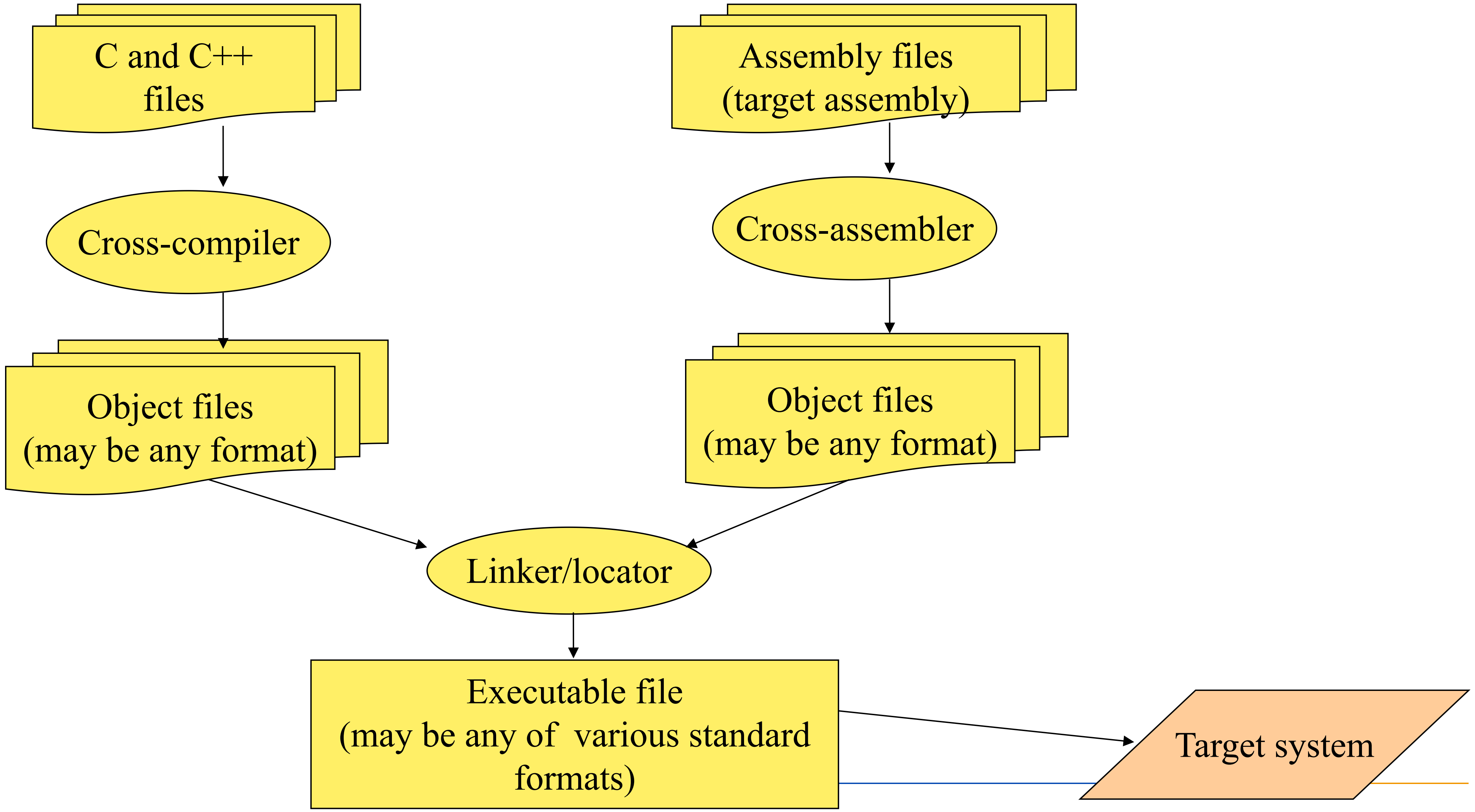
Host and Target Machines

- Cross-Compilers
  - A compiler that runs on your host system but produces the binary instructions that will be understood by your target microprocessor.

Host and Target Machines

- Cross-Assemblers
  - Another tool that you will need if you must write any of your program in assembly language is a cross-assembler.

Figure : Tool Chain for Building Embedded Software


C and C++ files
Assembly files (target assembly)
Cross-compiler
Cross-assembler
Object files (may be any format)
Object files (may be any format)
Linker/locator
Executable file (may be any of various standard formats)
Target system

Tool Chain for Building Embedded Software

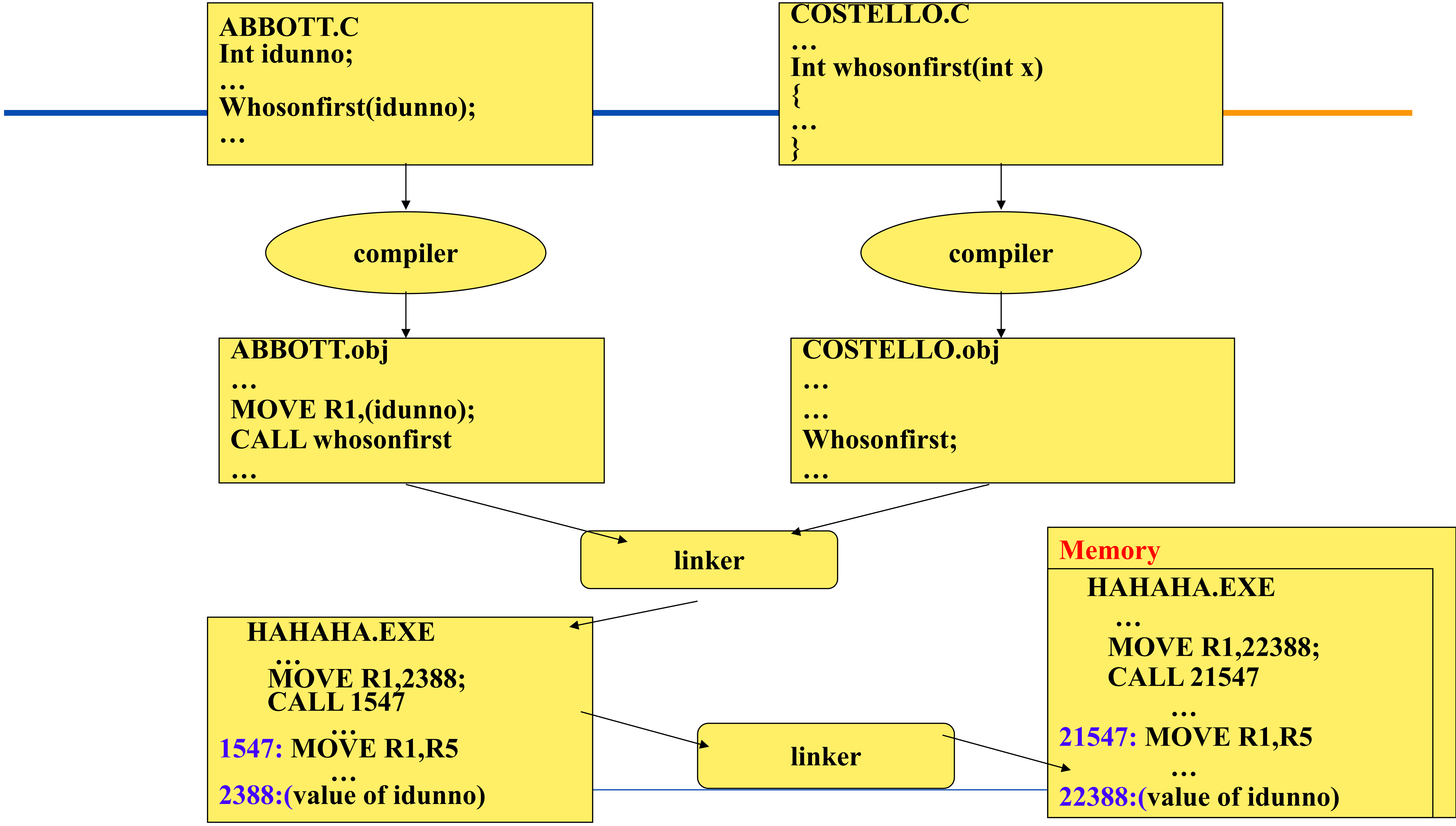
- the output files from each tools become the input files for the next. So the tools must be compatible with one another.
- A set of tools that is compatible in this way is called a tool chain.

Linker/Locators for Embedded Software

- Define
  - Linkers for embedded systems are often called locators or linker/locators, as well as cross-linker.

Linker/Locators for Embedded Software

- the differences between native linker and locator.
- Address Resolution (地址解析)
  - The first difference between a native linker and a locator is the nature of the output files that they create.



```
ABBOTT.C
Int idunno;
…
Whosonfirst(idunno);
…
```

```
COSTELLO.C
…
Int whosonfirst(int x)
{
…
}
```

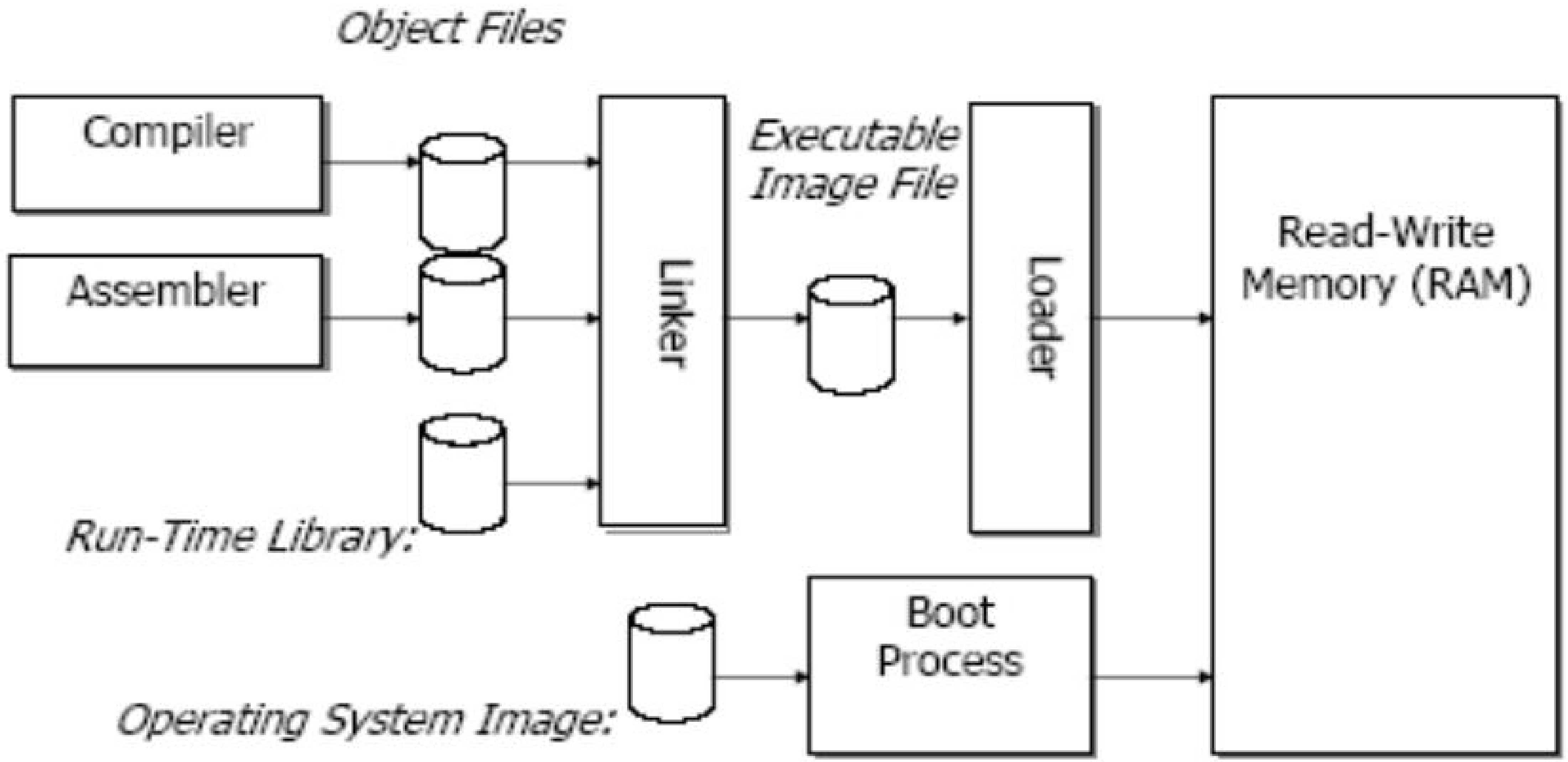
```
ABBOTT.obj
…
MOVE R1,(idunno);
CALL whosonfirst
…
```

```
COSTELLO.obj
…
…
Whosonfirst;
…
```

```
HAHAHA.EXE
   …
   MOVE R1,2388;
   CALL 1547
       …
1547: MOVE R1,R5
       …
2388:(value of idunno)
```

```
HAHAHA.EXE
   …
   MOVE R1,22388;
   CALL 21547
       …
21547: MOVE R1,R5
       …
22388:(value of idunno)
```

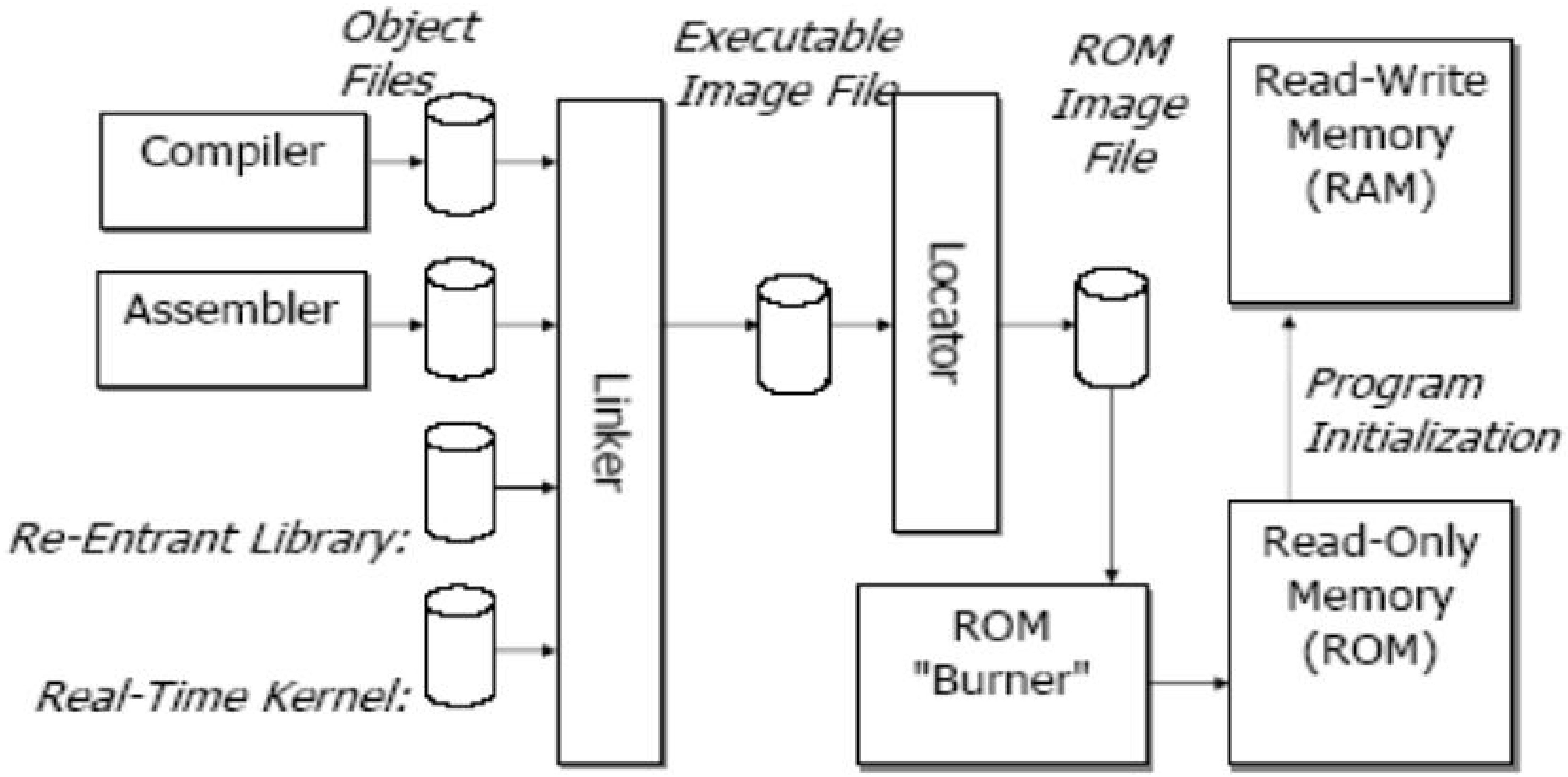
台式机软件从“源码程序”到“机器码文件”的过程

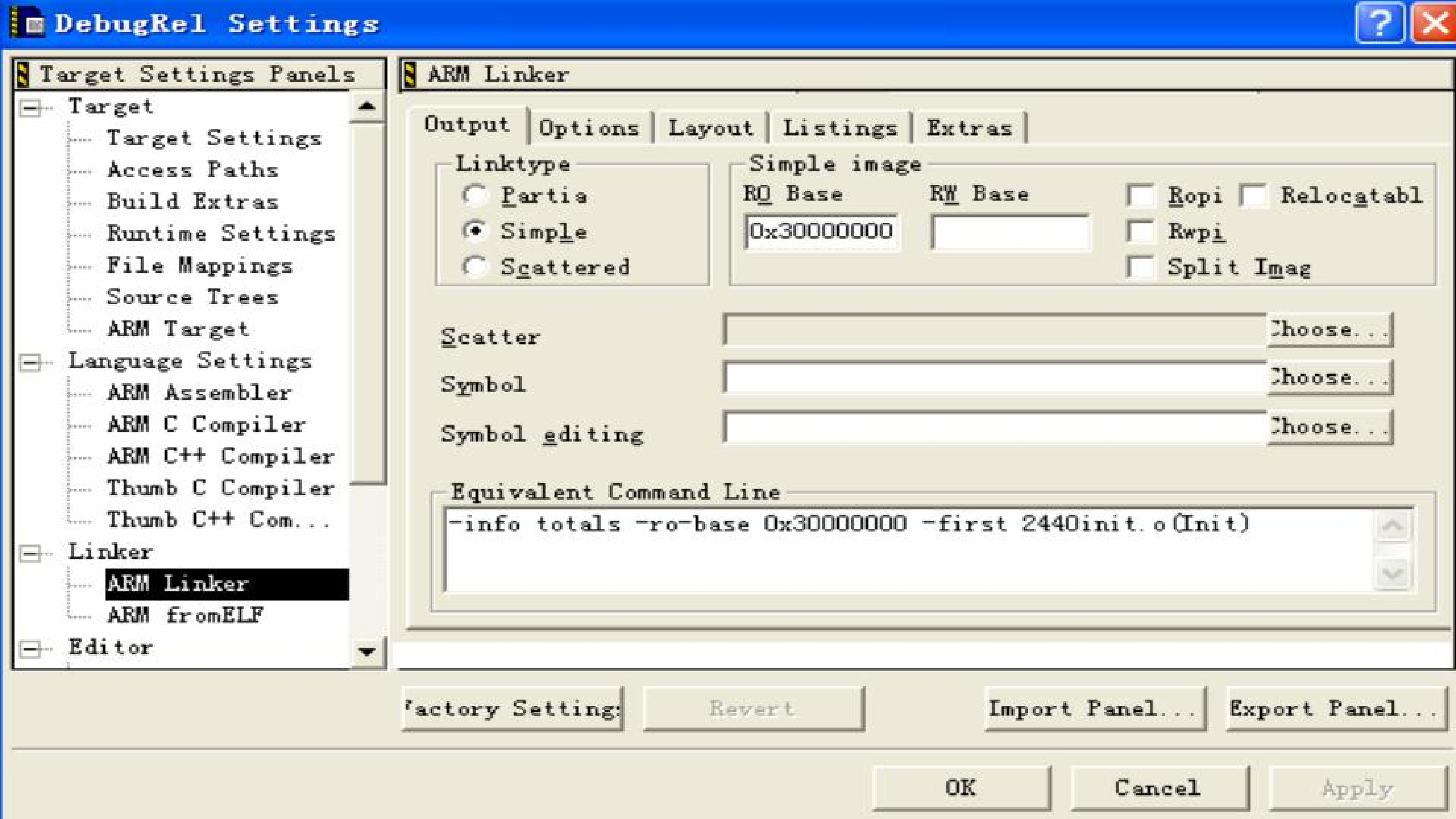


Linker/Locators for Embedded Software

- Address Resolution (locator)
  - In most embedded system, there is no loader.
  - When the locator is done, its output will be copied onto the target system.
  - Therefore, the locator must know where in memory the program will reside and fix up all of the addresses.
  - Locators have mechanisms that allow you to tell them where the program will be on the target system.

嵌入式软件从“源码程序”到“机器码文件”的过程





Linktype: Partia [✓ Simple] Scattered

Simple image

RO Base: 0x30000000

RW Base:

Ropi ☐ Relocatabl ☐ Rwpi ☐ Split Imag ☐

Scatter:

Symbol:

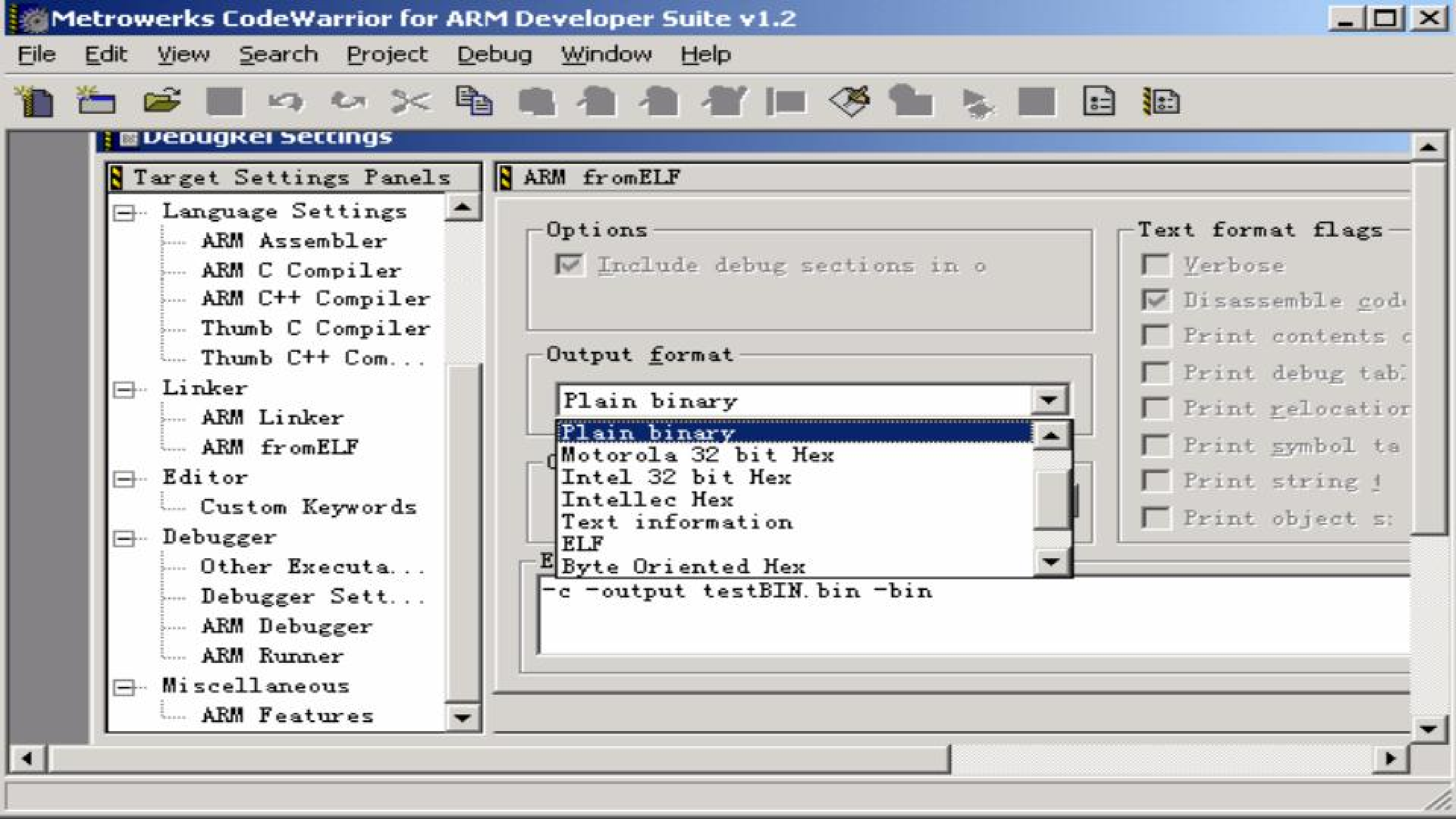
Symbol editing:

Equivalent Command Line

```
-info totals -ro-base 0x30000000 -first 2440init.o(Init)
```

Linker/Locators for Embedded Software

- Address Resolution (locator)
  - Locators use any number of different output file formats.
    - Intel Hex File Format
    - Motorola S-Record Format
    - And so on ……
  - The tools you’re using to load your program into your target system must understand wharever file format your locator produces.



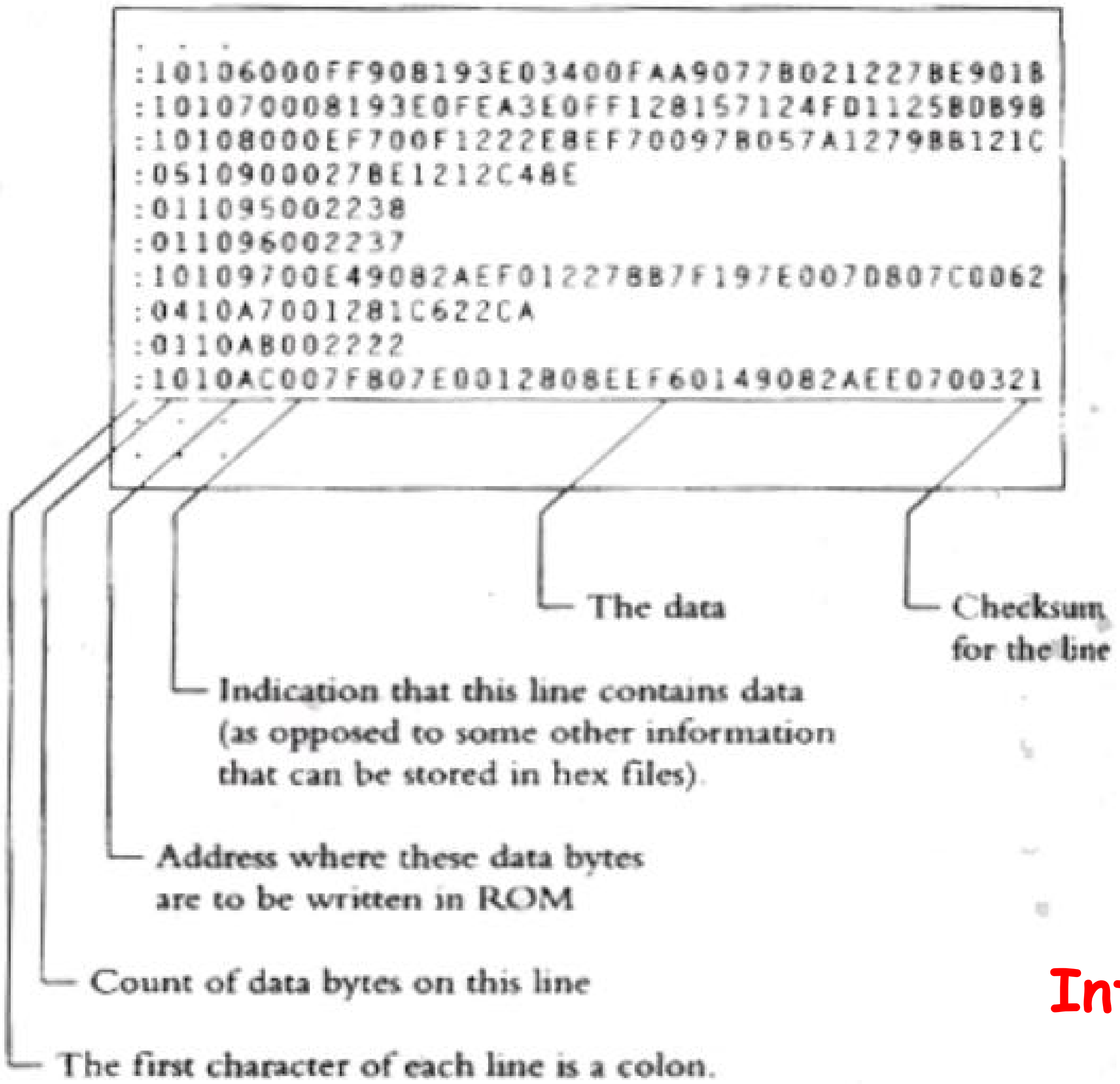
[✓] Include debug sections in o

Output format: Plain binary

- Plain binary
- Motorola 32 bit Hex
- Intel 32 bit Hex
- Intellec Hex
- Text information
- ELF
- Byte Oriented Hex

Text format flags:
- [ ] Verbose
- [✓] Disassemble cod
- [ ] Print contents c
- [ ] Print debug tab
- [ ] Print relocatio
- [ ] Print symbol ta
- [ ] Print string t
- [ ] Print object s

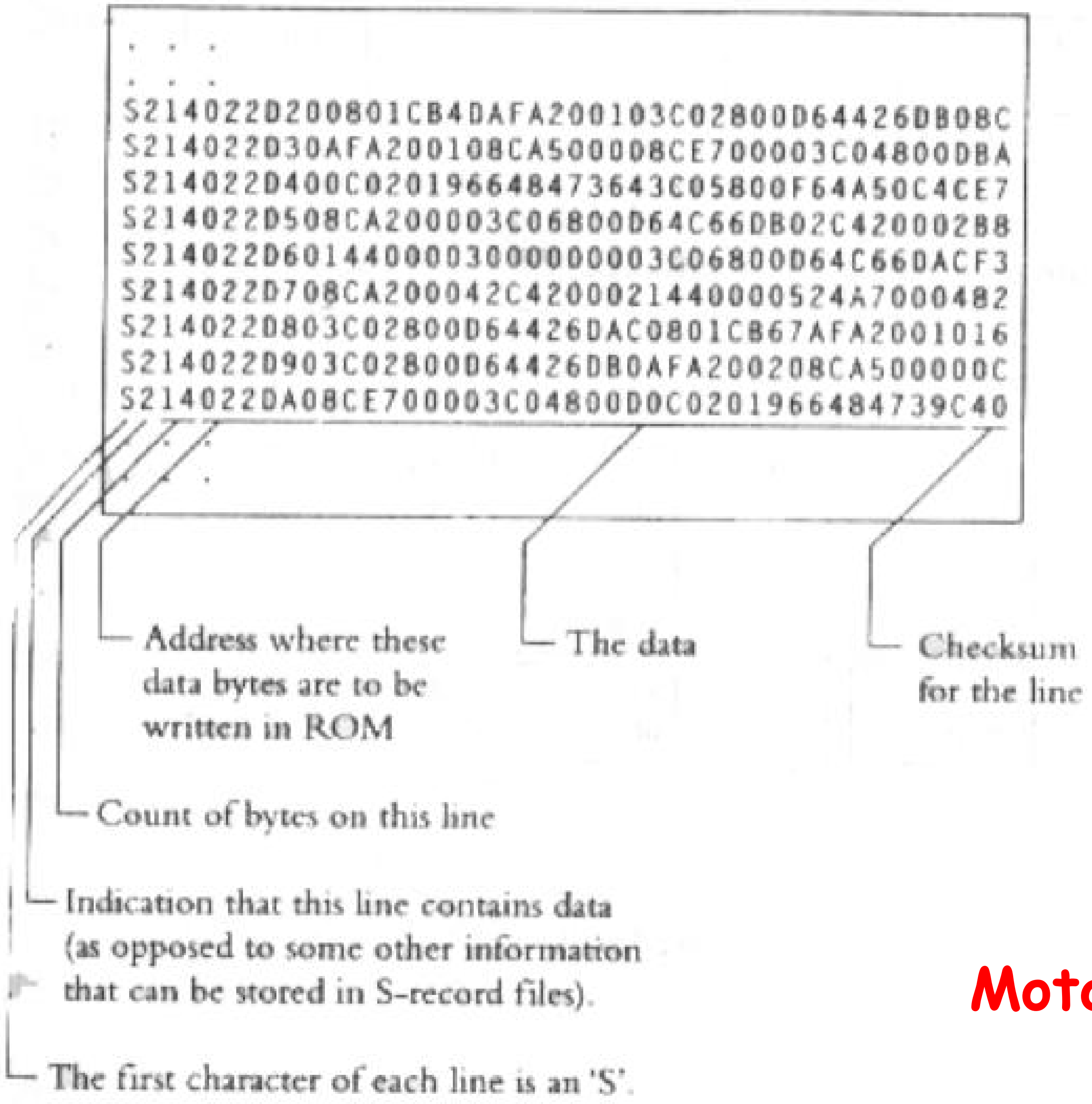
```
-c -output testBIN.bin -bin
```



```
.  .  .
:10106000FF90B193E03400FAA9077B021227BE901B
:101070008193E0FEA3E0FF128157124FD1125BDB98
:10108000EF700F1222E8EF70097B057A12798B121C
:0510900027BE1212C48E
:011095002238
:011096002237
:10109700E49082AEF012278B7F197E0070807C0062
:0410A7001281C622CA
:0110AB002222
:1010AC007F807E0012808EEF60149082AEE0700321
.  .  .
.  .  .
```

The data
Checksum for the line
Indication that this line contains data (as opposed to some other information that can be stored in hex files).
Address where these data bytes are to be written in ROM
Count of data bytes on this line
The first character of each line is a colon.

Intel Hex File Format



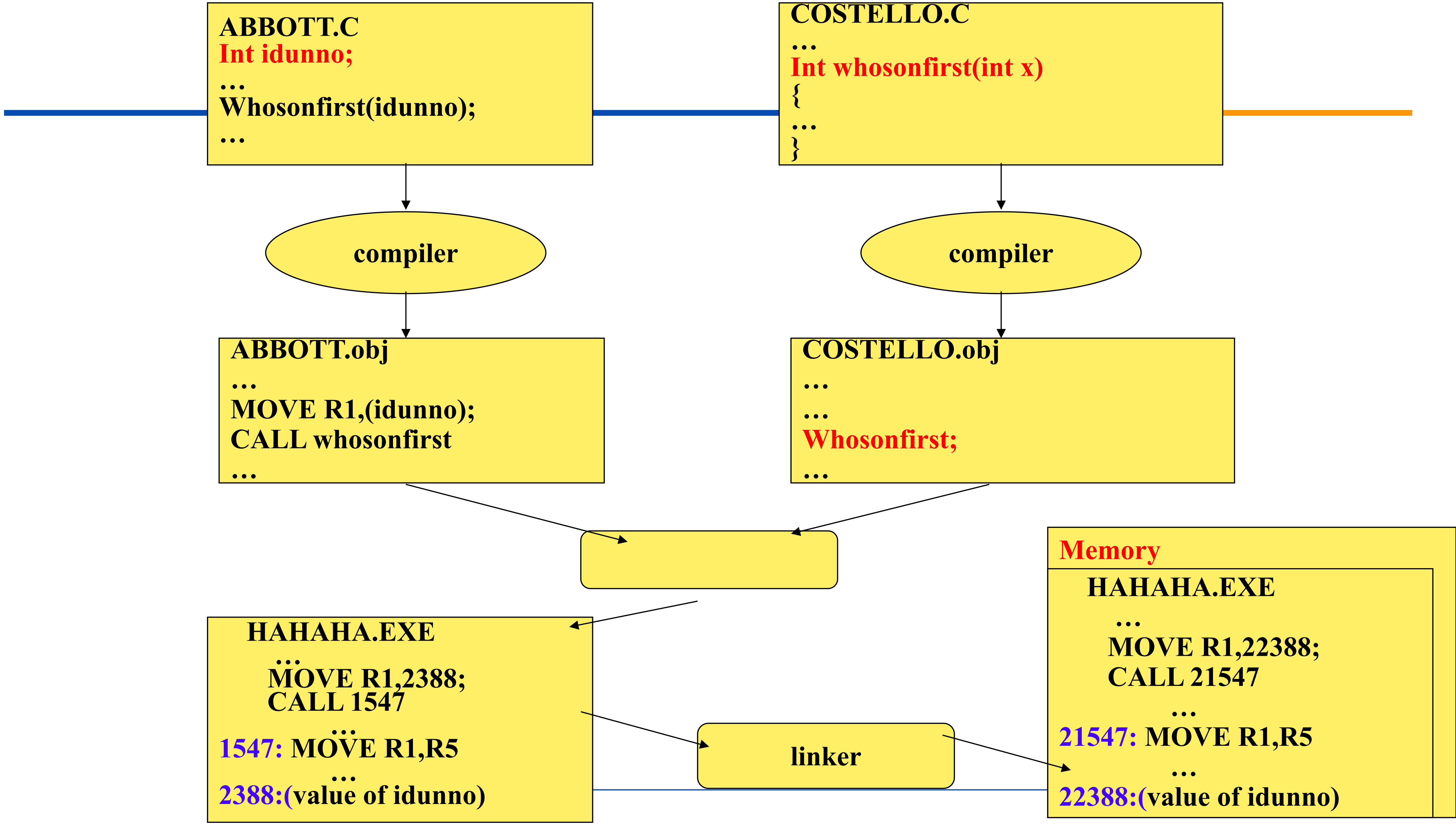
```
. . .
. . .
S214022D200801CB4DAFA200103C02800D64426DB08C
S214022D30AFA200108CA500008CE700003C04800DBA
S214022D400C020196648473643C05800F64A50C4CE7
S214022D508CA200003C06800D64C66DB02C420002B8
S214022D601440000300000003C06800D64C66DACF3
S214022D708CA200042C42000214400005 24A7000482
S214022D803C02800D64426DAC0801CB67AFA2001016
S214022D903C02800D64426DB0AFA200208CA500000C
S214022DA08CE700003C04800D0C020196648473 9C40
. . .
. . .
```

Address where these data bytes are to be written in ROM
The data
Checksum for the line
Count of bytes on this line
Indication that this line contains data (as opposed to some other information that can be stored in S-record files).
The first character of each line is an 'S'.

Motorola S-Record Format

Linker/Locators for Embedded Software

- Locating Program Components Properly
  - Another issue that locators must resolve in the embedded environment is that some parts of the program need to end up in the ROM and some parts of the program need to end up in the RAM.



```
ABBOTT.C
Int idunno;
…
Whosonfirst(idunno);
…
```

```
COSTELLO.C
…
Int whosonfirst(int x)
{
…
}
```

```
ABBOTT.obj
…
MOVE R1,(idunno);
CALL whosonfirst
…
```

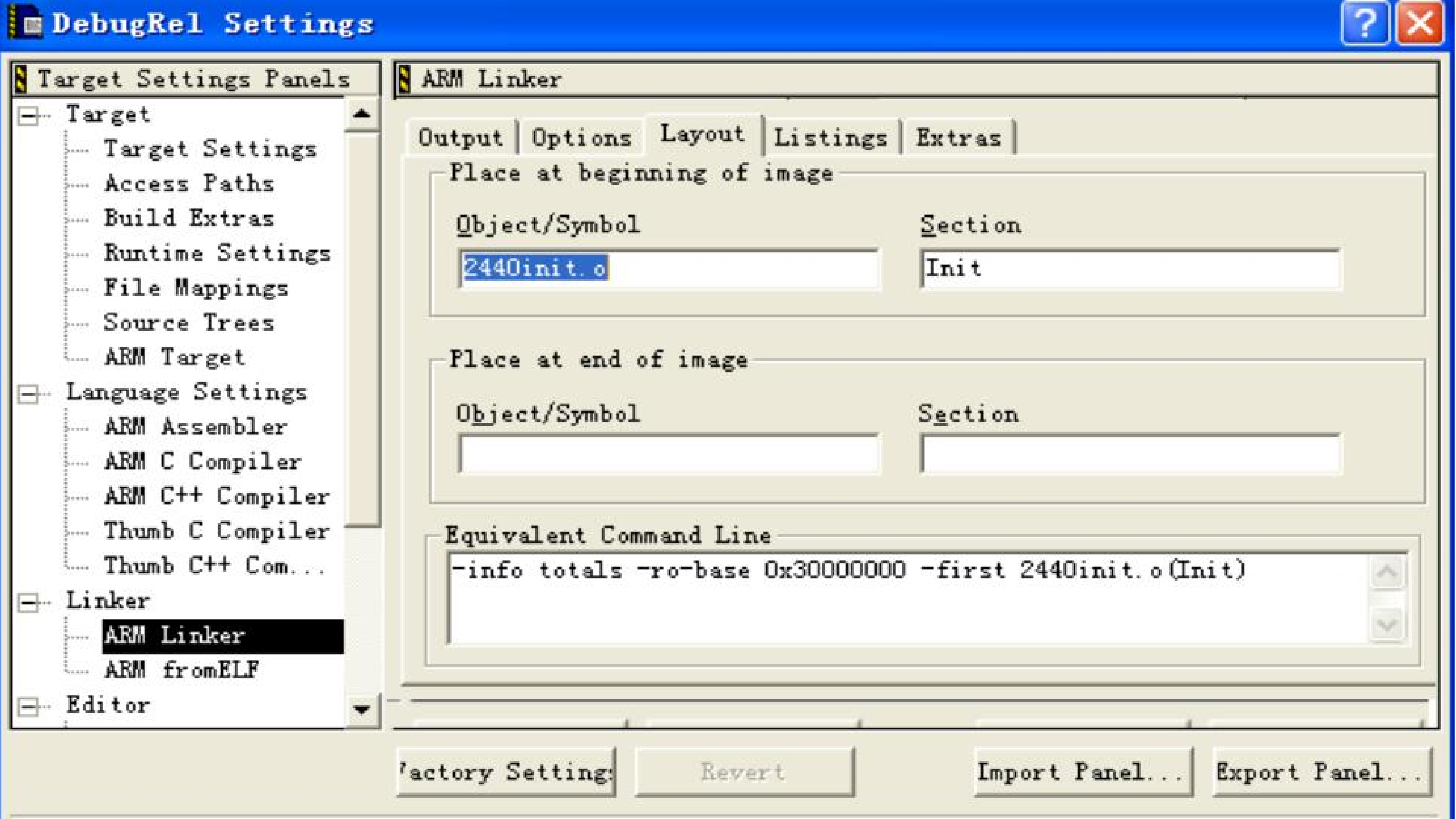
```
COSTELLO.obj
…
…
Whosonfirst;
…
```

```
   HAHAHA.EXE
     …
     MOVE R1,2388;
     CALL 1547
          …
1547: MOVE R1,R5
          …
2388:(value of idunno)
```

```
   HAHAHA.EXE
     …
     MOVE R1,22388;
     CALL 21547
          …
21547: MOVE R1,R5
          …
22388:(value of idunno)
```

Linker/Locators for Embedded Software

- Locating Program Components Properly
  - Most tool chains deal with this problem by dividing programs into segments.



Place at beginning of image

Object/Symbol: 2440init.o

Section: Init

Place at end of image

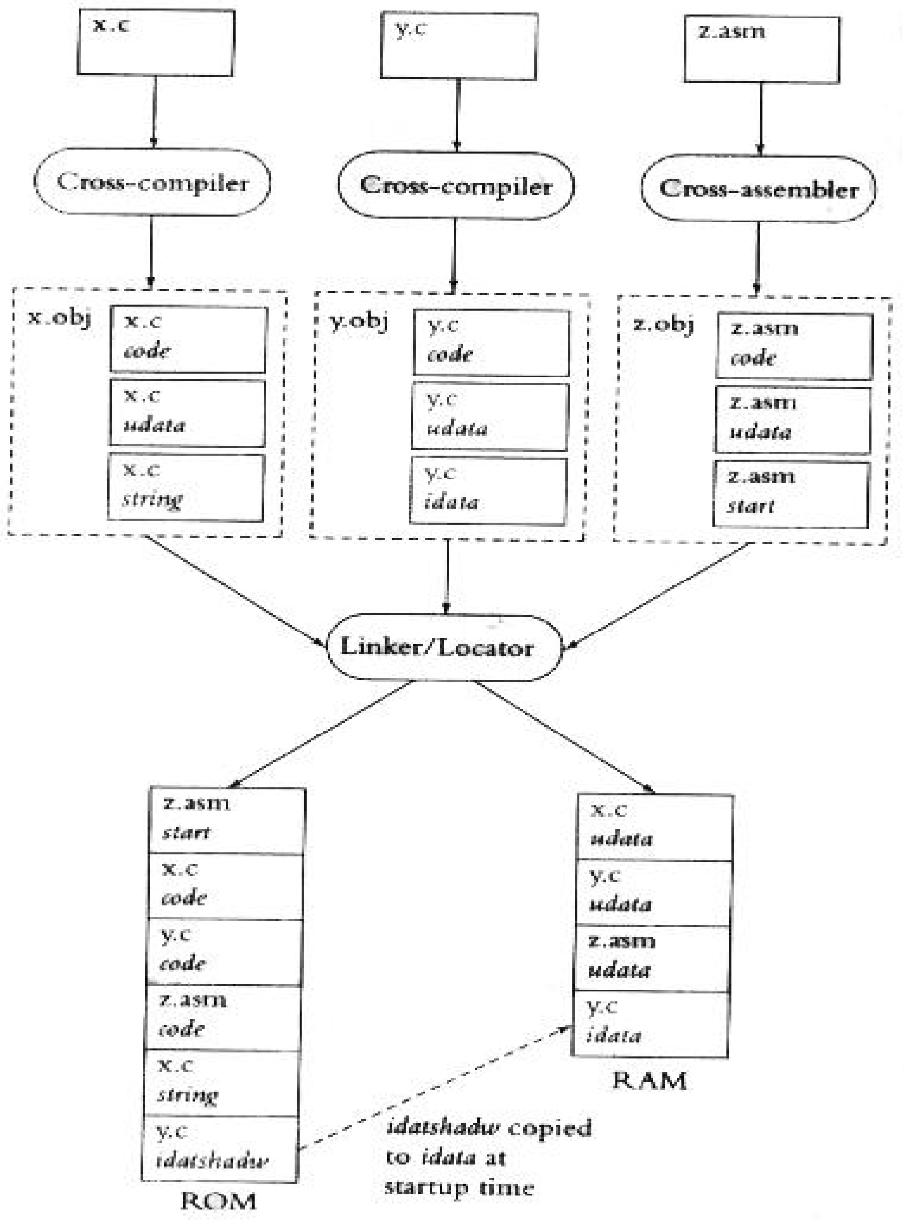
Object/Symbol:

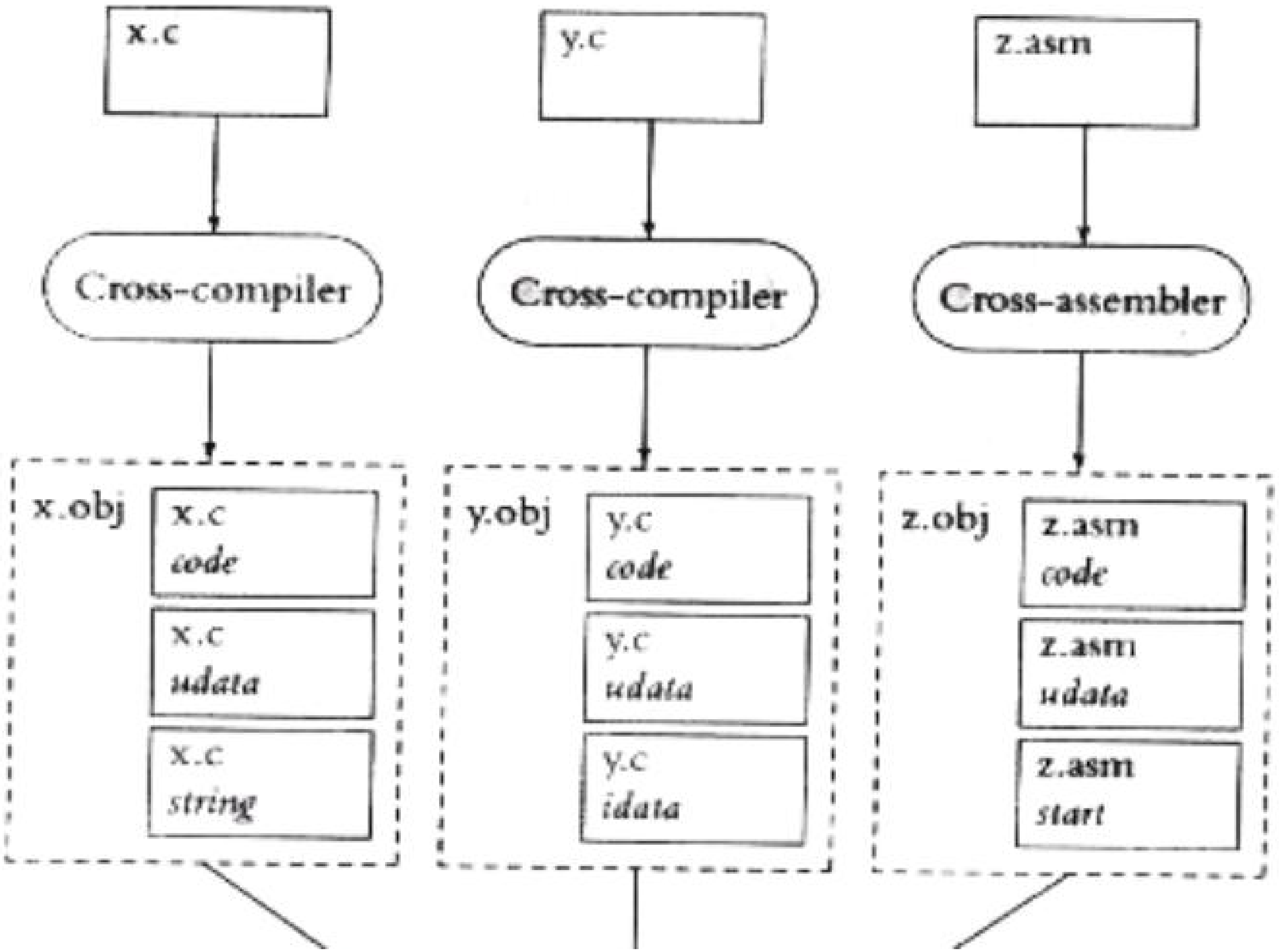
Section:

Equivalent Command Line

```
-info totals -ro-base 0x30000000 -first 2440init.o(Init)
```

Figure： How the Tool Chain Uses Segments

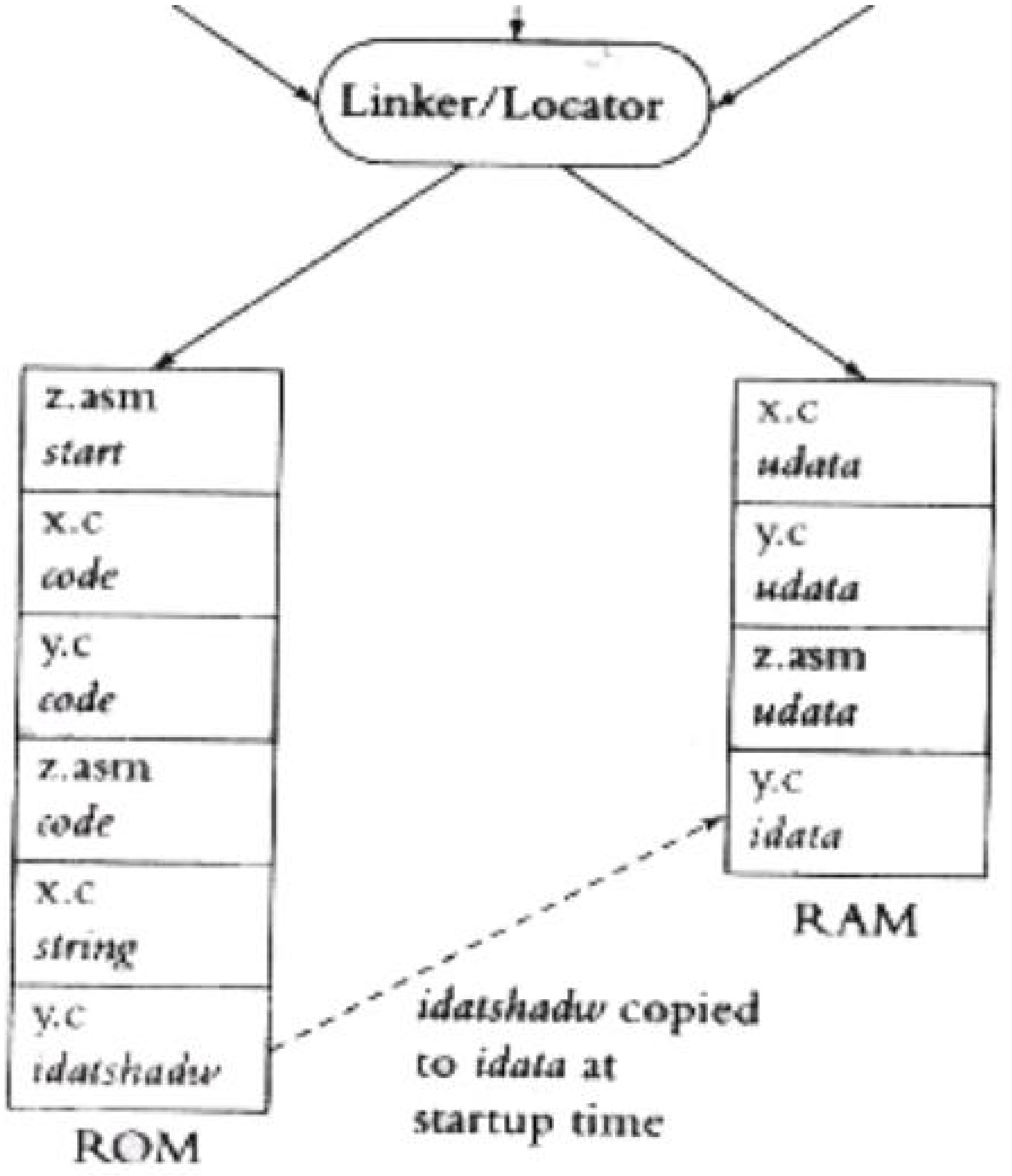



x.c
y.c
z.asm
Cross-compiler
Cross-compiler
Cross-assembler
x.obj
x.c code
x.c udata
x.c string
y.obj
y.c code
y.c udata
y.c idata
z.obj
z.asm code
z.asm udata
z.asm start

```
    AREA    Init, CODE, READONLY
    ENTRY
    EXPORT __ENTRY
__ENTRY
ResetEntry
    ……
    AREA RamData, DATA, READWRITE
    ^   _ISR_STARTADDRESS          ;  _ISR_STARTADDRESS=0x33FF_FF00
    HandleReset         #   4
    HandleUndef         #   4
    HandleSWI           #   4
    HandlePabort        #   4
    HandleDabort        #   4
    HandleReserved#   4
    HandleIRQ           #   4
    HandleFIQ           #   4
```

Linker/Locators for Embedded Software

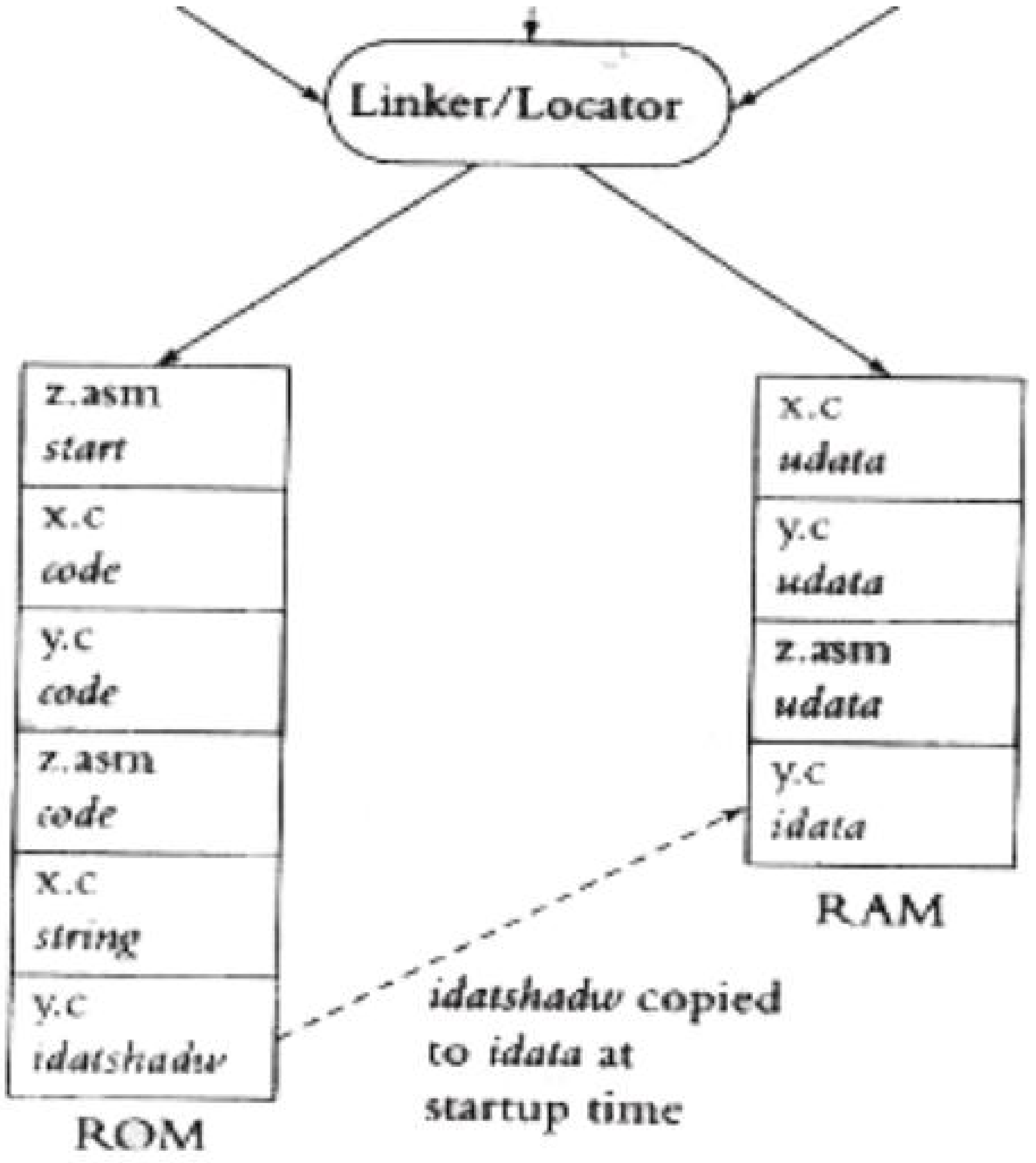
- Locating Program Components Properly
  - Most cross-compilers automatically divide each module they compile into two or more segments.
    - The instructions
    - Uninitialized data
    - Initialized data
    - Constant strings


Linker/Locator
z.asm
start
x.c
code
y.c
code
z.asm
code
x.c
string
y.c
idatshadw
ROM
x.c
udata
y.c
udata
z.asm
udata
y.c
idata
RAM
idatshadw copied
to idata at
startup time

Linker/Locators for Embedded Software

- Initialized Data and Constant Strings

```
#define FREQ_DEFAULT 2410
          ⋮
static int iFreq = FREQ_DEFAULT;
          ⋮
Void vSetFreq ( int iFreqNew)
{
          iFreq = iFreqNew;
}
```


Linker/Locator
z.asm
start
x.c
code
y.c
code
z.asm
code
x.c
string
y.c
idatshadw
ROM
x.c
udata
y.c
udata
z.asm
udata
y.c
idata
RAM
idatshadw copied
to *idata* at
startup time

Linker/Locators for Embedded Software

- Initialized Data and Constant Strings

  Char *sMsg = “Reactor is melting!”;

- If the only operation that you ever peform with the variable is to print it with a statement such as

  - Printf(“PROBLEM: %s”, sMsg) .

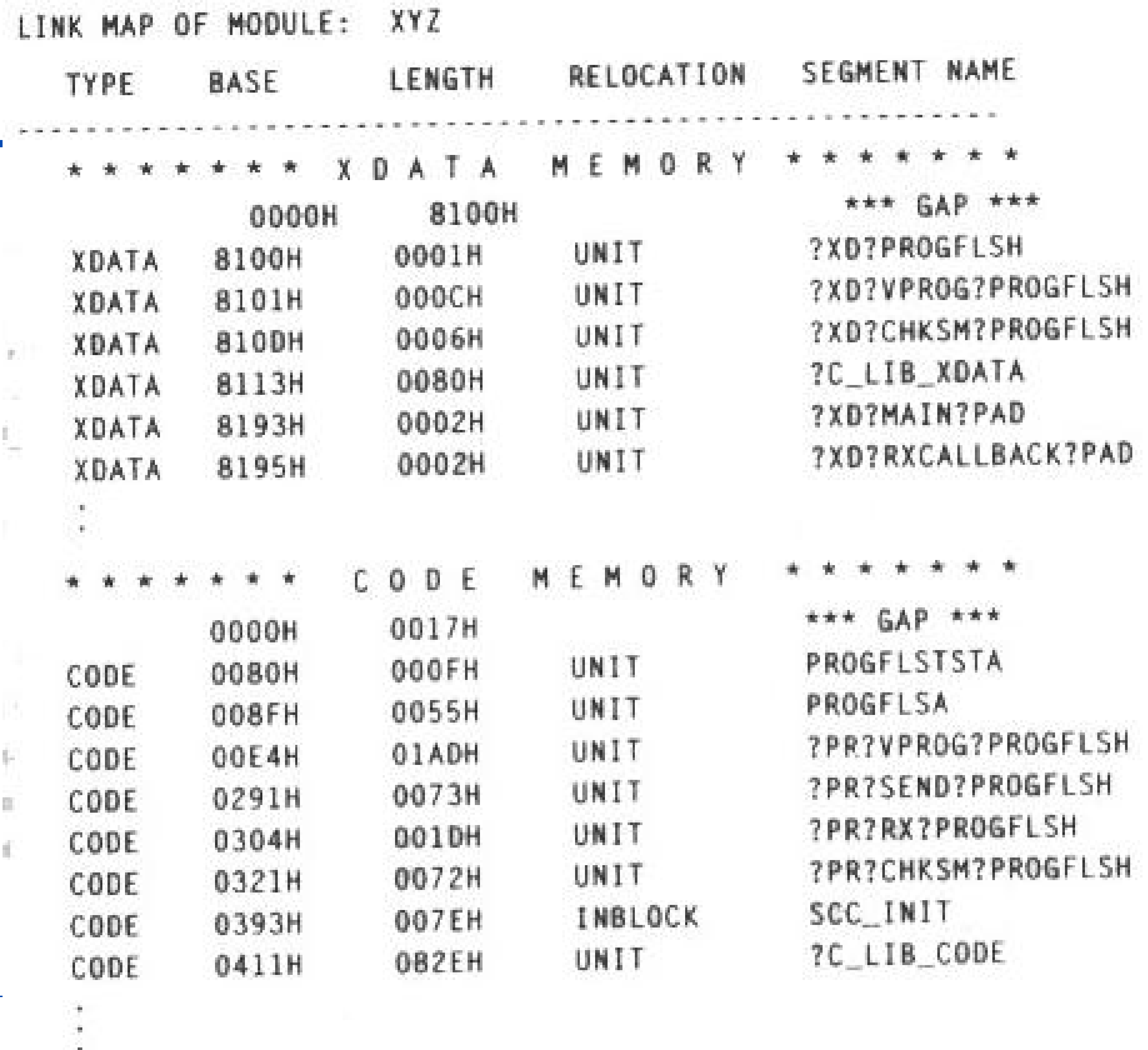
- On the other hand, the compiler has no way of knowing that you will not do something like this

  - Strcpy (&sMsg[11], “OK”)

Linker/Locators for Embedded Software

- Locator Maps
  - Most locators will create an output file, called a map.
  - The map lists where the locator placed each of the segments in memory.
  - The map also includes the addresses of public functions and perhaps the addresses of the global data variables.
  - Figure 2.4 for an example of a locator map.

Figure : Locator Map



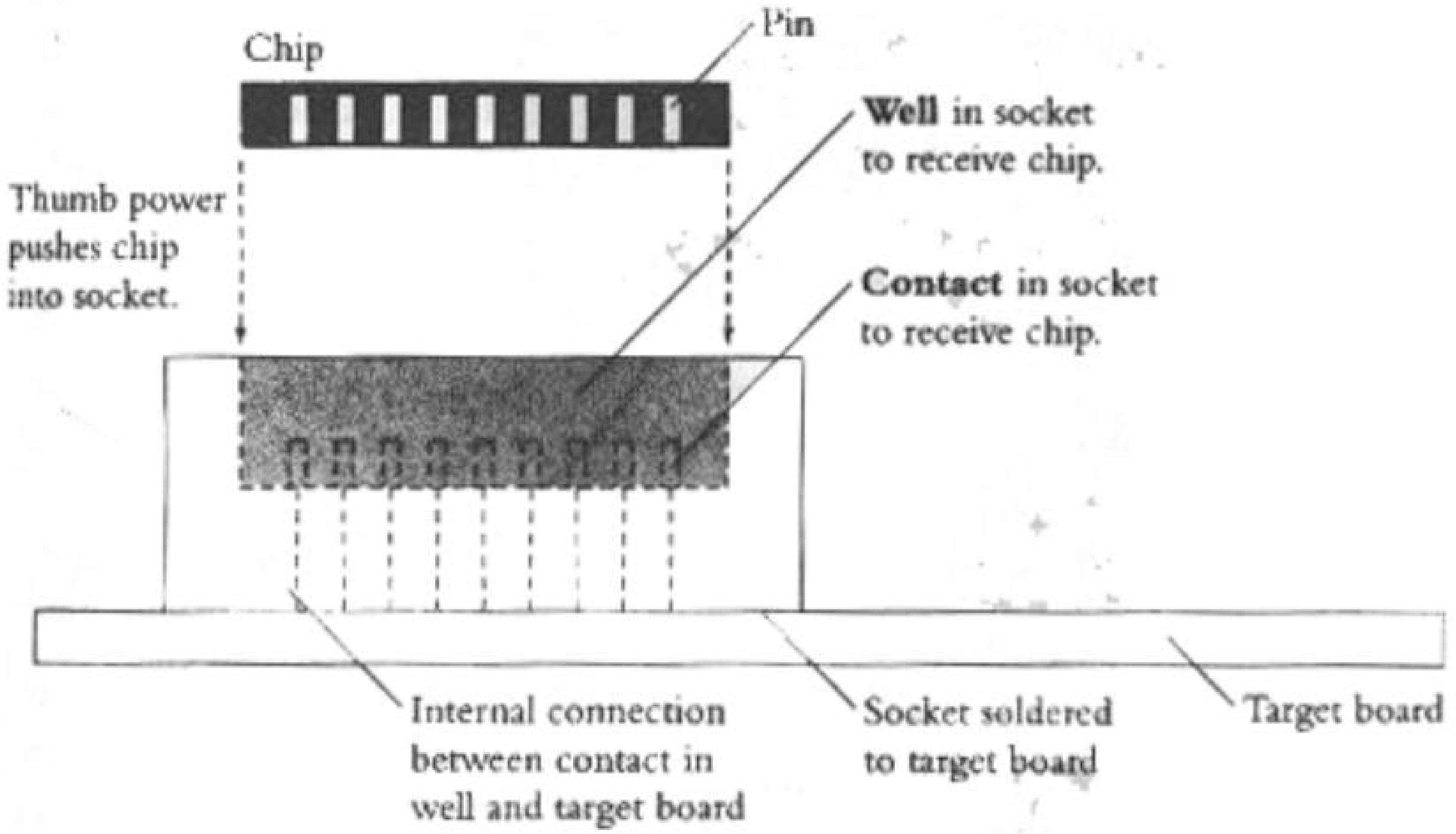
```
LINK MAP OF MODULE:  XYZ
    TYPE      BASE        LENGTH      RELOCATION    SEGMENT NAME
-----------------------------------------------------------------------
    * * * * * * *  X D A T A   M E M O R Y  * * * * * * *
              0000H       8100H                      *** GAP ***
    XDATA     8100H       0001H       UNIT          ?XD?PROGFLSH
    XDATA     8101H       000CH       UNIT          ?XD?VPROG?PROGFLSH
    XDATA     810DH       0006H       UNIT          ?XD?CHKSM?PROGFLSH
    XDATA     8113H       0080H       UNIT          ?C_LIB_XDATA
    XDATA     8193H       0002H       UNIT          ?XD?MAIN?PAD
    XDATA     8195H       0002H       UNIT          ?XD?RXCALLBACK?PAD
    :

    * * * * * * *  C O D E   M E M O R Y  * * * * * * *
              0000H       0017H                     *** GAP ***
    CODE      0080H       000FH       UNIT          PROGFLSTSTA
    CODE      008FH       0055H       UNIT          PROGFLSA
    CODE      00E4H       01ADH       UNIT          ?PR?VPROG?PROGFLSH
    CODE      0291H       0073H       UNIT          ?PR?SEND?PROGFLSH
    CODE      0304H       001DH       UNIT          ?PR?RX?PROGFLSH
    CODE      0321H       0072H       UNIT          ?PR?CHKSM?PROGFLSH
    CODE      0393H       007EH       INBLOCK       SCC_INIT
    CODE      0411H       082EH       UNIT          ?C_LIB_CODE
    :
```

Getting Embedded Software into the Target System

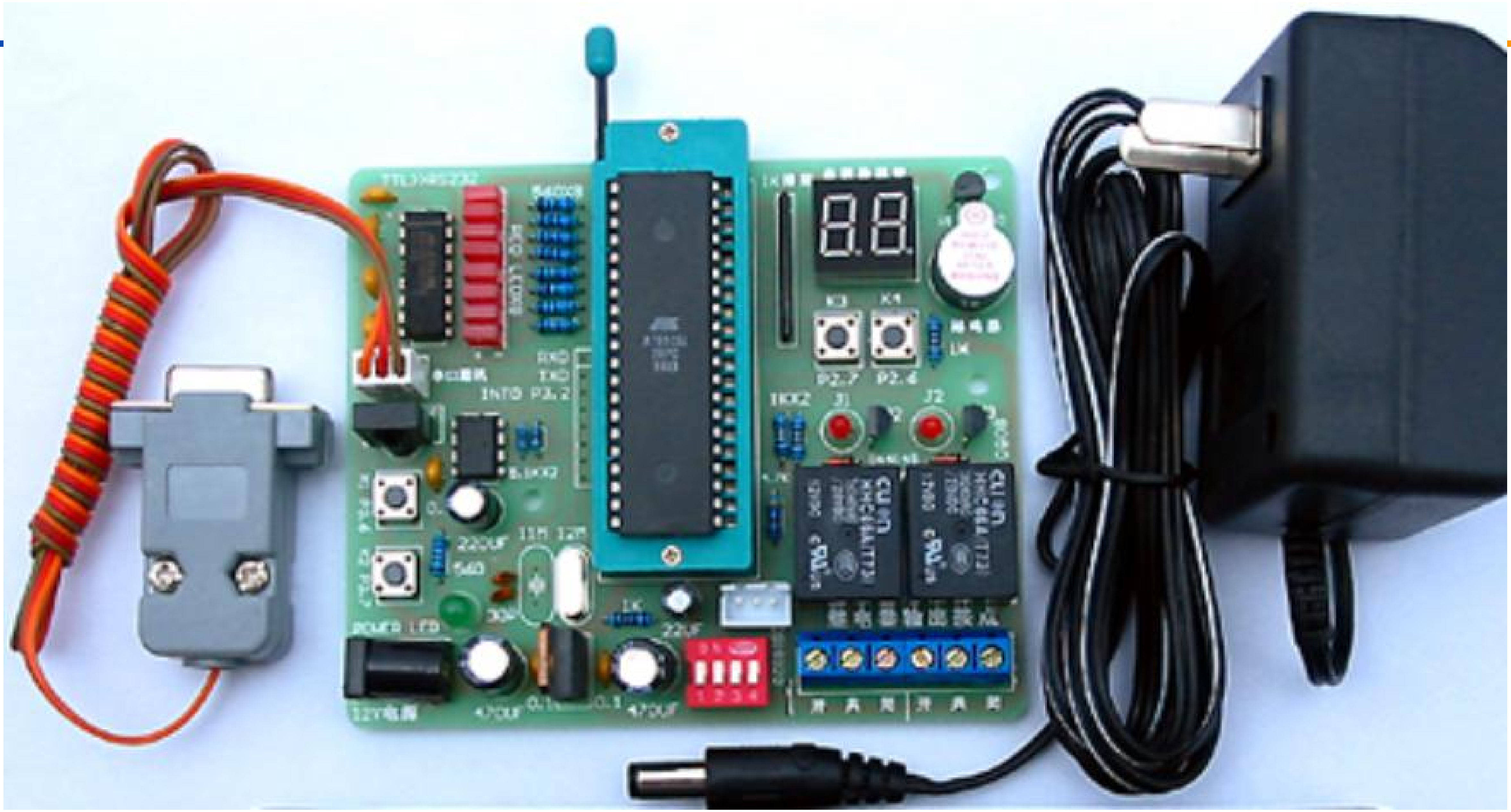
- PROM Programmers
- ROM Emulators
- In-Circuit Emulators
- Flash
- Monitors

Getting Embedded Software into the Target System

- PROM Programmers
  - The classic way to get the software from the locator output file into the target system is to use the file to create a ROM or a PROM.
  - Figure 2.5


Chip
Pin
Well in socket to receive chip.
Thumb power pushes chip into socket.
Contact in socket to receive chip.
Internal connection between contact in well and target board
Socket soldered to target board
Target board

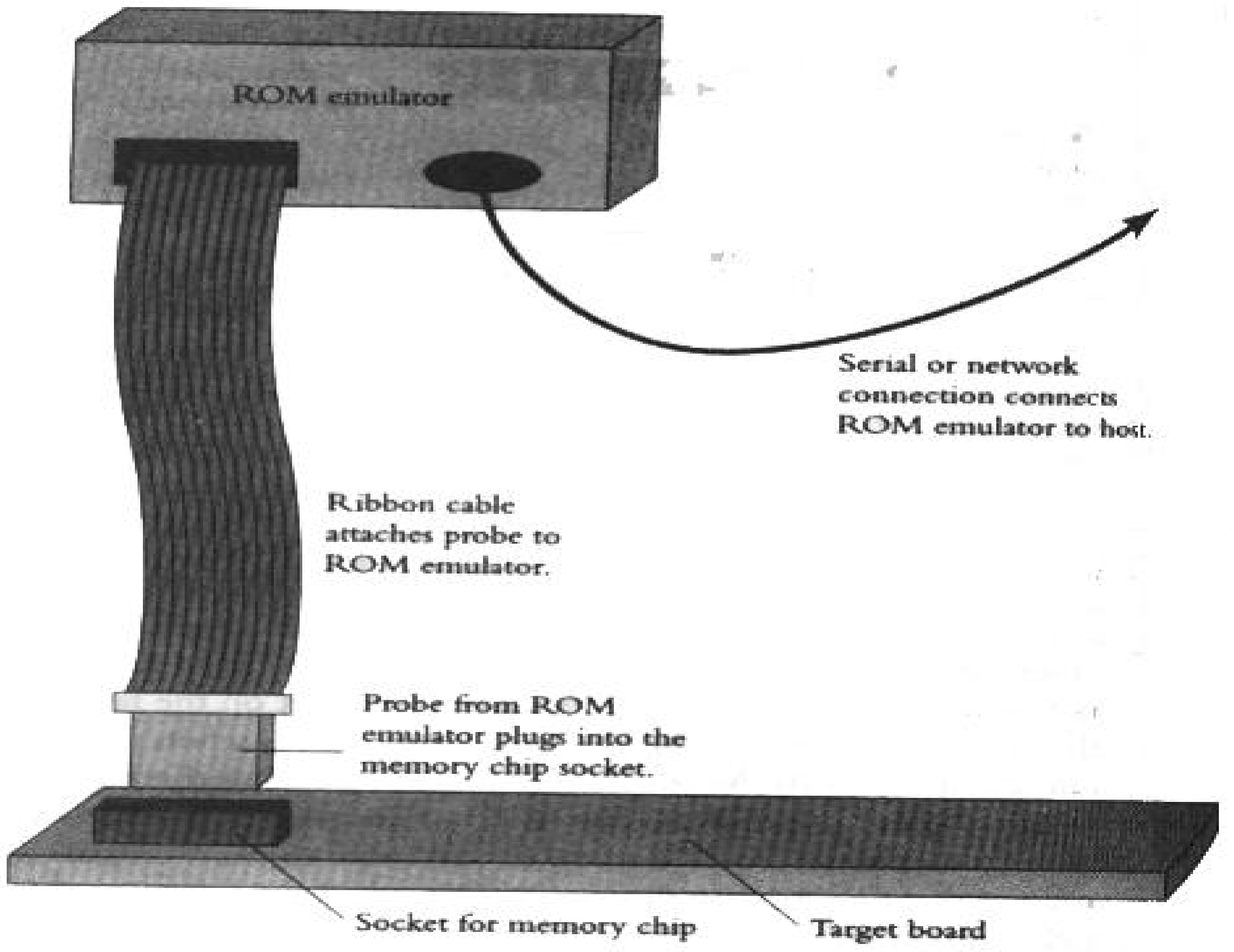
INTELLIGENT UNIVERSAL PROGRAMMER



Getting Embedded Software into the Target System

- ROM Emulators
  - Another popular mechanism for getting software into the target system for debugging purposes is to use a ROM emulator, a device that replaces the ROM in the target system.
  - see the Figure

Figure : ROM Emulator


ROM emulator
Serial or network connection connects ROM emulator to host.
Ribbon cable attaches probe to ROM emulator.
Probe from ROM emulator plugs into the memory chip socket.
Socket for memory chip
Target board

Getting Embedded Software into the Target System

- In-Circuit Emulators
  - An in-circuit emulator, sometimes referred to as an emulator or by the acronym ICE. replaces the microprocessor in the target circuit.
  - From the perspective of the other chips in the target circuit, the emulator appears to be the microprocessor.

Getting Embedded Software into the Target System

- In-Circuit Emulators
  - emulators give you debugging capabilities similar to standard desktop software debuggers.
    - Set breakpoints
    - Examine the contents of memory and registers
    - See the source code
    - Resume execution
    - Single-step through the code.

Getting Embedded Software into the Target System

- In-Circuit Emulators
  - Many emulators have a feature called overlay memory.
  - There are one or more blocks of memory inside the emulator,the emulated microprocessor can use instead of the memory on the target system.
  - You can tell the emulator the address ranges for ROM or RAM instead of the memory on the target.

Getting Embedded Software into the Target System

- Flash
  - If your target stores its program in flash memory, you have to place the flash in a socked and treat it like an EPROM.
  - If your target has a serial port, a network connection, or some other mechanism for communicating with the outside world, flash memories open up another possibility:
    - In-circuit programming

Getting Embedded Software into the Target System

- Flash
  - You can load new software into your system for debugging——without pulling chips out of sockets.
  - Downloading new software is much faster.
  - Allow you customers to load new versions of the software.
  - And so on.

Getting Embedded Software into the Target System

- Monitors
  - That is a program that resides in the target ROM and knows how to load new programs onto the system.
  - A typical monitor allows you to send your software across a serial port, stores that software in the target RAM, and then runs it.

Summary

- Embedded software development is typically done on a host machine, different from the target machine on which the software will eventually be shipped to customers.
- A tool chain for developing embedded software typically contains a cross-compiler, a cross-assembler, a linker/locator, and a method for loading the software into the target machine.

Summary

- A cross-compiler understands the same C language as a native compiler, but its output uses the instruction set of the target microprocessor.
- A cross-assembler understands an assembly language that is specific to your target microprocessor and outputs instructions for that microprocessor.

Summary

- A linker/locator combines separately compiled and assembled modules into an executable image. In addition, it places code, data, startup code, constand string, and so on ato suitable addresses in the ROM and RAM.
- Linker/locators use segments to decide where to put different parts of the code and data.

Summary

- Linker/locators produce output in a variety of formats; it is up to you to ensure that your linker/locator’s output is compatible with the tools you use for loading software into your target.
- You must find a way to load you software into the target system for testing. The most common ways include PROM programmers, ROM emulators, in-circuit emulators, flash , and monitors.