Code Development

Speaker: Juin-Nan Liu

Adopted from National Chiao-Tung University
IP Core Design

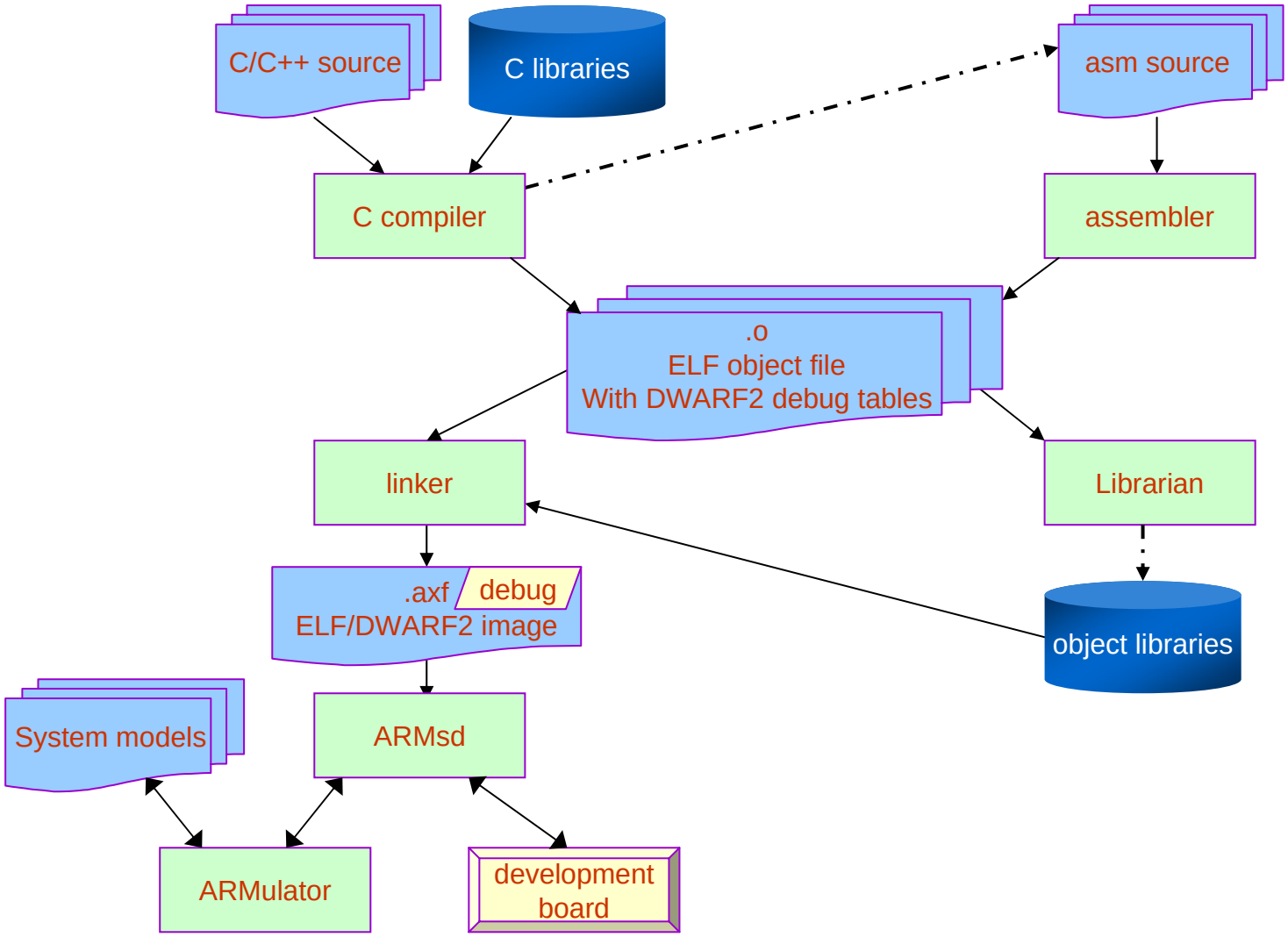
Goal of This Lab

- Familiarize with ARM software development tools: ARM Development Suite (ADS)
  - Project management
  - Configuring the settings of build targets for your project
- Writing code for ARM-based platform design
- Mixed instruction sets, ARM and Thumb interworking, is learned to balance the performance and code density of an application.

Outline

- ***Basic Code Development***
- ARM/Thumb Interworking
- Lab1 – Code Development

The Structure of ARM Tools



DWARF: Debug With Arbitrary Record Format **ELF: Executable and linking format**

Main Components in ADS (1/2)

- ANSI C compilers – armcc and tcc
- ISO/Embedded C++ compilers – armcpp and tcpp
- ARM/Thumb assembler - armasm
- Linker - armlink
- Project management tool for windows - CodeWarrior
- Instruction set simulator - ARMulator
- Debuggers - AXD, ADW, ADU and armsd
- Format converter - fromelf
- Librarian – armar
- ARM profiler - armprof

ADS: ARM Developer Suite

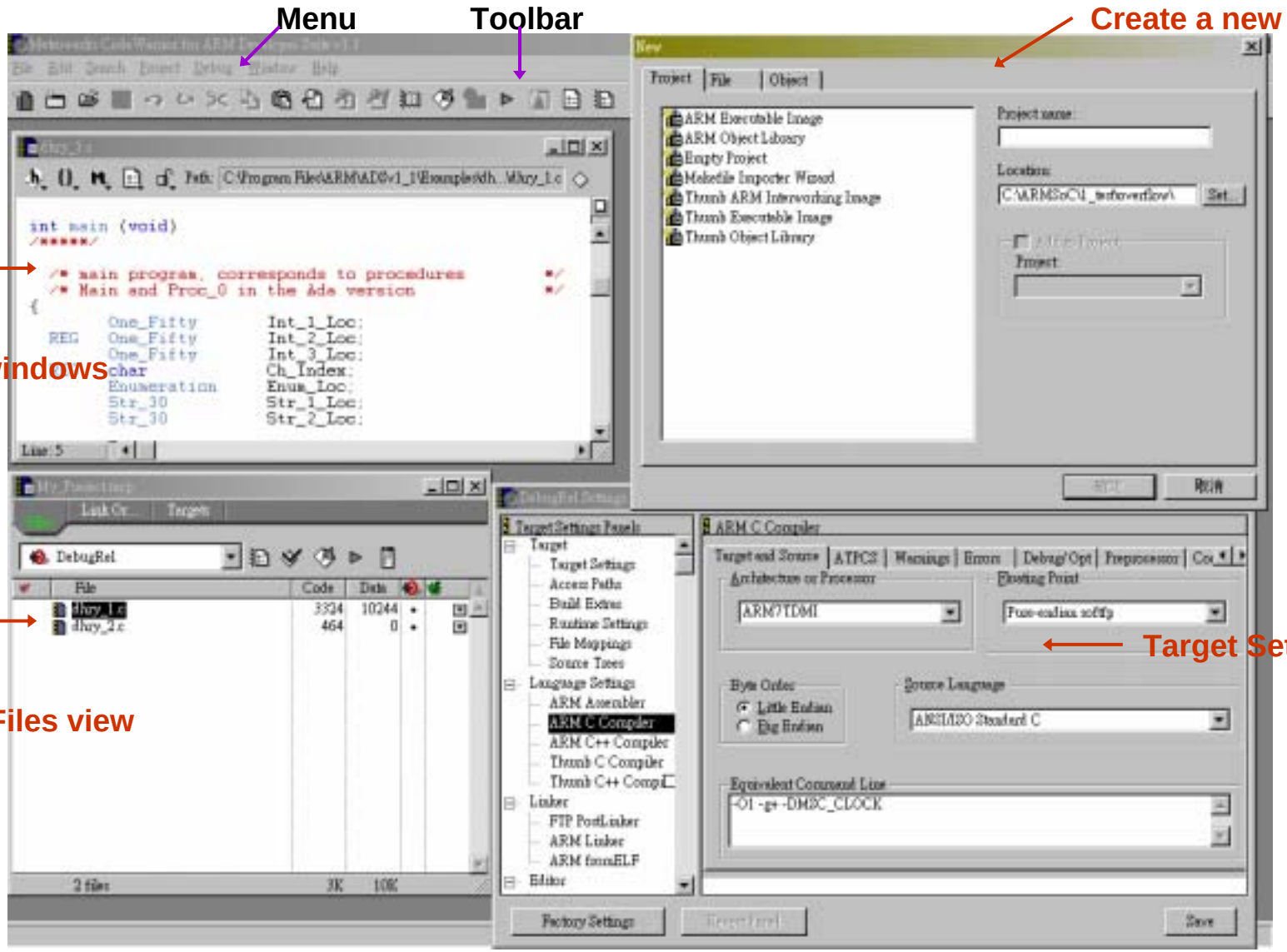
Main Components in ADS (2/2)

- C and C++ libraries
- ROM-based debug tools (ARM Firmware Suite, AFS)
- Real Time Debug and Trace support
- Support for all ARM cores and processors including ARM9E, ARM10, Jazelle, StrongARM and Intel Xscale

View in CodeWarrier

- The CodeWarrior IDE provides a simple, versatile, graphical user interface for managing your software development projects.
- Develop C, C++, and ARM assembly language code
- targeted at ARM and Thumb processors.
- It speeds up your build cycle by providing:
  - comprehensive project management capabilities
  - code navigation routines to help you locate routines quickly.

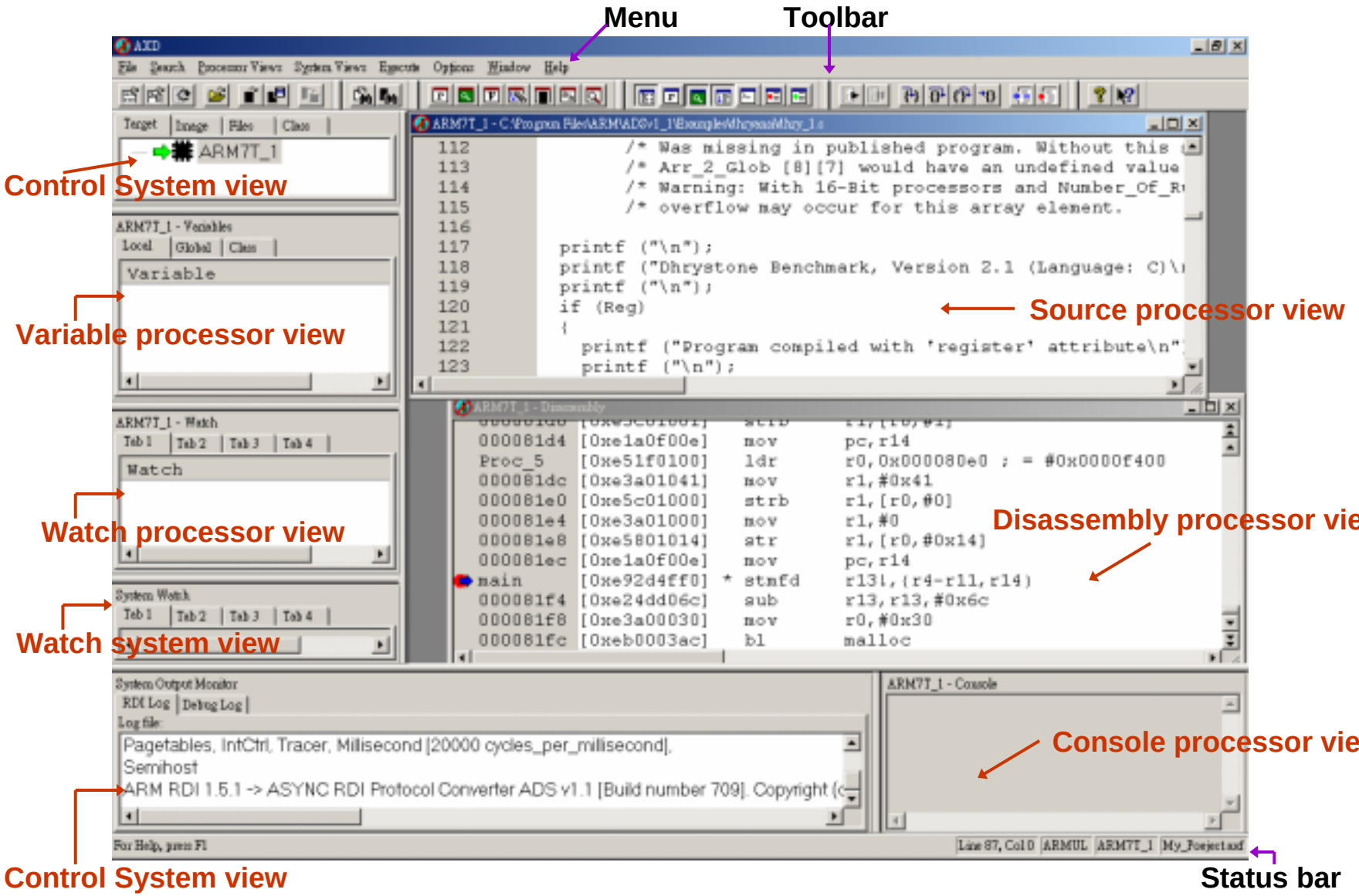
CodeWarrior Desktop



Views in AXD

- Various views allow you to **examine** and **control** the processes you are debugging.
- In the main menu bar, two menus contain items that display views:
  - The items in the **Processor Views menu** display views that apply to the **current processor only**
  - The items in the **System Views menu** display views that apply to the entire, possibly **multiprocessor**, target system

AXD Desktop


Menu
Toolbar
Control System view
Variable processor view
Watch processor view
Watch system view
Source processor view
Disassembly processor view
Console processor view
Control System view
Status bar

ARM Emulator: ARMulator (1/2)

- A suite of programs that models the behavior of various ARM processor cores and system architecture in software on a host system
- Can be operates at various levels of accuracy
  - Instruction accurate
  - Cycle accurate
  - Timing accurate

ARM Emulator: ARMulator (2/2)

- Benchmarking before hardware is available
  - Instruction count or number of cycles can be measured for a program.
  - Performance analysis.
- Run software on ARMulator
  - Through ARMsd or ARM GUI debuggers, e.g., AXD
  - The processor core model incorporates the remote debug interface, so the processor and the system state are visible from the ARM symbolic debugger
  - Supports a C library to allow complete C programs to run on the simulated system

ARM Symbolic Debugger

- ARMsd: ARM and Thumb symbolic debugger
  - can single-step through C or assembly language sources,
  - set break-points and watch-points, and
  - examine program variables or memory
- It is a front-end interface to debug program running either
  - under emulation (on the ARMulator) or
  - remotely on a ARM development board (via a serial line or through JTAG test interface)
- It allows the setting of
  - breakpoints, addresses in the code
  - watchpoints, memory address if accessed as data address

  ➔ cause exception to halt so that the processor state can be examined

Basic Debug Requirements

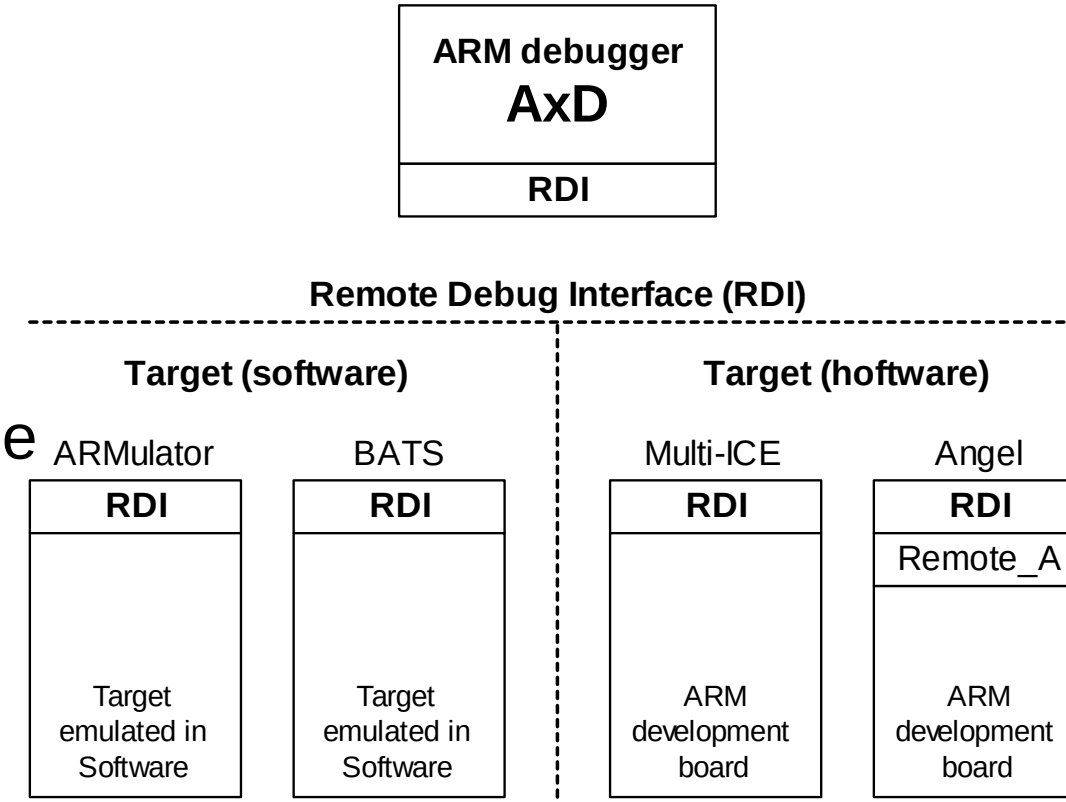
- Control of program execution
  - set watchpoints on interesting data accesses
  - set breakpoints on interesting instructions
  - single step through code
- Examine and change processor state
  - read and write register values
- Examine and change system state
  - access to system memory
    - download initial code

Debugger (1/2)

- A debugger is software that enables you to make use of a debug agent in order to examine and control the execution of software running on a debug target
- Different forms of the debug target
  - early stage of product development, software
  - prototype, on a PCB including one or more processors
  - final product
- The debugger issues instructions that can
  - load software into memory on the target
  - start and stop execution of that software
  - display the contents of memory, registers, and variables
  - allow you to change stored values
- A debug agent performs the actions requested by the debugger, such as
  - setting breakpoints
  - reading from / writing to memory

Debugger (2/2)

- Examples of debug agents
  - Multi-ICE
  - Embedded ICE
  - ARMulator
  - BATS
  - Angle
- Remote Debug Interface (RDI) is an open ARM standard procedural interface between a debugger and the debug agent



Program Design

- Start with understanding the requirements, translate the requirements into an unambiguous specifications
- Define a program structure, the data structure and the algorithms that are used to perform the required operations on the data
- The algorithms may be expressed in pseudo-code
- Individual modules should be coded, tested and documented
- Nearly all programming is based on high-level languages, however it may be necessary to develop small software components in assembly language to get the best performance

Outline

- Basic Code Development
- ***ARM/Thumb Interworking***
- Lab1 – Code Development

ARM Instruction Sets

- ARM processor is a 32-bit architecture, most ARM's implement two instruction sets
  - 32-bit **ARM** instruction set
  - 16-bit **Thumb** instruction set

ARM and Thumb Code Size

Simple C routine

```
if (x>=0)
    return x;
else
    return -x;
```

The equivalent ARM assembly

```
labs    CMP     r0,#0     ;Compare r0 to zero
        RSBLT   r0,r0,#0  ;If r0<0 (less than=LT) then do r0= 0-r0
        MOV     pc,lr     ;Move Link Register to PC (Return)
```

The equivalent Thumb assembly

```
        CODE16 ;Directive specifying 16-bit (Thumb) instructions
iabs    CMP     r0,#0     ;Compare r0 to zero
        BGE     return    ;Jump to Return if greater or
                          ;equal to zero
        NEG     r0,r0     ;If not, negate r0
return  MOV     pc,lr     ;Move Link register to PC (Return)
```

Code	Instructions	Size (Bytes)	Normalised
ARM	3	12	1.0
Thumb	4	8	0.67

The Need for Interworking

- The code density of Thumb and its performance from narrow memory make it ideal for the bulk of C code in many systems. However there is still a need to change between ARM and Thumb state within most applications:
  - ARM code provides better performance from wide memory
    - therefore ideal for speed-critical parts of an application
  - some functions can only be performed with ARM instructions, e.g.
    - access to CPSR (to enable/disable interrupts & to change mode)
    - access to coprocessors
  - exception Handling
    - ARM state is automatically entered for exception handling, but system specification may require usage of Thumb code for main handler
  - simple standalone Thumb programs will also need an ARM assembler header to change state and call the Thumb routine

Interworking Instructions

- Interworking is achieved using the Branch Exchange instructions
  - in Thumb state

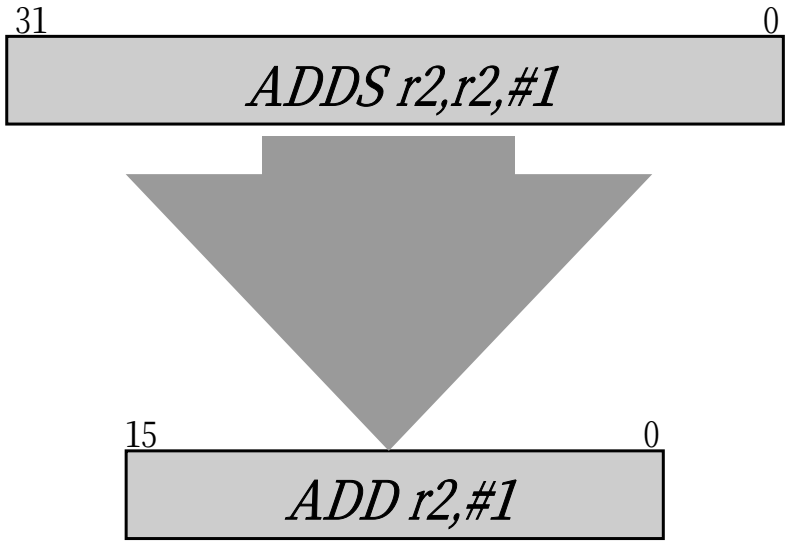
    ```
    BX Rn
    ```

  - in ARM state (on Thumb-aware cores only)

    ```
    BX<condition> Rn
    ```

  where Rn can be any registers (r0 to r15)
- This performs a branch to an absolute address in 4GB address space by copying Rn to the program counter
- Bit 0 of Rn specifies the state to be changed to

Switching between States



32-bit ARM instruction

For most instruction generated by compiler:

- Conditional execution is not used
- Source and destination registers identical
- Only Low registers used
- Constants are of limited size
- Inline barrel shifter not used

16-bit Thumb instruction

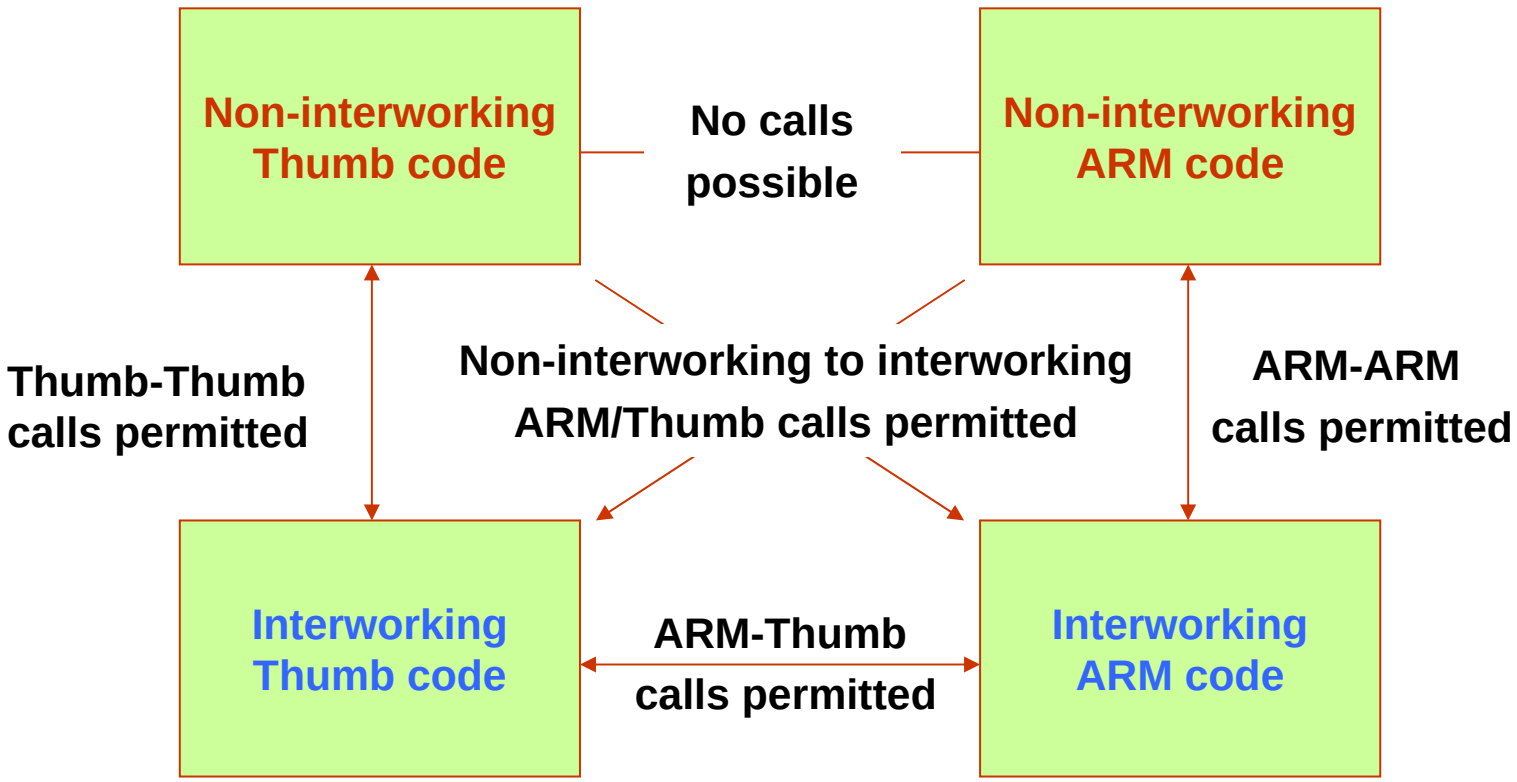
Example

```
;start off in ARM state
            CODE32
            ADR r0,Into_Thumb+1     ;generate branch target
                                    ;address & set bit 0,
                                    ;hence arrive Thumb state
            BX r0                   ;branch exchange to Thumb
            …
            CODE16                  ;assemble subsequent as
                                    ;Thumb
Into_Thumb  …
            ADR r5,Back_to_ARM      ;generate branch target to
                                    ;word-aligned address,
                                    ;hence bit 0 is cleared.
            BX r5                   ;branch exchange to ARM
            …
            CODE32                  ;assemble subsequent as
                                    ;ARM
Back_to_ARM …
```

ARM/Thumb Interworking between C/C++ and ASM

- C code compiled to run in one state may call assembler to execute in the other state, and vice-versa.
  - If the callee is in **C**, compile it using **–apcs /interwork**
  - If the callee is in **ASM**, assemble it using **–apcs /interwork** and return using **BX LR**
- Any assembler code used in this manner must conform to ATPCS where appropriate, e.g., function parameters passed in r0-r3 & r12 corruptible

Interworking Calls



Modules that are compiled for interworking generate slightly larger code, typically 2% larger for Thumb and less than 1% larger for ARM.

Outline

- Basic Code Development
- ARM/Thumb Interworking
- ***Lab1 – Code Development***

Lab 1: Code Development

- Goal
  - How to create an application using ARM Developer Suite (ADS)
  - How to change between ARM state and Thumb state when writing code for different instruction sets
- Principles
  - Processor's organization
  - ARM/Thumb Procedure Call Standard (ATPCS)
- Guidance
  - Flow diagram of this Lab
  - Preconfigured project stationery files
- Steps
  - Basic software development (tool chain) flow
  - ARM/Thumb Interworking
- Requirements and Exercises
  - See next slide
- Discussion
  - The advantages and disadvantages of ARM and Thumb instruction sets.

Lab 1: Code Development (cont')

- ARM/Thumb Interworking
  - Exercise 1: C/C++ for "Hello" program
    - Caller: Thumb
    - Callee: ARM
  - Exercise 2: Assembly for "SWAP" program, w/wo veneers
    - Caller: Thumb
    - Callee: ARM
  - Exercise 3: Mixed language for "SWAP" program, ATPCS for parameters passing
    - Caller: Thumb in Assembly
    - Callee: ARM in C/C++

References

[1] http://twins.ee.nctu.edu.tw/courses/ip_core_02/index.html
[2] ADS_AssemblerGuide_A.pdf
[3] ADS_CodeWarriorIDEGuide_C.pdf
[4] ADS_DeveloperGuide_C.pdf
[5] ADS_GettingStarted_C.pdf
[6] ADS_LINKERGUIDE_A.pdf