

Programming the ARM Microprocessor for Embedded Systems

Ajay Dudani
ajaydudani@gmail.com

Copyrights

- This document is available under the terms of Creative Commons Attribution-NonCommercial-NoDerivs 2.5 License
- In simpler terms, you may
 - Reproduce and redistribute this document as-is for non-commercial use

Goals

- Develop a good understanding of execution of ARM processor required to develop and debug embedded software with or without an operating system
- Students should have prior knowledge about programming, operating system concepts and understanding of embedded systems

Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- Embedded operating system
- ARM Caches
- Memory management and protection
- ARM Future development

Why ARM?

- Low power consumption
- Fast execution per Watt
- Good backward compatibility
- Inexpensive
- Variety of core offerings
- ... and other advantages we will see in upcoming slides

Why ARM?

- And ...
Marketing 😊

Detailed Outline

- ARM Technology
 - History
 - The competition

Detailed Outline

- ARM Tools and products
 - Microprocessors
 - Compilers and debuggers
 - Single board computers
 - Documents and reference text

Detailed Outline

- ARM Processor
 - Programming model
 - General purpose registers
 - Program control registers
 - Pipelining
 - Memory protection

Detailed Outline

- Toolchain
 - ARM Toolchain
 - GNU Toolchain
 - Others

Detailed Outline

- ARM Instruction Set
 - Instruction set encoding
 - Conditional field
 - Data processing instructions
 - Branch instructions
 - Load-store instructions
 - Software interrupt instruction
 - Program status register instructions
 - Semaphore instructions
 - Coprocessor instructions
 - Extending instructions

Detailed Outline

- ARM Thumb Mode
 - Why Thumb?
 - Instruction set encoding
 - Branch instructions
 - Data processing instructions
 - Load and store instructions
 - Switching between ARM and Thumb mode

Detailed Outline

- ARM Exceptions and Interrupts
 - Exception vector table
 - Exception modes
 - Exception and interrupt handling

Detailed Outline

- ARM Firmware
 - Bootloader
 - Standalone code
 - Example of initialization

Detailed Outline

- ARM Caches
 - Memory hierarchy
 - Cache policy
 - Flushing policy
 - Software performance

Detailed Outline

- ARM MMU and MPU
 - Protected memory
 - Example of memory protection
 - Virtual memory concepts
 - Example of virtual memory system

Outline

- **ARM Technology Overview**
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- ARM Caches
- Memory management and protection
- ARM Future development

ARM Technology Overview

- ARM: “The Architecture For The Digital World”
- ARM is a physical hardware design and intellectual property company
- ARM licenses its cores out and other companies make processors based on its cores
- ARM also provides toolchain and debugging tools for its cores

ARM Technology Overview (2)

Companies licensing ARM IP:

3Com

Agilent Technologies

Altera

Epson

Freescale

Fijitsu

NEC

Nokia

Intel

IBM

Microsoft

Motorola

Panasonic

Qualcomm

Sharp

Sanyo

Sun Microsystems

Sony

Symbian

Texas Instruments

Toshiba

Wipro

... and many more

Source: ARM Website

Copyright (c) 2006 by Ajay Dudani

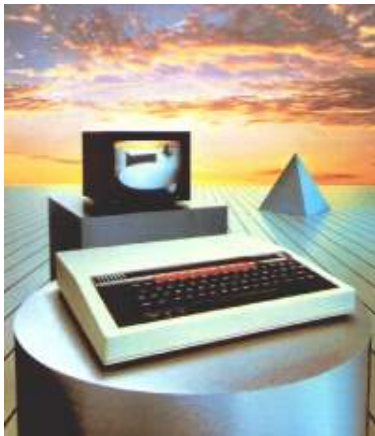
May not be reproduced or redistributed for commercial use.

ARM History

- Acorn Computer Group developed world's first RISC processor in 1985
- Roger Wilson and Steve Furber were the principle developers
- ARM (Advanced RISC Machines) was a spin out from Acorn in 1990 with goal of defining a new microprocessor standard

ARM History (2)

- Acorn Computer Group



Source: Wikipedia

Copyright (c) 2006 by Ajay Dudani
May not be reproduced or redistributed for commercial use.

ARM History (3)

- ARM delivered ARM6 in 1991
 - Introduced 32 bit addressing support
 - New instruction for program status registers
 - Variant used in Apple Newton PDA
- By 1996 ARM7 was being widely used
 - Microsoft started port of WinCE to ARM
 - Added multimedia extensions
- Exponential growth from then on...

ARM and StrongARM

- Intel gained certain IP from ARM as part of lawsuit settlement and modified ARM architecture branding it as StrongARM
- StrongARM name was changed to XScale
 - Processor SA1000 , SA1100
- XScale is close to ARMv5 instruction set
- XScale division of Intel was sold to Marvel Inc. in 2006

ARM Today

- ARM7xxx
 - 3 stage pipeline
 - Integer processor
 - MMU support for WinCE, Linux and Symbian
 - Used in entry level mobiles, mp3 players, pagers
- ARM9xxx
 - 5 stage pipeline
 - Separate data and instruction cache
 - Higher end mobile and communication devices
 - Telematic and infotainment systems
 - ARM and Thumb instruction set

ARM Today (2)

- ARM11xxx
 - 7 stage pipeline
 - Trustzone security related extensions
 - Reduced power consumption
 - Speed improvements
 - More DSP and SIMD extensions
 - Used in PDA, smartphones, industrial controllers, mobile gaming

ARM Processor Family

- ARM has devised a naming convention for its processors
- Revisions: ARMv1, v2 ... v6, v7
- Core implementation:
 - ARM1, ARM2, ARM7, StrongARM, ARM926EJ, ARM11, Cortex
- ARM11 is based on ARMv6
- Cortex is based on ARMv7

ARM Processor Family (2)

- Differences between cores
 - Processor modes
 - Pipeline
 - Architecture
 - Memory protection unit
 - Memory management unit
 - Cache
 - Hardware accelerated Java
 - ... and others

ARM Processor Family (3)

- Examples:
 - ARM7TDMI
 - No MMU, No MPU, No cache, No Java, Thumb mode
 - ARM922T
 - MMU, No MPU, 8K+8K data and instruction cache, No Java, Thumb mode
 - ARM1136J-S
 - MMU, No MPU, configurable caches, with accelerated Java and Thumb mode

ARM Processor Family (4)

- Naming convention
- ARM [x][y][z][T][D][M][I][E][J][F][S]
 - x – Family
 - y – memory management/protection
 - z – cache
 - T – Thumb mode
 - D – JTAG debugging
 - M – fast multiplier
 - I – Embedded ICE macrocell
 - E – Enhanced instruction (implies TDMI)
 - J – Jazelle, hardware accelerated Java
 - F – Floating point unit
 - S – Synthesizable version

Outline

- ARM Technology
- **ARM Tools & Products**
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- ARM Caches
- Memory management and protection
- ARM Future development

ARM Tools & Products



Disclaimer: All owners own their respective trademarks

Copyright (c) 2006 by Ajay Dudani
May not be reproduced or redistributed for commercial use.

ARM Chips

- ARM Ltd
 - Provides ARM cores
 - Intellectual property
- Analog Devices
 - ADuC7019, ADuC7020, ADuC7021, ADuC7022, ADuC7024, ADuC7025, ADuC7026, ADuC7027, ADuC7128, ADuC7129
- Atmel
 - AT91C140, AT91F40416, AT91F40816, AT91FR40162
- Freescale
 - MAC7101, MAC7104, MAC7105, MAC7106
- Samsung
 - S3C44B0X, S3C4510B
- Sharp
 - LH75400, LH75401, LH75410, LH75411
- Texas Instruments
 - TMS470R1A128, TMS470R1A256, TMS470R1A288
- And others...

ARM Development Tools

- Compilers:
 - GNU Compiler
 - ADS from ARM (older version)
 - RVCT Real View compiler tools from ARM
 - 3rd Party
- Debugging
 - GNU gdb
 - Lauterbach JTAG/Trace32 tools
 - ETM hardware debugging & profiling modules
 - Windriver tools

ARM Single Board Computers

- TS-7200 ARM Single board computer
 - 200 MHz ARM9 processor with MMU
 - 32 MB RAM
 - 8 MB Flash
 - Compact flash
 - 10/100 Ethernet



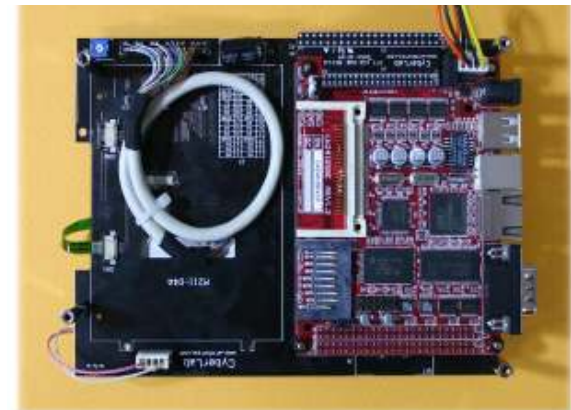
ARM Single Board Computers (2)

- Cirrus Logic ARM CS98712
 - 16 MB RAM
 - 1MB Flash
 - 1 Serial port
 - LED lights



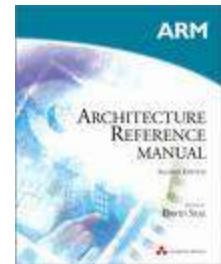
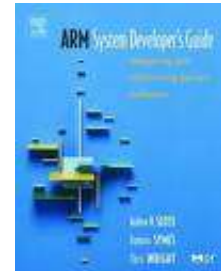
ARM Single Board Computers (3)

- LN24x0/LP64
 - 200 Mhz ARM9 processor
 - LCD controller
 - Touchscreen
 - USB, IrDA ports
 - HDD and CD-ROM support
 - Speaker
 - JTAG ports



Recommended Text

- “ARM System Developer’s Guide”
 - Sloss, et. al.
 - ISBN 1-55860-874-5
- “ARM Architecture Reference Manual”
 - David Seal
 - ISBN 0-201-737191
 - Softcopy available at www.arm.com
- “ARM system-on-chip architecture”
 - Steve Fuber
 - ISBN 0-201-67519-6



Outline

- ARM Technology Overview
- ARM Tools & Products
- **ARM Processor**
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- ARM Caches
- Memory management and protection
- ARM Future development

ARM Design Philosophy

- ARM core uses RISC architecture
 - Reduced instruction set
 - Load store architecture
 - Large number of general purpose registers
 - Parallel executions with pipelines
- But some differences from RISC
 - Enhanced instructions for
 - Thumb mode
 - DSP instructions
 - Conditional execution instruction
 - 32 bit barrel shifter

ARM Programming Model

- $A = B + C$
- To evaluate the above expression
 - Load A to a general purpose register R1
 - Load B to a general purpose register R2
 - Load C to a general purpose register R3
 - ADD R1, R2, R3
 - Store R1 to A

Registers

- ARM has a load store architecture
- General purpose registers can hold data or address
- Total of 37 registers each 32 bit wide
- There are 18 active registers
 - 16 data registers
 - 2 status registers

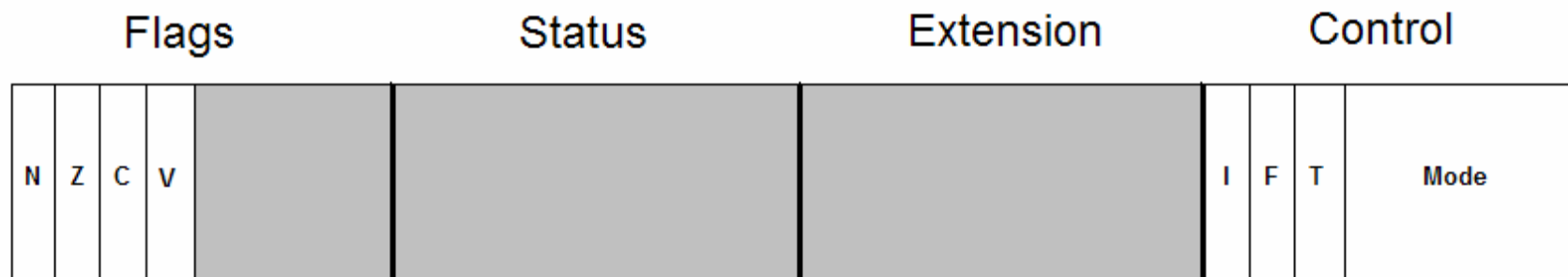
Registers (2)

- Registers R0 thru R12 are general purpose registers
- R13 is used as stack pointer (sp)
- R14 is used as link register (lr)
- R15 is used as a program counter (pc)
- CPSR – Current program status register
- SPSR – Stored program status register

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (sp)
R14 (lr)
R15 (pc)
CPSR
SPSR

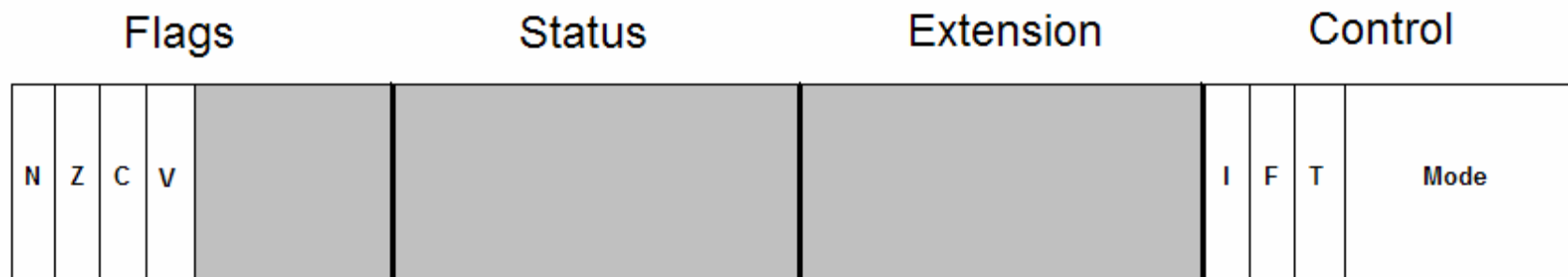
Registers (3)

- Program status register
 - CPSR is used to control and store CPU states
 - CPSR is divided in four 8 bit fields
 - Flags
 - Status
 - Extension
 - Control



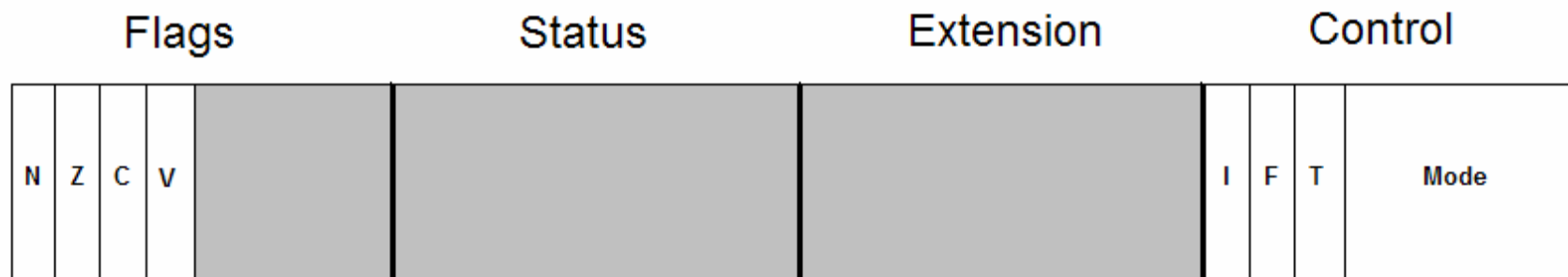
Registers (4)

- Program status register flags
 - N:1 – Negative result
 - Z:1 – Result is zero
 - C:1 – Carry in addition operation
 - C:0 – Borrow in subtraction operation
 - V:1 – Overflow or underflow



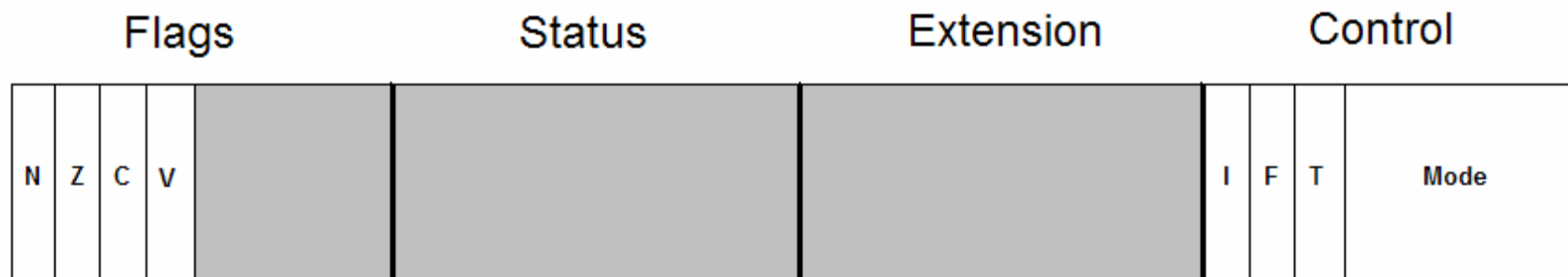
Registers (5)

- Program status register controls
 - I:1 – IRQ interrupts disabled
 - F:1 – FIQ interrupts disabled
 - T:0 – ARM Mode
 - T:1 – Thumb Mode



Registers (6)

- Program status register control modes
 - 0b10000 – User mode
 - 0b10001 – FIQ mode
 - 0b10010 – IRQ mode
 - 0b10011 – Supervisor mode
 - 0b10111 – Abort mode
 - 0b11011 – Undefined mode
 - 0b11111 – System mode



Processor Modes

- Processor modes are execution modes which determines active registers and privileges
- List of modes
 - Abort mode
 - Fast interrupt mode
 - Interrupt mode
 - Supervisor mode
 - System mode
 - Undefined mode
 - User mode
- All except User mode are privileged modes
 - User mode is used for normal execution of programs and applications
 - Privileged modes allow full read/write to CPSR

Banked Registers

- Of total 37 registers only 18 are active in a given register mode

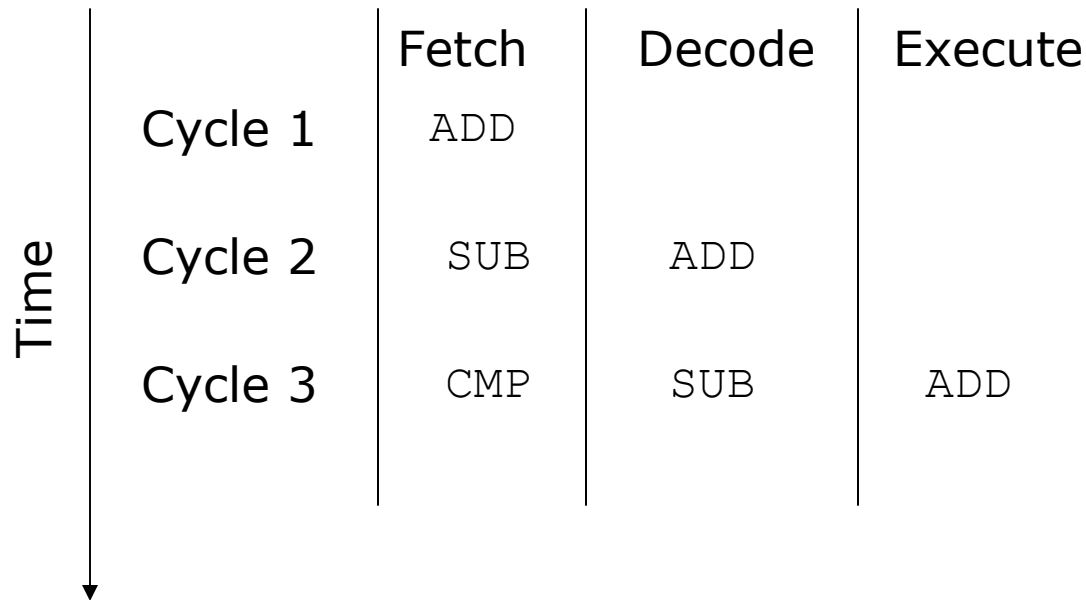
User/System	Supervisor	Abort	FIQ	IRQ	Undefined
R0					
R1					
R2					
R3					
R4					
R5					
R6					
R7					
R8			R8_fiq		
R9			R9_fiq		
R10			R10_fiq		
R11			R11_fiq		
R12			R12_fiq		
R13 (sp)	R13_svc (sp)	R13_abt (sp)	R13_fiq (sp)	R13_irq (sp)	R13_und (sp)
R14 (lr)	R14_svc (lr)	R14_abt (lr)	R14_fiq (lr)	R14_irq (lr)	R14_und (lr)
R15 (pc)					
CPSR					
SPSR	SPSR_svc	SPSR_abt	SPSR_fiq	SPSR_irq	SPSR_und

Pipeline

- Pipelining is breaking down execution into multiple steps, and executing each step in parallel
- Basic 3 stage pipeline
 - Fetch – Load from memory
 - Decode – Identify instruction to execute
 - Execute – Process instruction and write back result

Pipeline (2)

-



Pipeline (3)

- ARM7 has a 3 stage pipeline
 - Fetch, Decode, Execute
- ARM9 has a 5 stage pipeline
 - Fetch, Decode, Execute, Memory, Write
- ARM10 has a 6 stage pipeline
 - Fetch, Issue, Decode, Execute, Memory, Write

Pipeline (4)

- In theory, each instruction is one instruction cycle
- In practice, there is interdependency between instructions
 - Solution: instruction scheduling

Memory Protection

- Two modes for ARM memory protection
 - Unprotected mode
 - No hardware protection, software does protection of data between tasks
 - Protected mode
 - Hardware protects areas of memory and raises exceptions when policy is violated
- ARM divides memory to regions and programmer can set attributes on regions
- ARM provides mechanisms to define and set attributes of regions programmatically

Memory Management

- ARM supports memory management and virtual memory
- Programmatically access translation look-aside buffers
- ARM memory management unit also supports Fast Context Switching Extensions that optimizes use of caches in multitasking environments
- Details in later section

Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- **ARM Toolchain**
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- ARM Caches
- Memory management and protection
- ARM Future development

Toolchain

- **GNU Tools**
 - `gcc` – Front end to GNU compiler
 - `binutils` – Binary tools
 - `ld` – GNU Linker
 - `as` – GNU assembler
 - And others
 - `gdb` – GNU Debugger
 - `uClib` – Small footprint C Library

Toolchain (2)

- GCC
 - Invoked language specific modules
 - Invoked assembler and linker
 - `arm-elf-gcc` **command**
 - `arm-elf-gcc test.c -o test`
 - `arm-elf-gcc test.S -o test`

Toolchain (3)

- GCC ARM specific options
 - `mmapcs-frame`: Generate ARM procedure call compliant stack frame
 - `mbig-endian`: Generate big endian code
 - `mno-alignment-traps`: Generate code that assumes that MMU does not trap on handling misaligned data
 - `mcpu=name` : Specify CPU name; gcc can determine what instructions it can use to generate output accordingly
 - `mthumb`: Generate code for ARM Thumb mode
 - `msoft-float`: Generate code assuming floating point hardware is not present. Do floating point operation optimization in software
 - Refer to GCC manual page for more on compiler options

Inline assembly

- Developer can insert ARM assembly code in C code – for example

```
printf ("Hello ARM GCC");  
__asm__ ("ldr r15, r0");  
printf ("Program may have crashed");
```

Above code will corrupt program counter, so use inline assembly carefully

Also, it may lead to non portable code

Inline assembly (2)

- Developer can force use of certain registers using extended assembly

```
asm ( assembler template :  
      output operands /* optional */ :  
      input operands /* optional */ :  
      list of clobbered registers /* optional */ );
```

- Example:

```
int a = 10, b;  
__asm__ ("mov %0, %1" :  
        "=r" (b) :  
        "=r" (a) );
```

Inline assembly (3)

- Developer can request variable to be assigned to specific register

```
register int regVar __asm__ ("%r4");
```

- Can be used with local and global variables
- Need `-ffixed-<reg>` compiler option for global variables
- Refer to GCC Inline Assembly reference for more examples

GNU Toolchain

- Reference:
 - www.gnuarm.org
 - www.sourceware.org/binutils/
 - www.gnu.org/software/gdb/
 - www.sourceware.org/insight/
 - www.uclibc.org
 - GCC improvements for ARM
 - www.inf.u-szeged.hu/gcc-arm/

ARM Toolchain

- ADS: ARM Developer Suite is older version of compiler, assembler and linker tools from ARM
- RCVT: Latest ARM compiler, assembler and linker

ARM Toolchain

- ARM also provides support for RealView tools it acquired as part of Keil acquisition
- JTAG Support: ARM provides debugging tools to be used with JTAG supported hardware
- ETM Support: Embedded Trace Module is hardware debug unit that extends on-target debugging capabilities by providing extra memory and registers for debugging purpose
- Refer to www.arm.com and www.keil.com/arm/ for details on ARM tools

Other Toolchains

- GNU X-Tools
 - www.microcross.com
- IAR ARM Kit
 - www.iar.com
- Intel Development Suite for XScale
 - www.intel.com

Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- **ARM Instruction set**
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- ARM Caches
- Memory management and protection
- ARM Future development

ARM Instruction Set

- Overview

ARM Instruction Set (2)

- Condition fields

ARM Instruction Set (3)

- Add
- Subtract
- Multiply

ARM Instruction Set (4)

- Bit shifting

ARM Instruction Set (5)

- Status register operation

ARM Instruction Set (6)

- Semaphore instruction

ARM Instruction Set (7)

- Placeholder page

Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- **Thumb instruction set**
- ARM exception and interrupts
- ARM Firmware
- ARM Caches
- Memory management and protection
- ARM Future development

Thumb Instruction Set

- Overview of 16 bit mode

Thumb Instruction Set (2)

- Thumb Instruction set details

Thumb Instruction Set (3)

- Switching between ARM and Thumb mode

Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- **ARM exception and interrupts**
- ARM Firmware
- ARM Caches
- Memory management and protection
- ARM Future development

Exceptions

- Exception handling is a programming language or hardware mechanism to catch runtime errors
- C++ and Java support exception handling in software

```
1. void func()
2. {
3.     try
4.     {
5.         int a = 100/0;
6.     }
7.     catch(...)
8.     {
9.         cout << "Caught exception" << endl;
10.        return;
11.    }
12.    cout << "No exception detected!" << endl;
13.    return;
14. }
```

Exceptions (2)

- ARM support 7 exception modes
- Developers can write custom exception handlers to deal with exception conditions
- For example:
 - Consider a system that crashes if PC is corrupted. This will cause an exception. In corresponding exception handler, programmer can save state of all registers to file system for debugging and reset

Exceptions (3)

- ARM Exceptions in order of priority
 - Reset
 - Data abort
 - FIQ
 - IRQ
 - Prefetch abort
 - Undefined instruction
 - Software interrupt

Reset exception

- Highest priority exception
- The reset handler runs in supervisor mode
- Handler is generally located at 0x00000000
- In Reset handler, FIQ and IRQ are disabled
- Other exceptions are not likely to occur when in reset handler

Reset exception (2)

- Actions performed when reset is de-asserted
 - R14_svc is set to unpredictable value
 - SPSR_svc is set to unpredictable value
 - CPSR[4:0] is 0b10011 – supervisor mode
 - CPSR[5] is 0 – execute in ARM mode
 - CPSR[6] is 1 – disable fast interrupts
 - CPSR[7] is 1 – disable normal interrupts
 - PC = 0x00000000

Data abort exception

- Data abort exception mean software is trying to read/write an illegal memory location
- Data abort has higher priority than FIQ
- While handling this more, IRQ is disabled
- The abort handler should not cause further aborts
 - Consider case of prefetch abort in abort handler
 - This will cause abort handler to be reentered
- Abort handler is generally located at 0x00000010

Data abort exception (2)

- Actions performed on data abort
 - `R14_abt` = address of abort instruction + 8
 - `SPSR_svc` = `CPSR`
 - `CPSR[4:0]` is `0b10111` – abort mode
 - `CPSR[5]` is `0` – execute in ARM mode
 - `CPSR[6]` is unchanged
 - `CPSR[7]` is `1` – disable normal interrupts
 - `PC` = `0x00000010`

Fast interrupt

- FIQ exception mode exists if developer wants to handle certain interrupts faster
- Additional banked registers in FIQ mode make execution fast
- Higher priority in IRQ
- Disabled IRQ and FIQ
- Default ARM cores do not handle nested interrupts
- FIQ handler is generally located at 0x0000001C

Fast interrupt (2)

- Actions performed on fast interrupt
 - $R14_fiq = \text{address of next instruction to execute} + 4$
 - $SPSR_fiq = CPSR$
 - $CPSR[4:0]$ is $0b10111$ – FIQ mode
 - $CPSR[5]$ is 0 – execute in ARM mode
 - $CPSR[6]$ is 1 – disable fast interrupts
 - $CPSR[7]$ is 1 – disable normal interrupts
 - $PC = 0x0000001C$

Normal interrupt

- Normal interrupt is lower priority than fast interrupts
- Disabled IRQ when handling normal interrupts
- Default ARM cores do not handle nested interrupts
- IRQ handler is generally located at 0x00000018

Normal interrupt (2)

- Actions performed on normal interrupt
 - `R14_irq` = address of next instruction to execute + 4
 - `SPSR_irq` = CPSR
 - `CPSR[4:0]` is `0b10010` – IRQ mode
 - `CPSR[5]` is `0` – execute in ARM mode
 - `CPSR[6]` is unchanged
 - `CPSR[7]` is `1` – disable normal interrupts
 - `PC` = `0x00000018`

Prefetch abort

- If processor reads instruction from undefined memory, it causes a prefetch abort exception
- Prefetch abort occurs when instruction reaches execution stage of pipeline
- Disabled normal interrupts when handling prefetch abort
- Prefetch abort handler is generally located at 0x0000000C

Prefetch abort (2)

- Actions performed on prefetch abort
 - `R14_abt` = address of next instruction to **execute + 4**
 - `SPSR_irq` = `CPSR`
 - `CPSR[4:0]` is `0b10111` – abort mode
 - `CPSR[5]` is `0` – execute in ARM mode
 - `CPSR[6]` is unchanged
 - `CPSR[7]` is `1` – disable normal interrupts
 - `PC` = `0x0000000C`

Undefined instruction

- When executing coprocessor instructions, ARM waits for coprocessor to acknowledge that it can execute the instruction
- If no coprocessor can handle given instruction, undefined instruction exception is raised
- In simulators, this can be used to simulate coprocessor in software
- Undefined instruction handler can parse instructions and process them in software simulator
- Undefined instruction handler is generally located at 0x00000004

Undefined instruction (2)

- Actions performed on undefined instruction
 - R14_und = address of next instruction to execute
 - SPSR_und = CPSR
 - CPSR[4:0] is 0b11011 – undefined mode
 - CPSR[5] is 0 – execute in ARM mode
 - CPSR[6] is unchanged
 - CPSR[7] is 1 – disable normal interrupts
 - PC = 0x00000004

Software interrupt

- Software interrupt exception is used to enter supervisor mode to execute a privileged OS function
- Typically applications run in user mode and kernel in supervisor mode
- Execution of any system call will cause SWI software interrupt to change mode to supervisor mode
- Software interrupt handler is generally located at 0x00000008

Software interrupt (2)

- Actions performed on software interrupt
 - `R14_svc` = address of next instruction to execute after SWI
 - `SPSR_irq` = CPSR
 - `CPSR[4:0]` is `0b10011` – supervisor mode
 - `CPSR[5]` is `0` – execute in ARM mode
 - `CPSR[6]` is unchanged
 - `CPSR[7]` is `1` – disable normal interrupts
 - `PC` = `0x00000008`

Exception table

- **Memory layout**

- 0x00000000 – reset exception handler

- 0x00000004 – undefined instruction handler

- 0x00000008 – software interrupt handler

- 0x0000000C – prefetch abort handler

- 0x00000010 – data abort handler

- 0x00000018 – normal interrupt handler

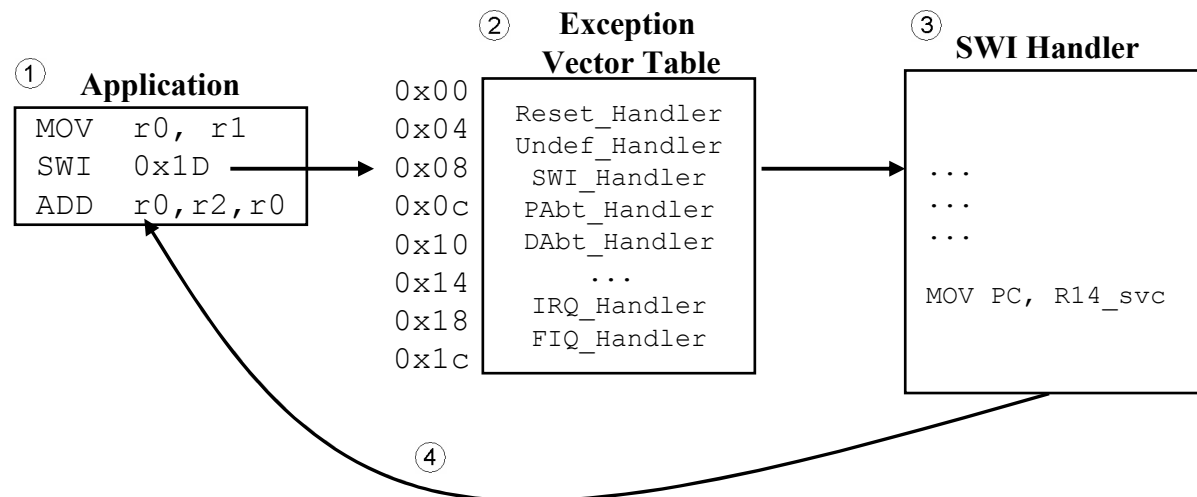
- 0x0000001C – fast interrupt handler

- **High vectors**

- Some implementation keep this table in 0xFFFF0000 to 0xFFFF001C range which is known as high vector location

Exception handling

1. Exception is raised
2. Lookup to exception table to find exception handler
3. Exception handler is executed
4. Return from exception handler



Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- **ARM Firmware**
- ARM Caches
- Memory management and protection
- ARM Future development

ARM Firmware

- “Now the earth was formless and empty. Darkness was on the surface of the deep. God's Spirit was hovering over the surface of the waters.”
“God said, “Let there be light”, and there was light.”
- For the processor, someone needs to read application and copy it into RAM to start execution
 - What address to copy it to RAM
 - Setup stack and heap
 - Transfer control to application

System initialization

- Two stages of initialization:
 1. Initialize stack, vectors and I/O system
 2. Initialize application and associated libraries
- The system set up could be
 - With an RTOS, in which case it does initialization of system environment (stack, vectors etc.). User application then starts with main() or RTOS specific entry point
 - Without an RTOS, ROM code handles transferring control to user application. User application need additional code to setup system environment

Initialization

- Execution environment:
 1. At reset
 - Processor is in SVC mode
 - Interrupts are disabled
 - Running in ARM mode
 2. Entry point on powerup
 - Use assembler directive ENTRY to specify entry point
 - ROM code usually has entry point 0x00

Initialization (2)

- Execution environment:
 3. Setup exception vector table
 - If ROM is mapped to address 0x00, it contains hard coded exception vector table
 - If ROM is mapped elsewhere, exception table is copied to RAM address 0x00
 4. Initialize memory system
 - Initialize memory management and memory protection before running any application code
 - Setup stack pointers, sp_SYS, sp_IRQ etc.
 - Initialize I/O devices. Note interrupts are still disabled
 - Change processor mode to user mode. At this stage we are ready to initialize application

Initialization (3)

- Application environment:
 1. Initialize ZI writable region with zeroes
 2. Initialize non zero data by copying initialization values
 3. Pass on control to main function

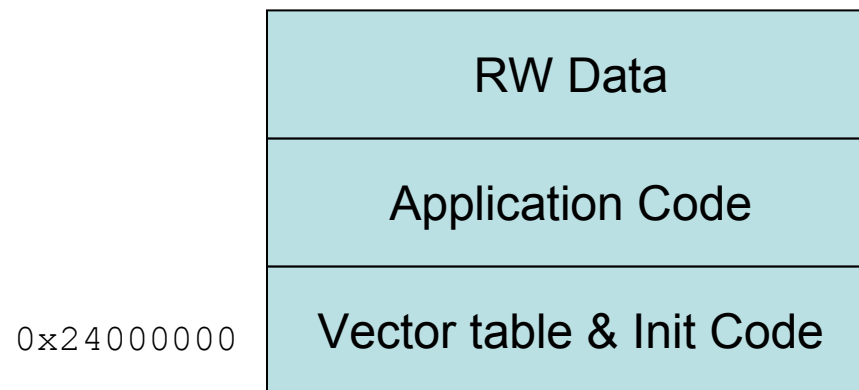
Initialization Example

- Consider user code:

```
1. int main (void)
2. {
3.     printf ("Hello ARM World");
4.     return 0;
5. }
```

Initialization Example (2)

- Consider ROM mapped at address 0x24000

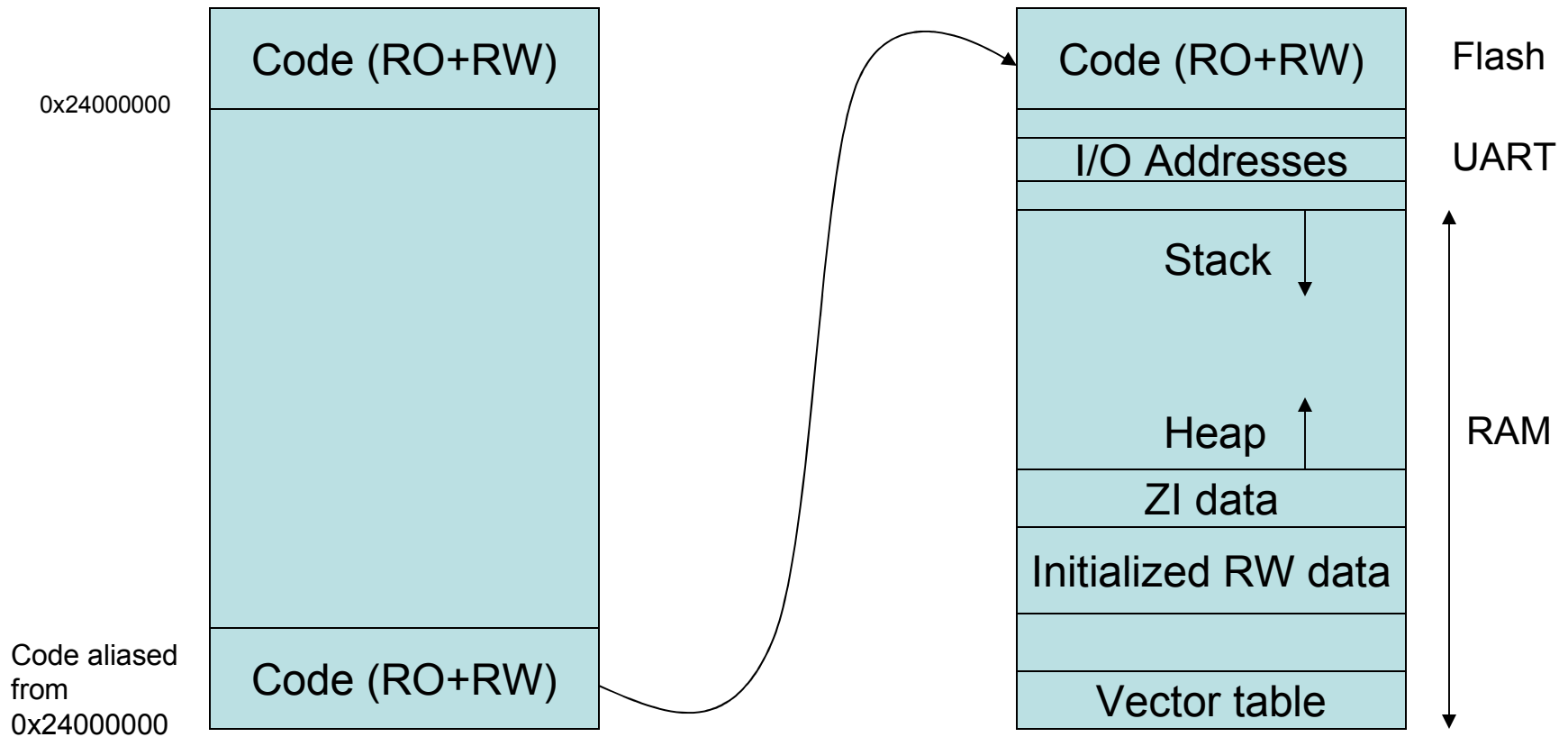


- Above image shows setup before application has been loaded

Initialization Example (3)

- At initialization, ROM code at 0x24000000 is in flash. Flash is mapped at 0x24000000 address
- On Reset, Flash will be remapped by hardware to address 0x00
 - ROM init code performs certain initialization
 - Initialized vector tables and data regions to RAM
 - Copies application code from ROM to RAM
 - Sets REMAP bit to map RAM to address 0x00

Initialization Example (4)



Firmware

- Typically the ROM code is bootloader
 - Developers configure the bootloader as per their requirements to specify memory map, bootup address, policy to copy code to RAM
- Application code can
 - Either be single binary with RTOS in which case RTOS provides basic OS services
 - For simple applications there may be no RTOS, just application compiled with simplified C library

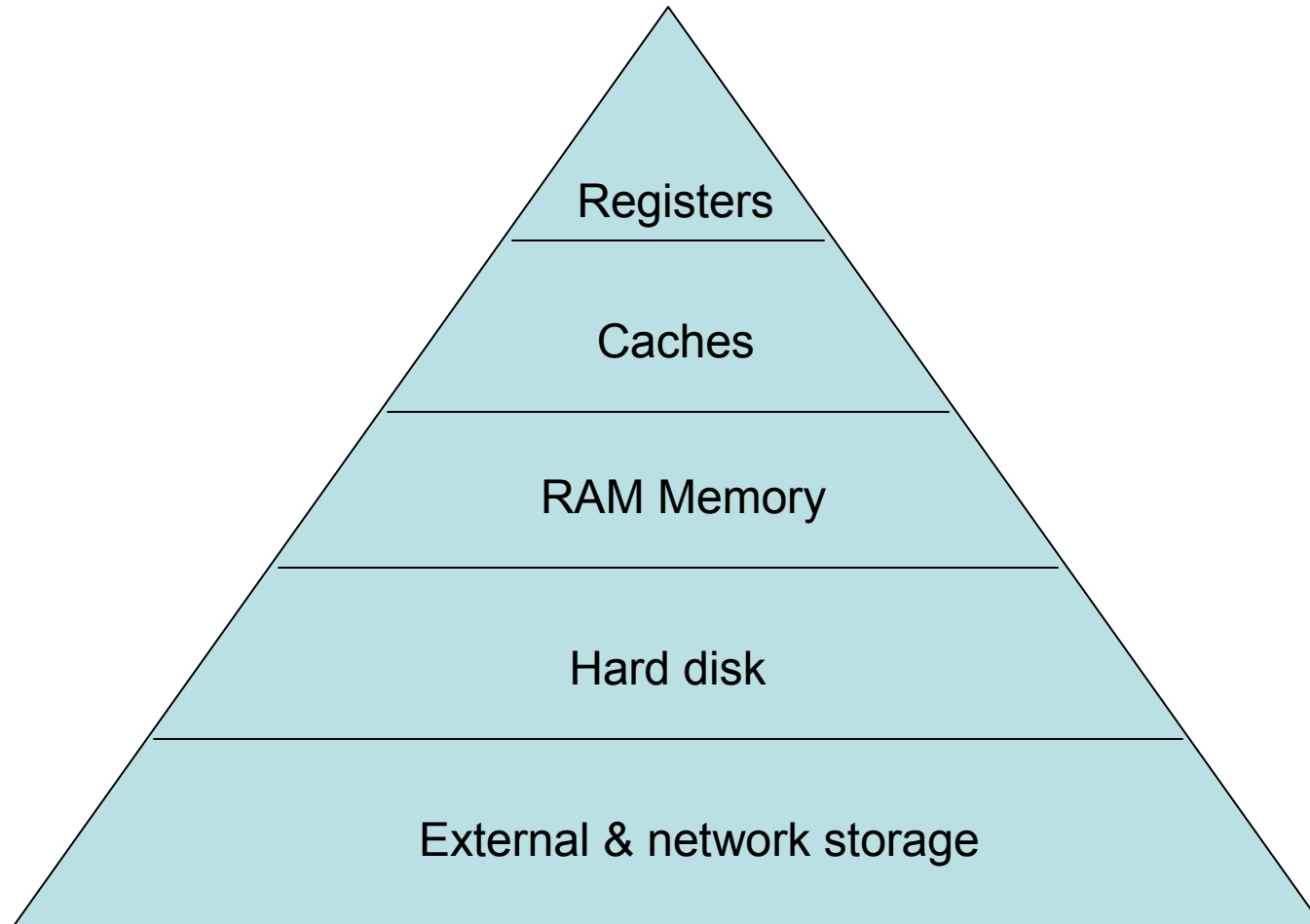
Firmware (2)

- Some examples of bootloaders
 - Grub
 - Lilo
 - RedBoot
- Some examples of RTOS
 - Linux
 - eCos
 - vxWorks
- Some examples of embedded libraries
 - Busybox
 - ucLib

Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- **ARM Caches**
- Memory management and protection
- ARM Future development

Memory hierarchy



Caches

- A *cache* is high speed memory location generally used to reduce latency in memory access for data and instructions
- Whenever CPU reads data from (slower) RAM, a copy is stored in (faster) cache
- If CPU access the same data again, it can be served from cache. This is a *hit*
- A *write buffer* speeds up writing to main memory

Cache properties

- Set associativity
 - Fully associative
 - Direct mapped
- Cache size
- Unified or separate
 - Caches for data and instruction
- Write through or write back
 - Property associated with mechanism to write data back to main memory
- Read allocate or write allocate
 - Property associated dealing with cache miss
- Replacement strategy
 - Property associated with replacing cache data with newer ones

Cache issues

- Address mapping changes
 - Virtual or physical address can get remapped
- Cache coherence and invalidation
- Direct Memory Access (DMA) operation can update main memory without going through cache

ARM and caches

- Cache and write buffers are controlled with coprocessor registers
- Register 1:
 - C bit: Cache enable/disable
 - W bit: Enable/disable write buffer
 - I bit: Enable/disable instruction cache
 - RR bit: Cache replacement strategy

ARM and caches (2)

- Register 7:
 - Controls cache and write buffers
 - Different opcodes written to this register result in different behavior. Some examples:
 - Invalidate entire instruction/data cache
 - Flush prefetch buffer
 - Writeback outstanding cache data to main memory

ARM and caches (3)

- Register 9:
 - Controls cache lockdown
 - Caches can slow down worst case execution time of code and be undeterministic because
 - It needs to handle cache misses
 - Write back of data to main memory can take time
 - Cache mechanism can load more data than request by process
 - Cache lockdown helps control these parameters to remove undeterministic behavior in critical code

Caches

- Caches can provide huge performance improvements to the system
- As a developer, one should do profiling by changing various caching options when developing board support package

Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- ARM Caches
- **Memory management and protection**
- **ARM Future development**

Memory Management

- We saw earlier that during initialization, code from ROM is copied to RAM
- But what if code size is larger than RAM available? Options:
 - Do not allow program to be loaded
 - Load only part of program and swap parts of program as requested
 - Or allow program to see large virtual memory

Virtual Memory

- Program is loaded into RAM until full
- Application starts running with partial program loaded in memory
- When program wants to access virtual memory address that is not in physical memory it generated a page fault
- Page fault handler is responsible to get new page of memory into RAM

ARM MMU

- MMU handles translation from virtual to physical memory
- Presents 4 GB address space
- Supports 3 options for memory granularity:
 - 1 MB sections
 - 64 KB pages
 - 4 KB pages
- Page fault is indicate by abort
- Abort handler is responsible for fetching pages

ARM Memory Protection

- Memory protection is required to prevent one application from overwriting other application's code
- This facility can be enabled by using MMU
- Using MMU, system goes to abort mode when application accesses memory to which it does not have permissions

ARM Memory Protection (2)

- ARM MPU allows memory protection without using MMU facilities
- ARM defines up to 8 protection regions which can be configured through MPU registers
- MPU offers good memory protection option for cases
 - Where 8 protection regions are enough
 - Virtual memory is not required

MPU Registers

- CP15
 - M bit: enable/disable memory protection
 - Cache bits: control if cache buffer is enabled/disabled
 - Buffer bits: control if write buffer is enabled/disabled
 - Access control bits: control access rights of 8 regions
 - Mechanism to define 8 memory regions

Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- ARM Caches
- Memory management and protection
- **ARM Future development**

ARM Future Development

- ARMv7 based processors
 - ARM Cortex-M3
 - TI OMAP
 - Qualcomm Scorpion
- In embedded systems its not just speed, its about speed/watt
- Approximately 2 billion ARM based devices selling each year... and growing!

ARM Future Development (2)

- Qualcomm's Snapdragon based on Scorpion:
 - 1 GHz microprocessor
 - CDMA and UMTS network support
 - Upto 12 mega pixels camera support
 - Enhanced multimedia support
 - DVD quality display support
 - GPS support
 - Support for various peripherals like hard disk, monitor, USB devices, bluetooth etc.

ARM Future Development (3)

- Its ARM v/s x86 for UMPC and mobile devices market

ARM



intel Leap ahead™



Outline

- ARM Technology Overview
- ARM Tools & Products
- ARM Processor
- ARM Toolchain
- ARM Instruction set
- Thumb instruction set
- ARM exception and interrupts
- ARM Firmware
- ARM Caches
- Memory management and protection
- ARM Future development

Questions & Comments