

# Embedded Computer Systems and Applications

## Class Notes

Instructor: Ken Q. Yang  
Dept. of ECE, URI



1



## Section 0

### Course Objectives, Plans, and Lab Tools



2



### Course Objectives: What to learn?

- Embedded Computer Architecture Concepts
  - Instruction Set Architecture
  - CPU, Memory, and I/O Organizations
  - Interfacing and Communication
    - Serial and parallel ports
    - GPIO, I<sup>2</sup>C, UART, DMA, Timer
    - AD/DAC, Programming, Coding, and Storage
- New Computing Era: mobile and cloud
- Machine Intelligence: smart device, smart city
- Applying Embedded Processor to Design Systems

**An example embedded processor: ARM Processor**



3



### Course Plan: How to Learn?

- Regular lectures (2.5 hours/week)
  - Covers basic concepts and knowledge
  - Explain tools and techniques necessary
- Weekly laboratory experiments (minimum 2 hours/week)
- Reading and Network surfing to learn tools, languages, and applications (2 hours/week)
- Assessments:
  - ❖ 2 Lab Experiments, 10% of your grade
  - ❖ 1 Design project, 60% of your grade
    - Design and documentations, 10% of your grade
    - Proper working prototype, 35%
    - Project proposal Presentation (10 minutes), 5%
    - Progress report and discussions (10 minutes), 5%
    - Final project demo and presentation (10 minutes), 5%
  - ❖ 1 Exam: 30%.



4



### Pope's Inauguration

Then...

“When smartphones and tablets

light up the sky,  
load up the clouds.”

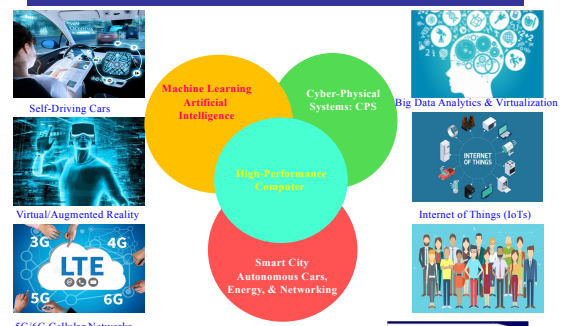
A few years later...



Source: <http://www.abcnews.com/technology/2013/02/20/pope-inauguration-when-smartphones-load-up-the-clouds/>



### A New Digital Era



## All Boil Down to One Thing

### Computer

A Very Large Fraction: Embedded Computers and Systems

- End user devices
- Variety of appliances
- Network cores
- Consumer Electronics
- IoT
- More and more



7



## Sec. 1. ARM Family Processors

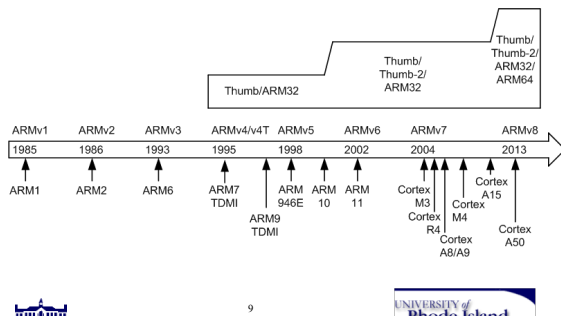
- ARM Cortex™-M Family
- Cortex™-M4 Features
- ARM AArch



8



## History



9



## The Cortex™ Processor Family

### Cortex™-A



### Cortex™-R



### Cortex™-M



10



## ARM Cortex Processor



- ARM Cortex-A family:
  - Applications processors
  - Support OS and high-performance applications
  - Such as Smartphones, Smart TV
- ARM Cortex-R family:
  - Real-time processors with high performance and high reliability
  - Support real-time processing and mission-critical control
- ARM Cortex-M family:
  - Microcontroller
  - Cost-sensitive, support SoC

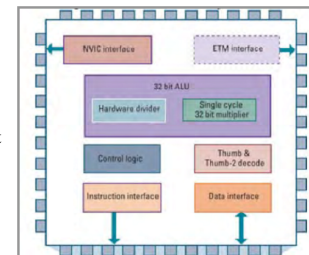


11



## What is Cortex™-M

- Harvard Architecture
- 3 stage pipeline
- Single cycle multiply
- Hardware Divide
- Thumb-2 Instruction Set
- Vectored Interrupt Controller

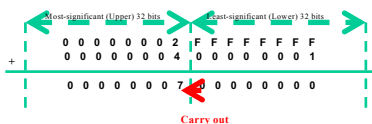


12





## Example: 64-bit Addition



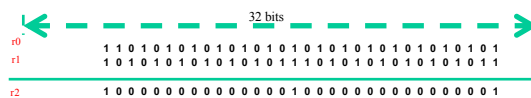
- A register can only store 32 bits
- A 64-bit integer needs two registers
- Split 64-bit addition into two 32-bit additions



19



## Example: AND r2, r0, r1



Bit-wise Logic AND



20



## Example: ORR r2, r0, r1



Bit-wise Logic OR



21



## Example: BIC r2, r0, r1

Bit Clear

$r2 = r0 \& \text{NOT } r1$

Step 1:



Step 2:



22



## Check a Bit

$$\text{bit} = a \& (1 \ll k)$$

Example:  $k = 5$

a	a7	a6	a5	a4	a3	a2	a1	a0
$1 \ll k$	0	0	1	0	0	0	0	0
$a \& (1 \ll k)$	0	0	a5	0	0	0	0	0



23



## Set a Bit

$$a \mid= (1 \ll k)$$

or

$$a = a \mid (1 \ll k)$$

Example:  $k = 5$

a	a7	a6	a5	a4	a3	a2	a1	a0
$1 \ll k$	0	0	1	0	0	0	0	0
$a \mid (1 \ll k)$	a7	a6	1	a4	a3	a2	a1	a0



24



## Clear a Bit

$$a \&= \sim(1 \ll k)$$

Example:  $k = 5$

a	a7	a6	a5	a4	a3	a2	a1	a0
$\sim(1 \ll k)$	1	1	0	1	1	1	1	1
$a \& \sim(1 \ll k)$	a7	a6	0	a4	a3	a2	a1	a0



25

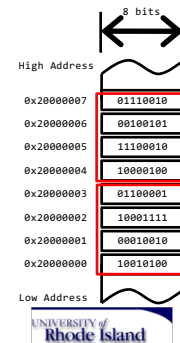


## Logic View of Memory

- When we refer to memory locations by address, we can only do so in units of bytes, halfwords or words
- Words
  - 32 bits = 4 bytes = 1 word = 2 halfwords
  - In the right diagram, we have two words at addresses:
    - 0x20000000
    - 0x20000004
  - Can you store a word anywhere? **NO.**
  - A word can only be stored at an address that's divisible by 4.
  - Memory address of a word is the lowest address of all four bytes in that word.

$$\text{Word-address mod } 4 = 0$$

26



## Quiz

What are the memory address of these four words?

32-bit Words	Bytes	Addr.
Word 3	Addr =	0015
	??	0014
	??	0013
	??	0012
Word 2	Addr =	0011
	??	0010
	??	0009
	??	0008
Word 1	Addr =	0007
	??	0006
	??	0005
	??	0004
Word 0	Addr =	0003
	??	0002
	??	0001
	??	0000



27



## Quiz (Answer)

What are the memory address of these four words?

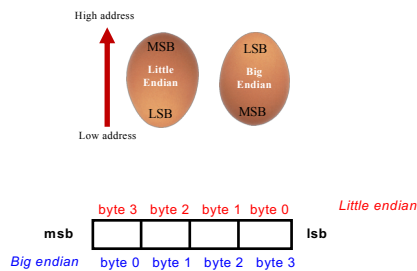
32-bit Words	Bytes	Addr.
Word 3	Addr =	0015
	0x0012	0014
	??	0013
	??	0012
Word 2	Addr =	0011
	0x0008	0010
	??	0009
	??	0008
Word 1	Addr =	0007
	0x0004	0006
	??	0005
	??	0004
Word 0	Addr =	0003
	0x0000	0002
	??	0001
	??	0000



28



## Endianess

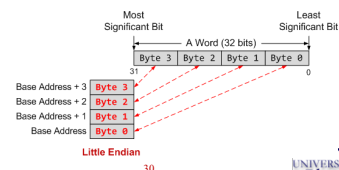


29



## Endianess

- Little Endian:** Most significant byte is stored at a high address
- Big Endian:** Most significant byte is stored at a low address
- Regardless endian, the address of a word is defined as the lowest address of all bytes it occupies.
- ARM is *Little Endian by default*. However it can be made Big Endian by configuration.

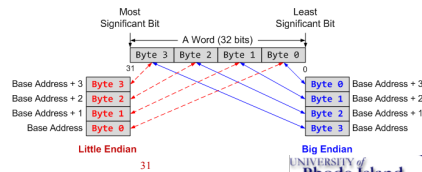


30



## Endianess

- ▶ **Little Endian:** Most significant byte is stored at a high address
- ▶ **Big Endian:** Most significant byte is stored at a low address
- ▶ Regardless endian, the address of a word is defined as the lowest address of all bytes it occupies.
- ▶ ARM is **Little Endian by default**. However it can be made Big Endian by configuration.



## Example

If big endianess is used

The word stored at address  
0x20008000 is

0xEE8C90A7

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE



32



## Example

If little endianess is used

The word stored at address  
0x20008000 is

0xA7908CEE

Endian only specifies byte order, not bit order in a byte!

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE



33



## Load-Modify-Store

C statement

x = x + 1;



```
; Assume the memory address of x is stored in r1
LDR r0, [r1]    ; load value of x from memory
ADD r0, r0, #1   ; x = x + 1
STR r0, [r1]    ; store x into memory
```



34



## Load Instructions

\* LDR rt, [rs]

- fetch data from memory into register rt.
- The memory address is specified in register rs.
- For Example:

```
; Assume r0 = 0x08200004
; Load a word:
LDR r1, [r0]    ; r1 = Memory.word[0x08200004]
```



35



## Store Instructions

\* STR rt, [rs]:

- save data in register rt into memory
- The memory address is specified in a base register rs.
- For Example:

```
; Assume r0 = 0x08200004
; Store a word
STR r1, [r0]    ; Memory.word[0x08200004] = r1
```



36



## Branch & Conditional Instructions

Instruction	Operands	Brief description	Flags
<b>B</b>	label	Branch	-
<b>BL</b>	label	Branch with Link	-
<b>BLX</b>	Rm	Branch indirect with Link	-
<b>BX</b>	Rm	Branch indirect	-

- **B label**: causes a branch to label.
- **BL label**: instruction copies the address of the next instruction into r14 (lr, the link register), and causes a branch to label.
- **BX Rm**: branch to the address held in Rm
- **BLX Rm**: copies the address of the next instruction into r14 (lr, the link register) and branch to the address held in Rm



37



## Branch With Link

- The "Branch with link (BL)" instruction implements a subroutine call by writing PC+4 into the LR of the current bank.
  - i.e. the address of the next instruction following the branch with link (allowing for the pipeline).
- To return from subroutine, simply need to restore the PC from the LR:
  - **MOV pc, lr**
  - Again, pipeline has to refill before execution continues.
- The "Branch" instruction does not affect LR.



38



## Example 1: Greatest Common Divisor

Euclid's Algorithm

```
while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
}
```

```
gcd      ; suppose r0 = a and r1 = b
        CMP r0, r1      ; a > b?
        BEQ end         ; if a = b, done

        BLT less        ; a < b
        SUB r0, r0, r1   ; a = a - b
        B gcd

less     SUB r1, r1, r0   ; b = b - a
        B gcd
```



39



## Example 2

```
int foo(int x, int y) {
    if (x + y < 0)
        return 0;
    else
        return 1;
}
```

```
foo      ADDS    r0, r0, r1
        BPL     PosOrZ
done     MOV     r0, #0
        MOV     pc, lr
PosOrZ   MOV     r0, r1
        done
```

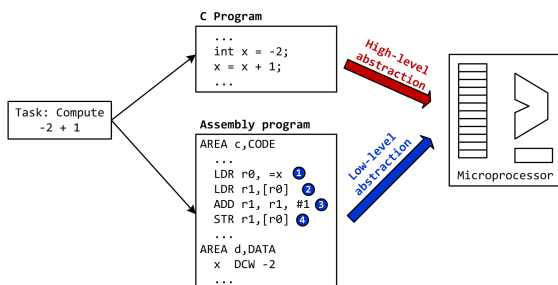
```
foo      ADDS    r0, r0, r1      ; r1 = x + y, setting CCs
        MOVPL   r0, #1         ; return 1 if n bit = 0
        MOVMI   r0, #0         ; return 0 if n bit = 1
        MOV     pc, lr         ; exit foo function
```



40



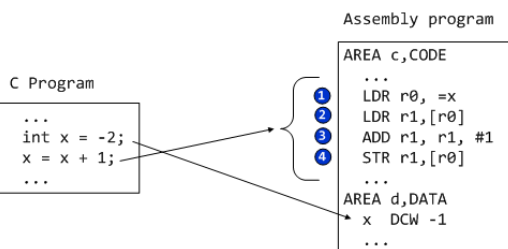
## From C to Assembly



41

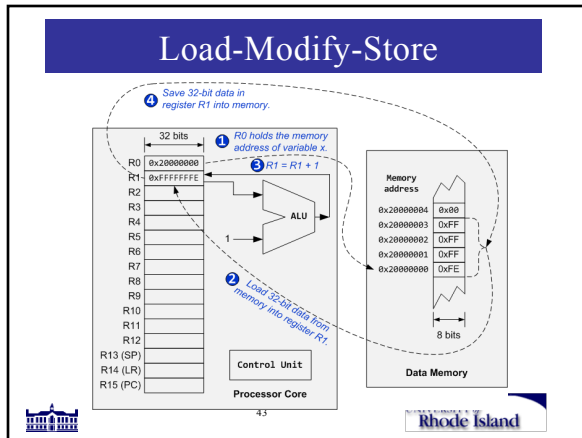


## Load-Modify-Store



42





## Assembly Instructions Supported

- Arithmetic and logic
  - Add, Subtract, Multiply, Divide, Shift, Rotate
- Data movement
  - Load, Store, Move
- Compare and branch
  - Compare, Test, If-then, Branch, compare and branch on zero
- Miscellaneous
  - Breakpoints, wait for events, interrupt enable/disable, data memory barrier, data synchronization barrier

44

UNIVERSITY of Rhode Island

## ARM Instruction Format

label mnemonic operand1, operand2, operand3 ; comments

- ▶ Label is a reference to the memory address of this instruction.
- ▶ Mnemonic represents the operation to be performed.
- ▶ The number of operands varies, depending on each specific instruction. Some instructions have no operands at all.
  - ▶ Typically, operand1 is the destination register, and operand2 and operand3 are source operands.
  - ▶ operand2 is usually a register.
  - ▶ operand3 may be a register, an immediate number, a register shifted to a constant amount of bits, or a register plus an offset (used for memory access).
- ▶ Everything after the semicolon “;” is a comment, which is an annotation explicitly declaring programmers’ intentions or assumptions.

45

UNIVERSITY of Rhode Island

## ARM Instruction Format

label mnemonic operand1, operand2, operand3 ; comments

target ADD r0, r2, r3 ; r0 = r2 + r3

label mnemonic destination operand 1<sup>st</sup> source operand 2<sup>nd</sup> source operand comment

46

UNIVERSITY of Rhode Island

## ARM Instruction Format

label mnemonic operand1, operand2, operand3 ; comments

Examples: Variants of the ADD instruction

```
ADD r1, r2, r3 ; r1 = r2 + r3
ADD r1, r3 ; r1 = r1 + r3
ADD r1, r2, #4 ; r1 = r2 + 4
ADD r1, #15 ; r1 = r1 + 15
```

47

UNIVERSITY of Rhode Island

## First Assembly

```
AREA string_copy, CODE, READONLY
EXPORT __main
ALIGN
ENTRY
__main
PROC
strcpy LDR r1, =srcStr ; Retrieve address of the source string
LDR r0, =dstStr ; Retrieve address of the destination string
loop LDRB r2, [r1], #1 ; Load a byte & increase src address pointer
STRB r2, [r0], #1 ; Store a byte & increase dst address pointer
CMP r2, #0 ; Check for the null terminator
BNE loop ; Copy the next byte if string is not ended
stop B stop ; Dead loop. Embedded program never exits.
ENDP
AREA myData, DATA, READWRITE
ALIGN
srcStr DCB "The source string.", 0 ; Strings are null terminated
dstStr DCB "The destination string.", 0 ; dstStr has more space than srcStr
END
```

48

UNIVERSITY of Rhode Island



## First Assembly

```

AREA string_copy, CODE, READONLY
EXPORT __main
ALIGN
ENTRY
PROC

strncpy
    LDR r1, =srcStr
    LDR r0, =dstStr
loop
    LDRB r2, [r1], #1
    STRB r2, [r0], #1
    CMP r2, #0
    BNE loop
    B stop
stop
    B stop

ENDP

AREA myData, DATA, READWRITE
ALIGN
srcStr
DCB "The source string.",0
dstStr
DCB "The destination string.",0
END
        
```

**Code Area**

**Data Area**

**Program Comments**

**Directives**

**Assembly Instructions**

**Labels**

49

UNIVERSITY of Rhode Island

## First Assembly

```

AREA string_copy, CODE, READONLY
EXPORT __main
ALIGN
ENTRY
PROC

strncpy
    LDR r1, =srcStr
    LDR r0, =dstStr
loop
    LDRB r2, [r1], #1
    STRB r2, [r0], #1
    CMP r2, #0
    BNE loop
    B stop
stop
    B stop

ENDP

AREA myData, DATA, READWRITE
ALIGN
srcStr
DCB "The source string.",0
dstStr
DCB "The destination string.",0
END
        
```

**Code Area**

**Data Area**

**Program Comments**

**Directives**

**Assembly Instructions**

**Labels**

50

UNIVERSITY of Rhode Island

## Structured Programming

### Divide and Conquer

*"Nothing is particularly hard if you divide it into small jobs."*

Henry Ford, Founder of Ford Motor

51

UNIVERSITY of Rhode Island

## Three basic control structures

Sequence Structure

Selection Structure

Loop Structure

52

UNIVERSITY of Rhode Island

## Top-Down Design

**Top-Down Design**

53

UNIVERSITY of Rhode Island

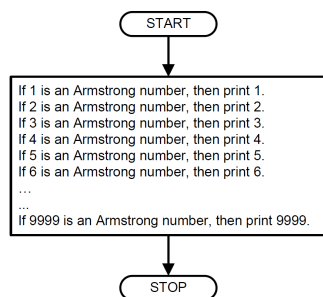
## Top-Down Design Example

- Find all Armstrong numbers less than 10,000
- Given a positive integer that has  $n$  digits, it is an Armstrong number if the sum of the  $n^{\text{th}}$  powers of its digits equals the number itself.
- For example, 371 is an Armstrong number since we have  $371 = 3^3 + 7^3 + 1^3$ .

54

UNIVERSITY of Rhode Island

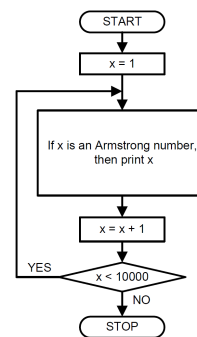
## Top-Down Design Example



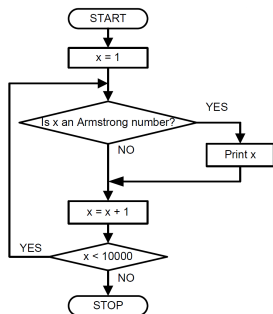
55



## Top-Down Design Example



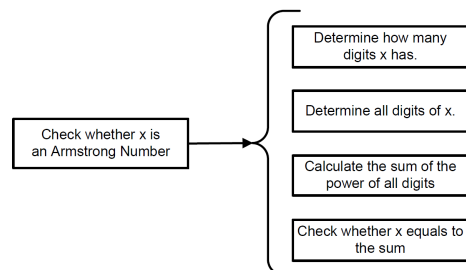
## Top-Down Design Example



57



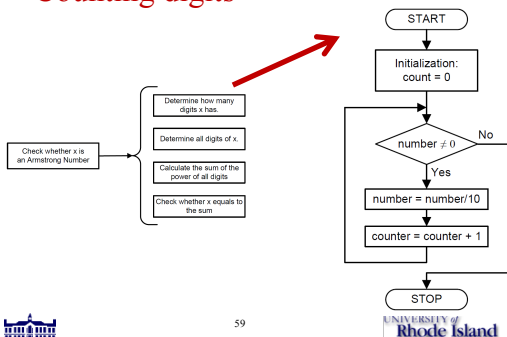
## Top-Down Design Example



58



## Top-Down Design Example: Counting digits



59



## Summary

- ❖ Introduction to ARM Processor
- ❖ Architecture of ARM
- ❖ Instruction set of ARM Cortex M4
  - A&L Instructions
  - Memory Instructions
  - Flow Control Instructions
  - Conditional Instructions
- ❖ C and Assembly Programming
  - Format and directives
- ❖ Structured Programming with examples

Read Chapter 1-7 of Textbook

60

