ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN-ĐIỆN TỬ
BỘ MÔN KỸ THUẬT ĐIỆN TỬ

Embedded Real-Time System

Chapter 3: ARM Cortex-M Microcontroller

3.1 Introduction to ARM processors
3.2 ARM Cortex-M3
3.3 ARM programming

1

cuu duong than cong . com

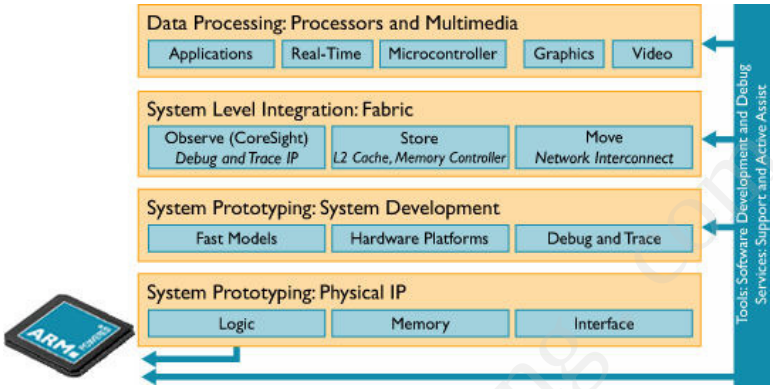
References

- Textbook
  - Joseph Yiu, "The Definitive Guide to the ARM Cortex-M3", Elsevier Newnes, 2007
- Websites
  - www.arm.com
  - www.st.com
  - www.ti.com

Bộ môn Kỹ Thuật Điện Tử - ĐHBK 2

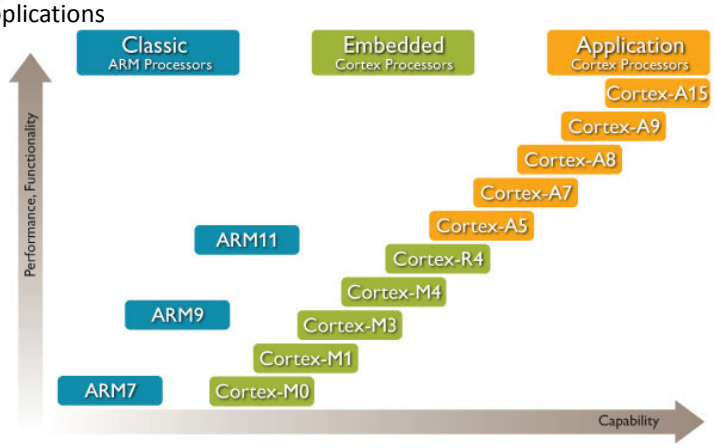
3.1 Introduction to ARM processors

- ARM (Advanced RISC Machine)
  - is the industry's leading provider of 32-bit embedded microprocessors
  - offering a wide range of processors that deliver high performance, industry leading power efficiency and reduced system cost



3.1 Introduction to ARM processors

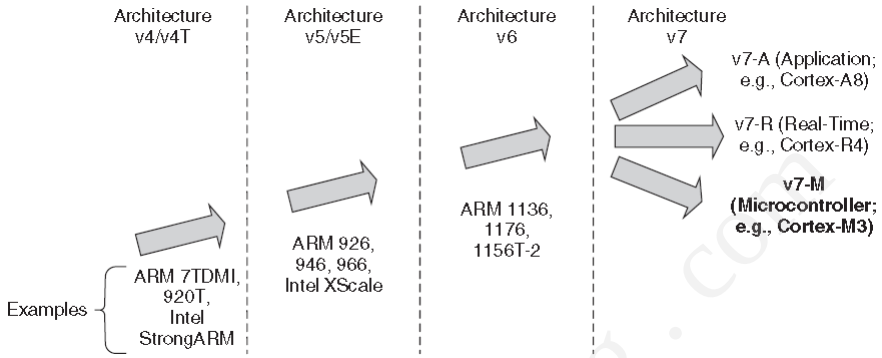
- **Cortex™-A Series -** High performance processors for open Operating Systems
- **Cortex-R Series** - Exceptional performance for real-time applications
- **Cortex-M Series** - Cost-sensitive solutions for deterministic microcontroller applications



cuu duong than cong . com

3.1 Introduction to ARM processors

- The Cortex processor families are the first products developed on architecture v7
- Cortex-M3 processor is based on one profile of the v7 architecture



3.1 Introduction to ARM processors

- Cortex-M Series:
  - Easy to use: Global standard across multiple vendors, code compatibility, unified tools and OS support
  - Low cost: smaller code, high density instruction set, small memory requirement
  - High performance: deliver more performance per MHz, enable richer features at lower power
  - Energy efficiency: run at lower MHz or with shorter activity periods
- Cortex-M Series applications
  - Microcontrollers
  - Mixed signal devices
  - Smart sensors
  - Automotive body electronics and airbags

cuu duong than cong . com

3.2 ARM Cortex-M3

- 32-bit embedded processor
- Improved code density
- Enhanced determinism, quick interrupts
- Low power consumption
- Lower-cost solutions (less than US$1)
- Wide choice of development tools




LPC1754FBD80 AT91SAM7S64-AU STM32F103RCT6 LM3S3749

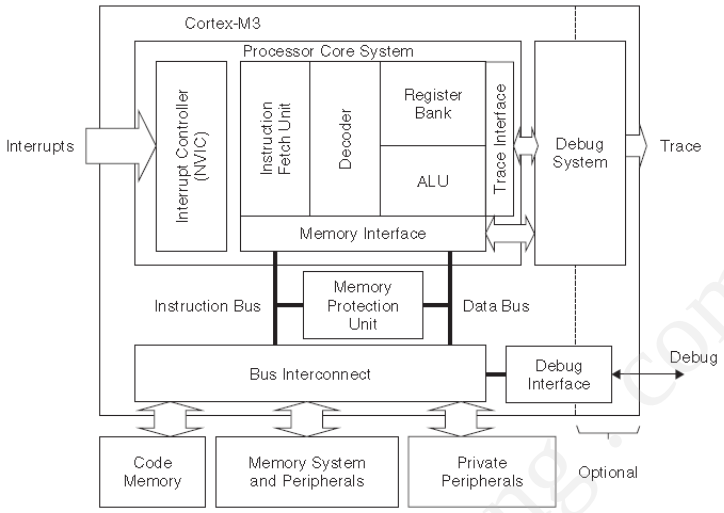
3.2 ARM Cortex-M3 - Fundamentals

- 32-bit microprocessor
  - 32-bit data path
  - 32-bit register bank
  - 32-bit memory interface
- Harvard architecture
  - separate instruction bus and data bus
  - share the same memory space
  - difference length of code and data
- Endian mode
  - both little Endian and big Endian are supported (setup at the time of core reset)
  - In most cases, Cortex-M3-based microcontrollers will be little endian.
  - Instruction fetches are always in little endian
  - PPB accesses are always in little endian.

cuu duong than cong . com

3.2 ARM Cortex-M3 - Fundamentals

Harvard architecture



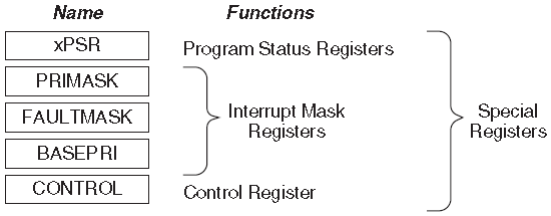
3.2 ARM Cortex-M3 - Registers

- Registers
  - R0 – R12: general purpose registers
  - R13: stack pointers
  - R14: link register (store return address)
  - R15: program counter
- Special Registers
  - Program Status Registers (PSRs)
  - Interrupt Mask Registers (PRIMASK, FAULTMASK, BASEPRI)
  - Control Register (CONTROL)

cuu duong than cong . com

3.2 ARM Cortex-M3 - Registers

- Special registers in the Cortex-M3



Register	Function
xPSR	Provide ALU flags (zero flag, carry flag), execution status, and current executing interrupt number
PRIMASK	Disable all interrupts except the nonmaskable interrupt (NMI) and HardFault
FAULTMASK	Disable all interrupts except the NMI
BASEPRI	Disable all interrupts of specific priority level or lower priority level
CONTROL	Define privileged status and stack pointer selection

cuu duong than cong . com

3.2 ARM Cortex-M3 - Registers

- **Program Status Registers**
  - Application PSR (APSR)
  - Interrupt PSR (IPSR)
  - Execution PSR (EPSR)

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT	T			ICI/IT						

Figure 3.3 Program Status Registers (PSRs) in the Cortex-M3

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number				

Figure 3.4 Combined Program Status Registers (xPSR) in the Cortex-M3

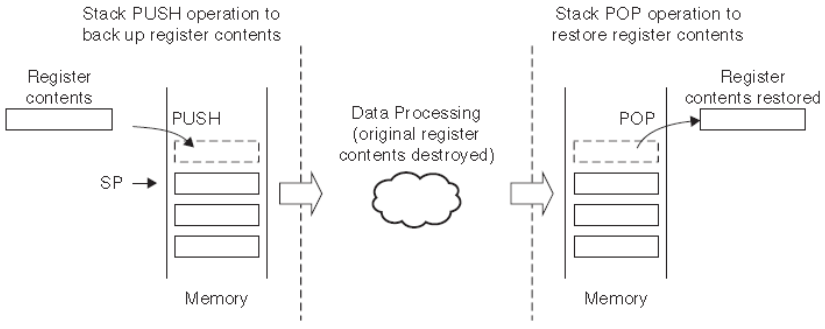
3.2 ARM Cortex-M3 - Registers

- **The Control register**
  - has two bits
  - used to define the privilege level and the stack pointer selection.

Bit	Function
CONTROL[1]	Stack status: 1 = Alternate stack is used 0 = Default stack (MSP) is used If it is in the Thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be zero when the processor is in handler mode.
CONTROL[0]	0 = Privileged in Thread mode 1 = User state in Thread mode If in handler mode (not Thread mode), the processor operates in privileged mode.

3.2 ARM Cortex-M3 - Stack Pointer

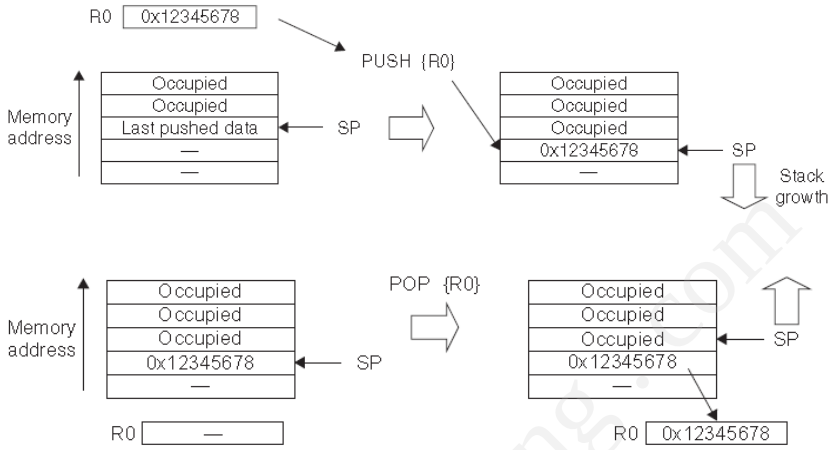
- Cortex-M3 processor has two stack pointers
  - Main Stack Pointer (MSP): used by the OS kernel, exception handlers
  - Process Stack Pointer (PSP): Used by the base-level application code
- When using the register name R13, you can only access the current stack pointer



cuu duong than cong . com

3.2 ARM Cortex-M3 – PUSH & POP

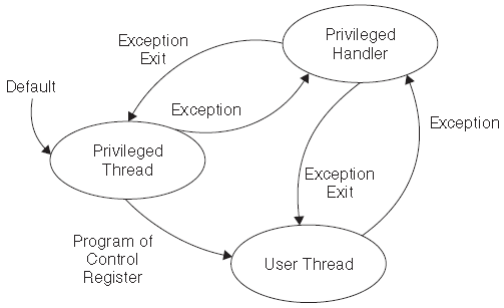
- The stack pointer (SP) points to the last data pushed to the stack memory,
- The SP decrements before a new PUSH operation



3.2 ARM Cortex-M3 – Operation Modes

- Cortex-M3 has 2 modes and two privilege levels

	Privileged	*User*
When running an exception	Handle Mode	
When running main program	Thread Mode	Thread Mode

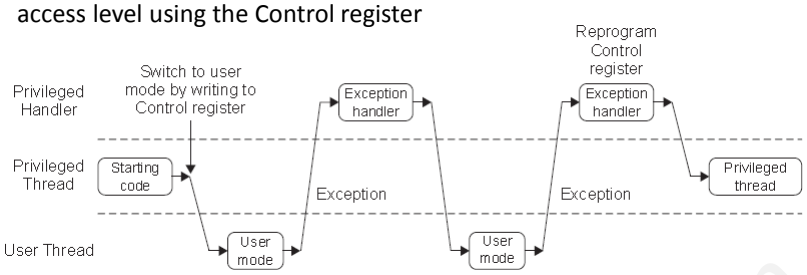


Allowed Operation Mode Transitions

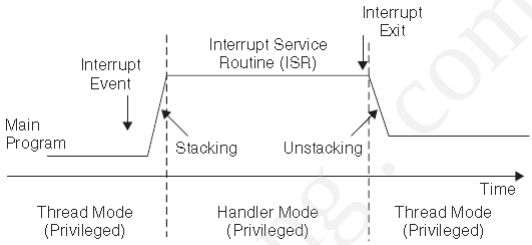
cuu duong than cong . com

3.2 ARM Cortex-M3 – Operation Modes

- Software in a privileged access level can switch the program into the user access level using the Control register



- Switching Processor Mode at Interrupt



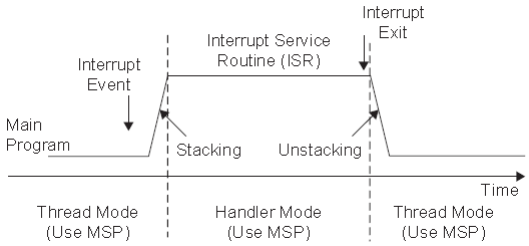
3.2 ARM Cortex-M3 - Interrupt

- The Built-In Nested Vectored Interrupt Controller (NVIC) provides a number of features:
  - **Nested interrupt support:** different priority levels
  - **Vectored interrupt support:** interrupt vector table in memory
  - **Dynamic priority changes support:** Priority levels of interrupts can be changed during run time.
  - **Reduction of interrupt latency:** automatic saving and restoring some register contents, reducing delay in switching
  - **Interrupt masking:** Interrupts and system exceptions can be masked

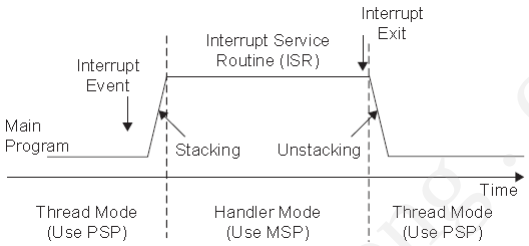
cuu duong than cong . com

3.2 ARM Cortex-M3 - Interrupt

- Control[1] = 0: Both Thread Level and Handler Use Main Stack

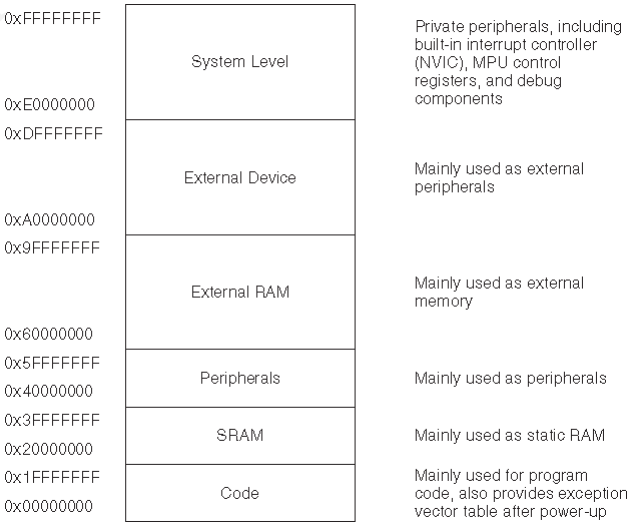


- Control[1] = 1: Thread Level Uses Process Stack and Handler Uses Main Stack



2.2 ARM Cortex-M3 – Memory Map

- The 4 GB memory space can be divided into the ranges shown in Figure:



cuu duong than cong . com

2.2 ARM Cortex-M3 – Bus Interface & MPU

- Bus Interface
  - Code memory bus for code memory, consist of two buses: I-Code and D-code
  - System bus: for memory and peripherals
  - Private peripheral bus: for private peripherals such as debugging components
- Memory Protection Unit (MPU)
  - allow access rules for privilege access and user program access
  - When an access rule is violated, a fault exception is generated
  - MPU is setup by an OS
  - allowing data used by privileged code to be protected from untrusted user programs

2.2 ARM Cortex-M3 – Instruction set

- Instruction Set:
  - support thumb-2 instruction set.
  - allow 32-bit and 16-bit instructions
- Traditional ARM processor has two operation states:
  - 32-bit ARM state
  - 16-bit Thumb state
- To get the best of both worlds, many applications have mixed ARM and Thumb codes
- Advantages over traditional ARM processor of ARM Cortex M3
  - No state switching overhead, saving both execution time and instruction space
  - No need to separate ARM code and Thumb code source files
  - easier to write software, because there is no need to worry about switching code between ARM and Thumb

cuu duong than cong . com

3.2 ARM Cortex-M3 - Exceptions

- The interrupt features in the Cortex-M3 are implemented in the NVIC

Exception Number	Exception Type	Priority (Default to 0 if Programmable)	Description
0	NA	NA	No exception running
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt (external NMI input)
3	Hard fault	−1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; MPU violation or access to illegal locations
5	Bus fault	Programmable	Bus error (Prefetch Abort or Data Abort)
6	Usage fault	Programmable	Exceptions due to program error
7–10	Reserved	NA	Reserved

cuu duong than cong . com

3.2 ARM Cortex-M3 - Exceptions

11	SVCall	Programmable	System service call
12	Debug monitor	Programmable	Debug monitor (break points, watchpoints, or external debug request)
13	Reserved	NA	Reserved
14	PendSV	Programmable	Pendable request for system device
15	SYSTICK	Programmable	System tick timer
16	IRQ #0	Programmable	External interrupt #0
17	IRQ #1	Programmable	External interrupt #1
...	...	...	...
255	IRQ #239	Programmable	External interrupt #239

- The number of external interrupt inputs is defined by chip manufacturers.
- A maximum of 240 external interrupt inputs can be supported

Questions

1. What are differences among Cortex- A, -R, and -M series?
2. What is Harvard architecture?
3. How many general purpose registers in Cortex-M3 are there?
4. How many special registers in Cortex-M3 are there?
5. What are difference between little endian and big endian?
6. How many allowed operation modes in Cortex-M3 are there?
7. What is functions of control register in Cortex-M3?
8. What are features of NVIC in Cortex-M3?
9. What is the maximum memory space of Cortex-M3?
10. What is the size of memory space for external RAM?
11. Can program code can be executed from an external RAM region?
12. Does Cortex-M3 support ARM code?
13. Why is Thumb-2 instruction set more advanced than previous?

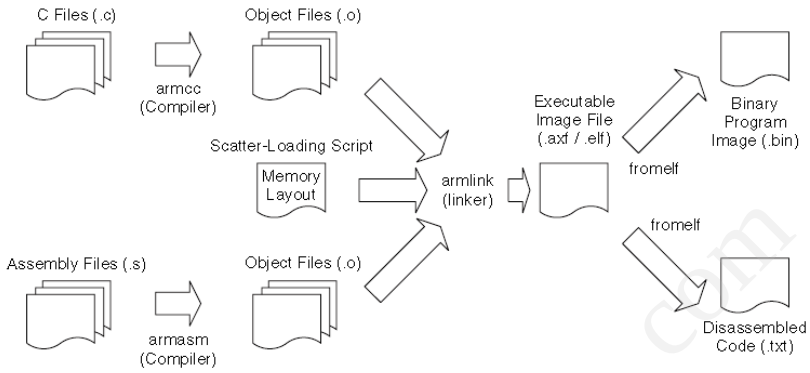
3.3 ARM Programming

- Using Assembly
  - for small projects
  - can get the best optimization, smallest memory size
  - increase development time, easy to make mistakes
- Using C
  - easier for implementing complex operations
  - larger memory size
  - able to include assembly code (*inline assembler*)
  - Tools: RealView Development Suite (RVDS), KEIL RealView Microcontroller Development Kit

cuu duong than cong . com

3.3 ARM Programming

- Typical development flow



3.3 ARM Programming – Simple program

This simple program contains the initial SP value, the initial PC value, and setup registers and then does the required calculation in a loop.

```
STACK_TOP       EQU 0x20002000 ; constant for SP starting value
                AREA |Header Code|, CODE
                DCD STACK_TOP     ; Stack top
                DCD Start         ; Reset vector
                ENTRY             ; Indicate program execution start here
Start                             ; Start of main program initialize registers
                MOV r0, #10       ; Starting loop counter value
                MOV r1, #0        ; starting result
loop                              ; Calculated 10+9+8+...+1
                ADD r1, r0 ; R1R1 + R0
                SUBS r0, #1       ; Decrement R0, update fl ag (“S” suffi x)
                BNE loop ; If result not zero jump to loop,
                                  ; result is now in R1
deadloop        B deadloop        ; Infi nite loop
                END               ; End of file
```


cuu duong than cong . com

3.3 ARM Programming - Simple program

- Compile a assemply code
  - armasm --cpu cortex-m3 -o test1.o test1.s
- Link to executable image
  - armlink --rw_base 0x20000000 --ro_base 0x0 --map -o test1.elf test1.o
- Create the binary image
  - fromelf --bin --output test1.bin test1.elf
- generate a disassembled code list file
  - fromelf -c --output test1.list test1.elf

cuu duong than cong . com

3.3 ARM Programming

- Using EQU to define constants

```
NVIC_IRQ_SETEN0  EQU 0xE000E100
NVIC_IRQ0_ENABLE          EQU 0x1
        LDR R0,NVIC_IRQ_SETEN0
        MOV R1,#NVIC_IRQ0_ENABLE ; Move immediate data to register
        STR R1, [R0]          ; Enable IRQ 0 by writing R1 to address in R0
```

- Using DCI to code an instruction

```
DCI 0xBE00 ; Breakpoint (BKPT 0), a 16-bit instruction
```

- Using DCB and DCD to define binary data

```
MY_NUMBER
        DCD 0x12345678
HELLO_TXT
        DCB “Hello\n”,0 ; null terminated string
```

3.3 ARM Programming – Moving data

- Data transfers can be of one of the following types:
  - Moving data between register and register
  - Moving data between memory and register
  - Moving data between special register and register
  - Moving an immediate data value into a register

```
        MOV     R8, R3          ; moving data from register R3 to register R8
        MOV     R0, #0x12       ; Set R0 = 0x12 (hexadecimal)
        MOV     R1, #'A'        ; Set R1 = ASCII character A
```

```
        MRS     R0, PSR         ; Read Processor status word into R0
        MSR     CONTROL, R1     ; Write value of R1 into control register
```

```
        LDR R0, address1        ; R0 set to 0x4001
        ...
address1
        0x4000: MOV R0, R1      ; address1 contains program code
```


3.3 ARM Programming – Using Stack

- Stack PUSH and POP

```
subroutine_1
        PUSH {R0-R7, R12, R14}      ; Save registers
        … ; Do your processing
        POP {R0-R7, R12, R14}       ; Restore registers
        BX R14                      ; Return to calling function
```

- Link register (LR or R14)

```
main                        ; Main program
        BL function1        ; Call function1 using Branch with Link
                            ; instruction.
                            ; PC  function1 and
                            ; LR  the next instruction in main
        …
function1
        …                   ; Program code for function 1
        BX LR ; Return
```


cuu duong than cong . com

3.3 ARM Programming – Special Register

- Special registers can only be accessed via MSR and MRS instructions

```
MRS     <reg>, <special_reg>        ; Read special register
MSR     <special_reg>, <reg>        ; write to special register
```

- ASP can be changed by using MSR instruction, but EPSR and IPSR are read-only

```
MRS     r0, APSR            ; Read Flag state into R0
MRS     r0, IPSR            ; Read Exception/Interrupt state
MRS     r0, EPSR            ; Read Execution state
MSR     APSR, r0            ; Write Flag state
MRS     r0, PSR             ; Read the combined program status word
MSR     PSR, r0             ; Write combined program state word
```


cuu duong than cong . com

3.3 ARM Programming – Special Register

- To access the Control register, the MRS and MSR instructions are used:

```
MRS     r0, CONTROL         ; Read CONTROL register into R0
MSR     CONTROL, r0         ; Write R0 into CONTROL register
```

Bit	Function
CONTROL[1]	Stack status: 1 = Alternate stack is used 0 = Default stack (MSP) is used If it is in the Thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be zero when the processor is in handler mode.
CONTROL[0]	0 = Privileged in Thread mode 1 = User state in Thread mode If in handler mode (not Thread mode), the processor operates in privileged mode.

3.3 ARM Programming

- 16-Bit Load and Store Instructions

Instruction	Function
LDR	Load word from memory to register
LDRH	Load half word from memory to register
LDRB	Load byte from memory to register
LDRSH	Load half word from memory, sign extend it, and put it in register
LDRSB	Load byte from memory, sign extend it, and put it in register
STR	Store word from register to memory
STRH	Store half word from register to memory
STRB	Store byte from register to memory
LDMIA	Load multiple increment after
STMIA	Store multiple increment after
PUSH	Push multiple registers
POP	Pop multiple registers

cuu duong than cong . com

3.3 ARM Programming

- 16-Bit Branch Instructions

Instruction	Function
B	Branch
B<cond>	Conditional branch
BL	Branch with link; call a subroutine and store the return address in LR
BLX	Branch with link and change state (BLX <reg> only)[1]
CBZ	Compare and branch if zero (architecture v7)
CBNZ	Compare and branch if nonzero (architecture v7)
IT	IF-THEN (architecture v7)

3.3 ARM Programming – Arithmetic Instructions

Instruction	Operation

`ADD Rd, Rn, Rm ; Rd = Rn + Rm`
`ADD Rd, Rm ; Rd = Rd + Rm`
`ADD Rd, #immed ; Rd = Rd + #immed`
	ADD operation

`ADC Rd, Rn, Rm ; Rd = Rn + Rm + carry`
`ADC Rd, Rm ; Rd = Rd + Rm + carry`
`ADC Rd, #immed ; Rd = Rd + #immed + ; carry`
	ADD with carry

`ADDW Rd, Rn,#immed ; Rd = Rn + #immed`	ADD register with 12-bit immediate value

`SUB Rd, Rn, Rm ; Rd = Rn - Rm`
`SUB Rd, #immed ; Rd = Rd - #immed`
`SUB Rd, Rn,#immed ; Rd = Rn -#immed`
	SUBTRACT

`SBC Rd, Rm ; Rd = Rd - Rm - ; carry flag`
`SBC.W Rd, Rn, #immed ; Rd = Rn - #immed - ; carry flag`
`SBC.W Rd, Rn, Rm ; Rd = Rn - Rm - ; carry flag`
	SUBTRACT with borrow (carry)

`RSB.W Rd, Rn, #immed ; Rd = #immed -Rn`
`RSB.W Rd, Rn, Rm ; Rd = Rm - Rn`
`MUL Rd, Rm ; Rd = Rd * Rm`
`MUL.W Rd, Rn, Rm ; Rd = Rn * Rm`
	Reverse subtract
	
	Multiply

3.3 ARM Programming – IF-THEN

- The IF-THEN (IT) instructions allow up to four succeeding instructions (called an *IT block) to* be conditionally executed.
- They are in the following formats:

IT<x> <cond>
IT<x><y> <cond>
IT<x><y><z> <cond>
where:
- *<x > specifi es the execution condition for the second instruction*
- *<y > specifi es the execution condition for the third instruction*
- *<z > specifi es the execution condition for the fourth instruction*

cuu duong than cong . com

3.3 ARM Programming – IF-THEN

Symbol	Condition	Flag
EQ	Equal	Z set
NE	Not equal	Z clear
CS/HS	Carry set/unsigned higher or same	C set
CC/LO	Carry clear/unsigned lower	C clear
MI	Minus/negative	N set
PL	Plus/positive or zero	N clear
VS	Overflow	V set
VC	No overflow	V clear
HI	Unsigned higher	C set and Z clear
LS	Unsigned lower or same	C clear or Z set
GE	Signed greater than or equal	N set or V set, or N clear and V clear (N == V)
LT	Signed less than	N set and V clear, or N clear and V set (N != V)
GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0, N == V)
LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)
AL	Always (unconditional)	–

cuu duong than cong . com

3.3 ARM Programming – IF-THEN

- An example of a simple conditional execution

```
if (R1<R2) then
        R2=R2-R1
        R2=R2/2
else
        R1=R1-R2
        R1=R1/2
```

- In assembly:

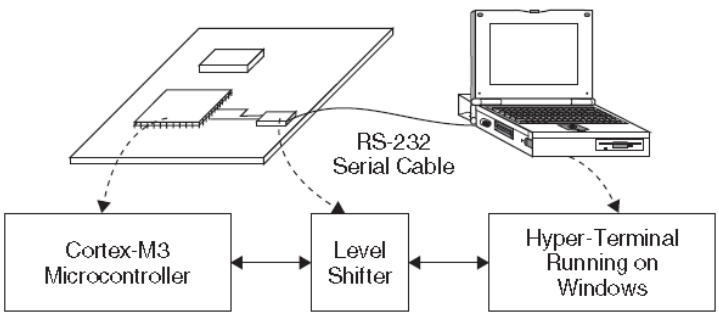
```
CMP     R1, R2     ; If R1 < R2 (less then)
ITTEE   LT         ; then execute instruction 1 and 2
                   ; (indicated by T)
                   ; else execute instruction 3 and 4
                   ; (indicated by E)
SUBLT.W R2,R1      ; 1st instruction
LSRLT.W R2,#1      ; 2nd instruction
SUBGE.W R1,R2      ; 3rd instruction (notice the GE is opposite of LT)
LSRGE.W R1,#1      ; 4th instruction
```

3.3 ARM Programming – Using Data Memory

```
STACK_TOP       EQU 0x20002000    ; constant for SP starting value
                AREA | Header Code|, CODE
                DCD STACK_TOP     ; SP initial value
                DCD Start         ; Reset vector
                ENTRY
Start           ; Start of main program, initialize registers
                MOV r0, #10       ; Starting loop counter value
                MOV r1, #0        ; starting result. Calculated 10+9+8+...+1
loop            ADD r1, r0        ; R1 = R1 + R0
                SUBS r0, #1       ; Decrement R0, update fl ag (“S” suffi x)
                BNE loop          ; If result not zero jump to loop; Result is now in R1
                LDR r0,=MyData1   ; Put address of MyData1 into R0
                STR r1,[r0]       ; Store the result in MyData1
deadloop        B deadloop        ; Infi nite loop
                AREA | Header Data|, DATA
                ALIGN 4
MyData1         DCD 0             ; Destination of calculation result
MyData2         DCD 0
                END               ; End of file
```


3.3 ARM Programming

- A Low-Cost Test Environment for Outputting Text Messages
  - UART interface is common output method to send messages to a console
  - Hyper-Terminal program can be used as a console



cuu duong than cong . com

2.3 ARM Programming

- A simple routine to output a character through UART

```
UART0_BASE      EQU 0x4000C000
UART0_FLAG      EQU UART0_BASE+0x018
UART0_DATA      UART0_BASE+0x000
Putc                        ; Subroutine to send a character via UART
                            ; Input R0 = character to send
        PUSH {R1,R2, LR}    ; Save registers
        LDR R1,=UART0_FLAG
PutcWaitLoop
        LDR R2,[R1]          ; Get status fl ag
        TST R2, #0x20        ; Check transmit buffer full flag bit
        BNE PutcWaitLoop ; If busy then loop
        LDR R1,=UART0_DATA ; otherwise
        STRB R0, [R1]        ; Output data to transmit buffer
        POP {R1,R2, PC}      ; Return
```

The register addresses and bit definitions here are just examples

ARM Development Kit

- Mini2440 | S3C2440 ARM9 Board
  - CPU: 400 MHz Samsung S3C2440A ARM920T
  - RAM: 64 MB SDRAM, 32 bit Bus
  - Flash: 256 MB NAND Flash and 2 MB NOR Flash with BIOS
  - EEPROM: 256 Byte (I2C)

- LM3S9B96 development Kit
  - Stellaris LM3S9B96 MCU with fully-integrated Ethernet, CAN, and USB OTG/Host/Device
  - Bright 3.5" QVGA LCD touch-screen display
  - Navigation POT switch and select pushbuttons
  - Integrated Interchip Sound (I2S) Audio Interface

IC Design Lab, 116B1

cuu duong than cong . com

Assignments

1. Write a program to move 10 words from 0x20000000 to 0x3000000.
2. Write a program to read STATUS register and write to 0x20000004
3. Write a program to write a value in 0x30000000 to CONTROL register
4. Write a subroutine to perform a function 40*X + 50
5. Write a subroutine to convert data of 10 words form big endian to little endian.
6. Write a program as pseudo code below:

```
if (R0 equal R1) then {
        R3 = R4 + R5
        R3 = R3 / 2  }
else {
        R3 = R6 + R7
        R3 = R3 / 2
}
```


cuu duong than cong . com