Embedded System Design

Chapter 2: Microcontroller Series (Part 2)

3. ARM Cortex-M4
4. ARM Cortex-M3 & M4 Microcontroller Series
5. ARM programming
3. ARM Cortex-M4

- **Features**
  - tight integration of system peripherals reduces area and development costs
  - Thumb instruction set combines high code density with 32-bit performance
  - optional IEEE754-compliant single-precision FPU
  - code-patch ability for ROM system updates
  - power control optimization of system components
  - integrated sleep modes for low power consumption
  - **DSP extension**: Single cycle 16/32-bit MAC, single cycle dual 16-bit MAC, 8/16-bit SIMD arithmetic
3. ARM Cortex-M4 - Architecture
3. ARM Cortex-M4 – Peripherals

• **Nested Vectored Interrupt Controller**
  – An interrupt controller that supports low latency interrupt processing.

• **System Control Block**
  – Provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

• **System timer**
  – The system timer, SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter.

• **Memory Protection Unit**
  – *defining the* memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region.

• **Floating-point Unit**
  – The *Floating-Point Unit (FPU)* provides IEEE754-compliant operations on single-precision, 32-bit, floating-point values.
3. ARM Cortex-M4 – Instruction Set

- Extra instructions for floating point operations

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VABS.F32</td>
<td>Sd, Sm</td>
<td>Floating-point Absolute</td>
</tr>
<tr>
<td>VADD.F32</td>
<td>{Sd,} Sn, Sm</td>
<td>Floating-point Add</td>
</tr>
<tr>
<td>VCMP.F32</td>
<td>Sd, &lt;Sm</td>
<td>#0.0&gt;</td>
</tr>
<tr>
<td>VCMPE.F32</td>
<td>Sd, &lt;Sm</td>
<td>#0.0&gt;</td>
</tr>
<tr>
<td>VCVT.S32.F32</td>
<td>Sd, Sm</td>
<td>Convert between floating-point and integer</td>
</tr>
<tr>
<td>VCVT.S16.F32</td>
<td>Sd, Sd, #fbits</td>
<td>Convert between floating-point and fixed point</td>
</tr>
<tr>
<td>VCVTR.S32.F32</td>
<td>Sd, Sm</td>
<td>Convert between floating-point and integer with rounding</td>
</tr>
</tbody>
</table>
3. ARM Cortex-M4 – Instruction Set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCVT&lt;B</td>
<td>H&gt;.F32.F16</td>
</tr>
<tr>
<td>VCVTT&lt;B</td>
<td>T&gt;.F32.F16</td>
</tr>
<tr>
<td>VDIV.F32</td>
<td>Floating-point Divide</td>
</tr>
<tr>
<td>VFMA.F32</td>
<td>Floating-point Fused Multiply Accumulate</td>
</tr>
<tr>
<td>VFNMA.F32</td>
<td>Floating-point Fused Negate Multiply Accumulate</td>
</tr>
<tr>
<td>VFMS.F32</td>
<td>Floating-point Fused Multiply Subtract</td>
</tr>
<tr>
<td>VFNMS.F32</td>
<td>Floating-point Fused Negate Multiply Subtract</td>
</tr>
<tr>
<td>VLDM.F&lt;32</td>
<td>64&gt;</td>
</tr>
<tr>
<td>VLDR.F&lt;32</td>
<td>64&gt;</td>
</tr>
<tr>
<td>VLMA.F32</td>
<td>Floating-point Multiply Accumulate</td>
</tr>
<tr>
<td>VLMS.F32</td>
<td>Floating-point Multiply Subtract</td>
</tr>
<tr>
<td>VMOV.F32</td>
<td>Floating-point Move immediate</td>
</tr>
</tbody>
</table>
### 3. ARM Cortex-M4 – Instruction Set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMOV</td>
<td>Sd, Sm</td>
<td>Floating-point Move register</td>
</tr>
<tr>
<td>VMOV</td>
<td>Sn, Rt</td>
<td>Copy ARM core register to single precision</td>
</tr>
<tr>
<td>VMOV</td>
<td>Sm, Sm1, Rt, Rt2</td>
<td>Copy 2 ARM core registers to 2 single precision</td>
</tr>
<tr>
<td>VMOV</td>
<td>Dd[x], Rt</td>
<td>Copy ARM core register to scalar</td>
</tr>
<tr>
<td>VMOV</td>
<td>Rt, Dn[x]</td>
<td>Copy scalar to ARM core register</td>
</tr>
<tr>
<td>VMRS</td>
<td>Rt, FPSCR</td>
<td>Move FPSCR to ARM core register or APSR</td>
</tr>
<tr>
<td>VMSR</td>
<td>FPSCR, Rt</td>
<td>Move to FPSCR from ARM Core register</td>
</tr>
<tr>
<td>VMUL.F32</td>
<td>{Sd}, Sn, Sm</td>
<td>Floating-point Multiply</td>
</tr>
<tr>
<td>VNEG.F32</td>
<td>Sd, Sm</td>
<td>Floating-point Negate</td>
</tr>
<tr>
<td>VNMLA.F32</td>
<td>Sd, Sn, Sm</td>
<td>Floating-point Multiply and Add</td>
</tr>
<tr>
<td>VNMLS.F32</td>
<td>Sd, Sn, Sm</td>
<td>Floating-point Multiply and Subtract</td>
</tr>
<tr>
<td>VNMUL</td>
<td>{Sd}, Sn, Sm</td>
<td>Floating-point Multiply</td>
</tr>
</tbody>
</table>
### 3. ARM Cortex-M4 – Instruction Set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPOP</td>
<td>list</td>
</tr>
<tr>
<td>VPOP</td>
<td>Pop extension registers</td>
</tr>
<tr>
<td>VPUSH</td>
<td>list</td>
</tr>
<tr>
<td>VPUSH</td>
<td>Push extension registers</td>
</tr>
<tr>
<td>VSQRT.F32</td>
<td>Sd, Sm</td>
</tr>
<tr>
<td>VSQRT.F32</td>
<td>Calculates floating-point Square Root</td>
</tr>
<tr>
<td>VSTM</td>
<td>Rn{!}, list</td>
</tr>
<tr>
<td>VSTM</td>
<td>Floating-point register Store Multiple</td>
</tr>
<tr>
<td>VSTR.F&lt;32</td>
<td>64&gt;</td>
</tr>
<tr>
<td>VSTR.F&lt;32</td>
<td>64&gt;</td>
</tr>
<tr>
<td>VSUB.F&lt;32</td>
<td>64&gt;</td>
</tr>
<tr>
<td>VSUB.F&lt;32</td>
<td>64&gt;</td>
</tr>
</tbody>
</table>
3. ARM Cortex-M4 – Instruction Set

- Extra instructions for DSP

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UADD16</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Add 16</td>
</tr>
<tr>
<td>UADD8</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Add 8</td>
</tr>
<tr>
<td>USAX</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Subtract and Add with Exchange</td>
</tr>
<tr>
<td>UHADD16</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Halving Add 16</td>
</tr>
<tr>
<td>UHADD8</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Halving Add 8</td>
</tr>
<tr>
<td>UHASX</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Halving Add and Subtract with Exchange</td>
</tr>
<tr>
<td>UHSAX</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Halving Subtract and Add with Exchange</td>
</tr>
<tr>
<td>UHSUB16</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Halving Subtract 16</td>
</tr>
<tr>
<td>UHSUB8</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Halving Subtract 8</td>
</tr>
<tr>
<td>UBFX</td>
<td>Rd, Rn, #lsb, #width</td>
<td>Unsigned Bit Field Extract</td>
</tr>
<tr>
<td>UDIV</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Divide</td>
</tr>
</tbody>
</table>
### 3. ARM Cortex-M4 – Instruction Set

- **Extra instructions for DSP**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMAAL</td>
<td>RdLo, RdHi, Rn, Rm</td>
</tr>
<tr>
<td>UMLAL</td>
<td>RdLo, RdHi, Rn, Rm</td>
</tr>
<tr>
<td>UMULL</td>
<td>RdLo, RdHi, Rn, Rm</td>
</tr>
<tr>
<td>UQADD16</td>
<td>{Rd,} Rn, Rm</td>
</tr>
<tr>
<td>UQADD8</td>
<td>{Rd,} Rn, Rm</td>
</tr>
</tbody>
</table>
3. ARM Cortex-M4 – Instruction Set

- Extra instructions for DSP

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UQASX</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Saturating Add and Subtract with Exchange</td>
</tr>
<tr>
<td>UQSAX</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Saturating Subtract and Add with Exchange</td>
</tr>
<tr>
<td>UQSUB16</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Saturating Subtract 16</td>
</tr>
<tr>
<td>UQSUB8</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Saturating Subtract 8</td>
</tr>
<tr>
<td>USAD8</td>
<td>{Rd,} Rn, Rm</td>
<td>Unsigned Sum of Absolute Differences</td>
</tr>
<tr>
<td>USADA8</td>
<td>{Rd,} Rn, Rm, Ra</td>
<td>Unsigned Sum of Absolute Differences and Accumulate</td>
</tr>
<tr>
<td>USAT</td>
<td>Rd, #n, Rm {,shift #s}</td>
<td>Unsigned Saturate</td>
</tr>
<tr>
<td>USAT16</td>
<td>Rd, #n, Rm</td>
<td>Unsigned Saturate 16</td>
</tr>
</tbody>
</table>
4. ARM Cortex Microcontroller Series

ST Electronics
4. ARM Cortex Microcontroller Series

Texas Instruments

Keystone Processors
Up to 5.6GHz Cortex-A15

Sitara™ Processors
Up to 1.35GHz Cortex-A8

Hercules™ Safety MCUs
Up to 220MHz Cortex-R4

C2000™ C28x + ARM
Up to 150MHz Cortex-M3

Tiva™ C Series MCUs
Up to 80MHz Cortex-M
4. ARM Cortex Microcontroller Series

- Stellaris® Roadmap

**ARM Cortex-M3**
- LM3S9000
- LM3S8000
- LM3S6000

**ARM Cortex-M4F**
- LM4F23x
- LM4F13x
- LM4F21x
- LM4F12x
- LM4F11x

**TMS / RTP 2H13**
- Ethernet + USB + CAN
  - 120 MHz
  - 1MB Flash, 256KB SRAM
  - 10/100 ENET MAC + PHY
  - USB H/D/OTG w/FS PHY & HS ULPI
  - Up to 2 x CAN
  - Parallel Bus Interface (EPI)
  - Crypto

**RTP Feb ‘13 (TMX Now)**
- USB H/D/OTG + CAN
- 80 MHz
- 256K Flash / 32K SRAM
- Low-power hibernate
- 2 x 1 Msps 12-bit ADCs
- Motion control options

**TMS / RTP 2H13**
- USB + CAN
  - 120 MHz
  - 1MB Flash, 256KB SRAM
  - USB H/D/OTG w/FS PHY & HS ULPI
  - Up to 2 x CAN
  - Parallel Bus Interface (EPI)
  - Crypto
4.1 ARM Cortex-M3 – LM3S9B96

• Stellaris LM3S9B96 microcontroller (Texas Instruments)
  – ARM Cortex-M3
  – 80MHz, 100 DMIPS
  – 256 KB Flash
  – 96KB SRAM
  – UART, SSI, I2C, I2S, CAN, Ethernet, USB
  – Timer, DMA, GPIO
  – PWM
  – ADC
  – JTAG, ARM Serial Wire Debug
4.1 ARM Cortex-M3 – LM3S9B96

- Block diagram:
  - 2 on-chip buses: AHB, APB
4.1 ARM Cortex-M3 – LM3S9B96

- Target applications
  - Gaming equipment
  - Network appliances
  - Home and commercial site monitoring
  - Motion control
  - Medical instruments
  - Remote monitoring
  - Test equipment
  - Fire and security
  - Lighting control
  - Transportation
4.2 ARM Cortex-M4 – TM4F

• Features:
  – ARM Cortex-M4F core CPU speed up to 80 MHz with floating point
  – Up to 256-KB Flash
  – Up to 32-KB single-cycle SRAM
  – Two high-speed 12-bit ADCs up to 1MSPS
  – Up to two CAN 2.0 A/B controllers
  – Optional full-speed USB 2.0 OTG/Host/Device
  – Up to 40 PWM outputs
  – Serial communication with up to: 8 UARTs, 6 I2Cs, 4 SPI/SSI
  – Intelligent low-power design power consumption as low as 1.6 μA
4.2 ARM Cortex-M4 – Development Kit

- **EK-TM4F120 LaunchPad Evaluation Kit**
  - 80-MHz, 32-bit ARM Cortex-M4 CPU
  - 256 Kbytes of FLASH
  - Many peripherals such as MC PWMs, 1-MSPS ADCs, eight UARTs, four SPIs, four I2Cs, USB Host|Device, and up to 27 timers.

- **EK-LM4F232 Development Kit**
  - ARM® Cortex™-LX4F232
  - color OLED display,
  - USB OTG,
  - A micro SD card, a coin cell battery,
  - A temperature sensor,
  - A three axis
4.2 ARM Cortex-M3 – TM4F

- **Connectivity features:**
  - CAN, USB Device, SPI/SSI, I2C, UARTs

- **High-performance analog integration**
  - Two 1 MSPS 12-bit ADCs
  - Analog and digital comparators

- **Best-in-class power consumption**
  - As low as 370 μA/MHz
  - 500μs wakeup from low-power modes
  - RTC currents as low as 1.7μA

- **Solid roadmap**
  - Higher speeds, Ultra-low power
  - Larger memory
4.3. ARM Cortex-M4 - LM4F120H5QR

- LM4F120H5QR package
4.3. ARM Cortex-M4 - LM4F120H5QR

- LM4F120H5QR has **6 GPIO blocks**, supporting up to **43 IO pins**
  - Port A: 8 bits
  - Port B: 8 bits
  - Port C: 8 bits
  - Port D: 8 bits
  - Port E: 6 bits
  - Port F: 5 bits

- GPIO pad configuration
  - Weak pull-up or pull-down resistors
  - 2-mA, 4-mA, and 8-mA pad drive
  - Slew rate control for 8-mA pad drive
  - Open drain enables
  - Digital input enables
4.3. ARM Cortex-M4 - LM4F120H5QR

- 256KB Flash memory
  - Single-cycle to **40MHz**
  - Pre-fetch buffer and speculative branch improves performance above 40 MHz

- 32KB single-cycle SRAM with bit-banding
  - Internal ROM loaded with StellarisWare software
  - Stellaris Peripheral Driver Library
  - Stellaris Boot Loader
  - Advanced Encryption Standard (AES) cryptography tables
  - Cyclic Redundancy Check (CRC) error detection functionality

- 2KB EEPROM (fast, saves board space)
  - Wear-leveled 500K program/erase cycles
  - 10 year data retention
  - 4 clock cycle read time
4.3. ARM Cortex-M4 - LM4F120H5QR

- Clock and reset
4.3. ARM Cortex-M4 - LM4F120H5QR
### USB Device Signals

<table>
<thead>
<tr>
<th>GPIO Pin</th>
<th>Pin Function</th>
<th>USB Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD4</td>
<td>USB0DM</td>
<td>D-</td>
</tr>
<tr>
<td>PD5</td>
<td>USB0DP</td>
<td>D+</td>
</tr>
</tbody>
</table>

### Stellaris® In-Circuit Debug Interface (ICDI) Signals

<table>
<thead>
<tr>
<th>GPIO</th>
<th>Pin Pin Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC0</td>
<td>TCK/SWCLK</td>
</tr>
<tr>
<td>PC1</td>
<td>PC1 TMS/SWDO</td>
</tr>
<tr>
<td>PC2</td>
<td>TDI</td>
</tr>
<tr>
<td>PC3</td>
<td>PC3 TDO/SWO</td>
</tr>
</tbody>
</table>

### Virtual COM Port Signals

<table>
<thead>
<tr>
<th>GPIO Pin</th>
<th>Pin Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA0</td>
<td>U0RX</td>
</tr>
<tr>
<td>PA1</td>
<td>U0TX</td>
</tr>
</tbody>
</table>
### Virtual COM Port Signals

<table>
<thead>
<tr>
<th>GPIO Pin</th>
<th>Pin Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA0</td>
<td>U0RX</td>
</tr>
<tr>
<td>PA1</td>
<td>U0TX</td>
</tr>
</tbody>
</table>
4.3. ARM Cortex-M4 – STM32F4

• **Features:**
  
  – 180 MHz/225 DMIPS Cortex-M4
  
  – Single cycle DSP MAC and floating point unit
  
  – Memory accelerator
  
  – Graphic accelerator
  
  – Multi DMA controllers
  
  – SDRAM interface support
  
  – Ultra-low dynamic power in Run mode: 260 μA/MHz at 180 MHz
4.3. ARM Cortex-M4 – STM32F4

- Block diagram
4.3. ARM Cortex-M4 – STM32F4

• Development kit
  – STM32F407VGT6 microcontroller
  – 168MHz/210 DMIPS
  – DSP MAC and floating point unit
  – 1 MB Flash, 192 KB RAM
  – On-board ST-LINK/V2
  – Power supply: 3 V and 5 V
  – 3-axis accelerometer
  – Audio sensor, omni-directional digital microphone
  – audio DAC with integrated class D speaker driver
  – Eight LEDs:
  – Two push buttons (user and reset)
4.3. ARM Cortex-M4 – STM32F4

- Package LQFP100
- GPIO
  - Port A: 16 bit
  - Port B: 16 bit
  - Port C: 16 bit
  - Port D: 16 bit
  - Port E: 16 bit
- can sink or source up to ±8mA
- except PC13, PC14 and PC15 which can sink or source up to ±3mA
4.3. ARM Cortex-M4 – STM32F4

Typical application with an 8 MHz crystal

Typical application with a 32.768 kHz crystal
4.3. ARM Cortex-M4 – STM32F4

• Reset circuit
5. 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, Code Composer, IAR
5. ARM Programming

• Typical development flow
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 ; R1 = R1 + R0
          SUBS r0, #1 ; Decrement R0, update flag ("S" suffix)
          BNE loop ; If result not zero jump to loop,
deadloop  ; result is now in R1
          B deadloop ; Infinite loop
END       ; End of file
```
5. ARM Programming - Simple program

• Compile a assembly 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
5. 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
```
5. 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
```
5. 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
    ...
    BX LR ; Return ; Program code for function 1
5. ARM Programming – Special Register

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

\[
\begin{align*}
\text{MRS} & \quad <\text{reg}>, <\text{special\_reg}> & \text{; Read special register} \\
\text{MSR} & \quad <\text{special\_reg}>, <\text{reg}> & \text{; write to special register}
\end{align*}
\]

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

\[
\begin{align*}
\text{MRS} & \quad r0, APSR & \text{; Read Flag state into R0} \\
\text{MRS} & \quad r0, IPSR & \text{; Read Exception/Interrupt state} \\
\text{MRS} & \quad r0, EPSR & \text{; Read Execution state} \\
\text{MSR} & \quad APSR, r0 & \text{; Write Flag state} \\
\text{MRS} & \quad r0, PSR & \text{; Read the combined program status word} \\
\text{MSR} & \quad PSR, r0 & \text{; Write combined program state word}
\end{align*}
\]
5. 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
```

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
</table>
| 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. |
5. ARM Programming

- 16-Bit Load and Store Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR</td>
<td>Load word from memory to register</td>
</tr>
<tr>
<td>LDRH</td>
<td>Load halfword from memory to register</td>
</tr>
<tr>
<td>LDRB</td>
<td>Load byte from memory to register</td>
</tr>
<tr>
<td>LDRSH</td>
<td>Load halfword from memory, sign extend it, and put it in register</td>
</tr>
<tr>
<td>LDRSB</td>
<td>Load byte from memory, sign extend it, and put it in register</td>
</tr>
<tr>
<td>STR</td>
<td>Store word from register to memory</td>
</tr>
<tr>
<td>STRH</td>
<td>Store half word from register to memory</td>
</tr>
<tr>
<td>STRB</td>
<td>Store byte from register to memory</td>
</tr>
<tr>
<td>LDMIA</td>
<td>Load multiple increment after</td>
</tr>
<tr>
<td>STMIA</td>
<td>Store multiple increment after</td>
</tr>
<tr>
<td>PUSH</td>
<td>Push multiple registers</td>
</tr>
<tr>
<td>POP</td>
<td>Pop multiple registers</td>
</tr>
</tbody>
</table>
5. ARM Programming

• 16-Bit Branch Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Branch</td>
</tr>
<tr>
<td>B&lt;cond&gt;</td>
<td>Conditional branch</td>
</tr>
<tr>
<td>BL</td>
<td>Branch with link; call a subroutine and store the return address in LR</td>
</tr>
<tr>
<td>BLX</td>
<td>Branch with link and change state (BLX &lt;reg&gt; only)(^1)</td>
</tr>
<tr>
<td>CBZ</td>
<td>Compare and branch if zero (architecture v7)</td>
</tr>
<tr>
<td>CBNZ</td>
<td>Compare and branch if nonzero (architecture v7)</td>
</tr>
<tr>
<td>IT</td>
<td>IF-THEN (architecture v7)</td>
</tr>
</tbody>
</table>
### 5. ARM Programming – Arithmetic Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD Rd, Rn, Rm ; Rd = Rn + Rm</td>
<td>ADD operation</td>
</tr>
<tr>
<td>ADD Rd, Rm     ; Rd = Rd + Rm</td>
<td></td>
</tr>
<tr>
<td>ADD Rd, #immed ; Rd = Rd + #immed</td>
<td></td>
</tr>
<tr>
<td>ADC Rd, Rn, Rm ; Rd = Rn + Rm + carry</td>
<td>ADD with carry</td>
</tr>
<tr>
<td>ADC Rd, Rm     ; Rd = Rd + Rm + carry</td>
<td></td>
</tr>
<tr>
<td>ADC Rd, #immed ; Rd = Rd + #immed + carry</td>
<td></td>
</tr>
<tr>
<td>ADDW Rd, Rn,#immed ; Rd = Rn + #immed</td>
<td>ADD register with 12-bit immediate value</td>
</tr>
<tr>
<td>SUB  Rd, Rn, Rm     ; Rd = Rn - Rm</td>
<td>SUBTRACT</td>
</tr>
<tr>
<td>SUB  Rd, #immed ; Rd = Rd - #immed</td>
<td></td>
</tr>
<tr>
<td>SUB  Rd, Rn,#immed ; Rd = Rn -#immed</td>
<td></td>
</tr>
<tr>
<td>SBC  Rd, Rm     ; Rd = Rd - Rm - carry flag</td>
<td>SUBTRACT with borrow (carry)</td>
</tr>
<tr>
<td>SBC.W Rd, Rn, #immed ; Rd = Rn - #immed - carry flag</td>
<td></td>
</tr>
<tr>
<td>SBC.W Rd, Rn, Rm ; Rd = Rn - Rm - carry flag</td>
<td></td>
</tr>
<tr>
<td>RSB.W Rd, Rn, #immed ; Rd = #immed -Rn</td>
<td>Reverse subtract</td>
</tr>
<tr>
<td>RSB.W Rd, Rn, Rm     ; Rd = Rm - Rn</td>
<td></td>
</tr>
<tr>
<td>MUL  Rd, Rm     ; Rd = Rd * Rm</td>
<td>Multiply</td>
</tr>
<tr>
<td>MUL.W Rd, Rn, Rm ; Rd = Rn * Rm</td>
<td></td>
</tr>
</tbody>
</table>
5. 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:

\[
\begin{align*}
&\text{IT}<x> <\text{cond}> \\
&\text{IT}<x><y> <\text{cond}> \\
&\text{IT}<x><y><z> <\text{cond}>
\end{align*}
\]

where:
- \(<x> \) specifies the execution condition for the second instruction
- \(<y> \) specifies the execution condition for the third instruction
- \(<z> \) specifies the execution condition for the fourth instruction
## 5. ARM Programming – IF-THEN

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

• An example of a simple conditional execution

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

• In assembly:

```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
```
### 5. ARM Programming – Using Data Memory

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STACK_TOP</td>
<td>EQU 0x20002000 ; constant for SP starting value</td>
</tr>
<tr>
<td>AREA</td>
<td>Header Code</td>
</tr>
<tr>
<td>DCD STACK_TOP ; SP initial value</td>
<td></td>
</tr>
<tr>
<td>DCD Start ; Reset vector</td>
<td></td>
</tr>
<tr>
<td>ENTRY</td>
<td></td>
</tr>
<tr>
<td>Start ; Start of main program, initialize registers</td>
<td></td>
</tr>
<tr>
<td>MOV r0, #10 ; Starting loop counter value</td>
<td></td>
</tr>
<tr>
<td>MOV r1, #0 ; starting result. Calculated 10+9+8+...+1</td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td></td>
</tr>
<tr>
<td>ADD r1, r0 ; R1 = R1 + R0</td>
<td></td>
</tr>
<tr>
<td>SUBS r0, #1 ; Decrement R0, update flag (“S” suffix)</td>
<td></td>
</tr>
<tr>
<td>BNE loop ; If result not zero jump to loop; Result is now in R1</td>
<td></td>
</tr>
<tr>
<td>LDR r0,=MyData1 ; Put address of MyData1 into R0</td>
<td></td>
</tr>
<tr>
<td>STR r1,[r0] ; Store the result in MyData1</td>
<td></td>
</tr>
<tr>
<td>deadloop</td>
<td></td>
</tr>
<tr>
<td>B deadloop ; Infinite loop</td>
<td></td>
</tr>
<tr>
<td>AREA</td>
<td>Header Data</td>
</tr>
<tr>
<td>ALIGN 4</td>
<td></td>
</tr>
<tr>
<td>MyData1</td>
<td>DCD 0 ; Destination of calculation result</td>
</tr>
<tr>
<td>MyData2</td>
<td>DCD 0</td>
</tr>
<tr>
<td>END ; End of file</td>
<td></td>
</tr>
</tbody>
</table>
5. 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
5. 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
          LDR R1,=UART0_FLAG

PutcWaitLoop
          LDR R2,[R1] ; Get status flag
          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
TI’s ARM Cortex-M Development Kit

- **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

- **The Tiva C Series EK-TM4C123GXL LaunchPad Evaluation Kit**
  - A TM4C123G LaunchPad Evaluation board
  - On-board In-Circuit Debug Interface (ICDI)
  - USB Micro-B plug to USB-A plug cable
  - Preloaded RGB quickstart application
  - ReadMe First quick-start guide
Quiz

1. What are different features between ARM Cortex M3 and M4?
2. What are differences between Thumb and Thumb-2 instructions?
3. Compare the features between TM4C and STM32F4 microcontroller
4. What are extra instructions that ARM Cortex-M4 supports?
Assignments

1. Write a program to move 10 words from 0x20000000 to 0x30000000.
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 from 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
   }
Assignments

• Design a circuit described as follows:
  – Using Cortex-M4 processor LM4F120H5QR
  – Port A connects to 8 single LEDs
  – Port B connects to 8 buttons
  – Write a program to control 8 LEDs by 8 buttons
Assignments

• Design a circuit described as follows:
  – Using Cortex-M4 processor STM32F407VGT6
  – Port A connects to a character LCD
  – Port B connects to 3 buttons START, STOP, CLEAR
  – Write a program to control as follows:
    • START: start to count number in millisecond
    • STOP: stop to count
    • CLEAR: clear the number to zero

![Character LCD Diagram]