ARM Architecture

Acknowledgement: Part of the slides come from EECS373 Univ. Michigan
Architecture

- Computer Organization (or Microarchitecture)
  - Control and data paths
  - I/D pipeline design
  - Cache design
  - ...

- System Design (or Platform Architecture)
  - Memory and I/O buses
  - Memory controllers
  - Direct memory access
  - ...

- Instruction Set Architecture (ISA)
Cortex-M0

- Simplest, smallest “current generation” ARM
- 85uW/MHz
- 12k gates
- 56 instructions
  - subset of M3/M4; Thumb and some Thumb 2
- 3-stage pipeline
- Interrupts: NMI+1-32 physical interrupts
- Complex hardware Ops
  - single-cycle 32-x32 multiply
Cortex-M0

Instruction Set
- ADD Rd, Rn, <op2>

Branching
Data processing
Load/Store
Exceptions
Miscellaneous

Register Set
- R0
- R1
- R2
- R3
- R4
- R5
- R6
- R7
- R8
- R9
- R10
- R11
- R12
- R13 (SP)
- R14 (LR)
- R15 (PC)
- xPSR

Address Space
- System
- Private peripheral bus - External
- Private peripheral bus - Internal
- External device 1.0GB
- External RAM 1.0GB
- Peripheral 0.5GB
- SRAM 0.5GB
- Code 0.5GB
- Data 0.5GB

32-bits
Endianess

32-bits
Endianess
mov r0, #1
ld r1, [r0,#5]
mem((r0)+5)
bne loop
subs r2, #1
ARM Registers

Low registers
- R0
- R1
- R2
- R3
- R4
- R5
- R6
- R7
- R8
- R9
- R10
- R11
- R12

High registers
- Stack Pointer (SP)
- Link Register (LR)
- Program Counter (PC)

General-purpose registers
- PSP
- MSP

Special registers
- Program status register (PSR)
- Exception mask registers (PRIMASK, FAULTMASK, BASEPRI)
- CONTROL register

Banked version of SP
ARM Registers

Figure 4. PSR bit assignments
Interrupt

- In systems programming, an interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.

- An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.

- The processor responds by suspending its current activities, saving its state, and executing a small program called an interrupt service routine (ISR) to deal with the event.
# Exceptions and Interrupts

<table>
<thead>
<tr>
<th>Exception Type</th>
<th>Exception Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>1</td>
<td>Power on reset or system reset.</td>
</tr>
<tr>
<td>NMI</td>
<td>2</td>
<td>Nonmaskable interrupt—highest priority exception that cannot be disabled. For safety critical events.</td>
</tr>
<tr>
<td>Hard fault</td>
<td>3</td>
<td>For fault handling—activated when a system error is detected.</td>
</tr>
<tr>
<td>SVCALL</td>
<td>11</td>
<td>Supervisor call—activated when SVC instruction is executed. Primarily for OS applications.</td>
</tr>
<tr>
<td>PendSV</td>
<td>14</td>
<td>Pendable service (system) call—activated by writing to an interrupt control and status register. Primarily for OS applications.</td>
</tr>
<tr>
<td>SysTick</td>
<td>15</td>
<td>System Tick Timer exception—typically used by an OS for a regular system tick exception. The system tick timer (SysTick) is an optional timer unit inside the Cortex-M0 processor.</td>
</tr>
<tr>
<td>IRQ0 to IRQ31</td>
<td>16 - 47</td>
<td>Interrupts—can be from external sources or from on-chip peripherals.</td>
</tr>
</tbody>
</table>
The Cortex-M0 processor contains a built-in interrupt controller, which supports up to 32 interrupt request (IRQ) inputs,
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Exception Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000000</td>
<td></td>
</tr>
<tr>
<td>0x0000000004</td>
<td>Interrupt#3 vector</td>
</tr>
<tr>
<td>0x0000000008</td>
<td>Interrupt#2 vector</td>
</tr>
<tr>
<td>0x000000000C</td>
<td>Interrupt#1 vector</td>
</tr>
<tr>
<td>0x0000000010</td>
<td>Interrupt#0 vector</td>
</tr>
<tr>
<td>0x0000000014</td>
<td>SysTick vector</td>
</tr>
<tr>
<td>0x0000000018</td>
<td>PendSV vector</td>
</tr>
<tr>
<td>0x000000001C</td>
<td>Not used</td>
</tr>
<tr>
<td>0x0000000020</td>
<td>Not used</td>
</tr>
<tr>
<td>0x0000000024</td>
<td>Not used</td>
</tr>
<tr>
<td>0x0000000028</td>
<td>Not used</td>
</tr>
<tr>
<td>0x000000002C</td>
<td>SVC vector</td>
</tr>
<tr>
<td>0x0000000030</td>
<td>Not used</td>
</tr>
<tr>
<td>0x0000000034</td>
<td>Not used</td>
</tr>
<tr>
<td>0x0000000038</td>
<td>Not used</td>
</tr>
<tr>
<td>0x000000003C</td>
<td>Not used</td>
</tr>
<tr>
<td>0x0000000040</td>
<td>Not used</td>
</tr>
<tr>
<td>0x0000000044</td>
<td>Not used</td>
</tr>
<tr>
<td>0x0000000048</td>
<td>Not used</td>
</tr>
<tr>
<td>0x000000004C</td>
<td>Not used</td>
</tr>
<tr>
<td>0x0000000050</td>
<td>Hard Fault vector</td>
</tr>
<tr>
<td>0x0000000054</td>
<td>NMI vector</td>
</tr>
<tr>
<td>0x0000000058</td>
<td>Reset vector</td>
</tr>
<tr>
<td>0x000000005C</td>
<td>MSP initial value</td>
</tr>
</tbody>
</table>

Note: LSB of each vector must be set to 1 to indicate Thumb state.
How to boot Cortex-M0

Program memory

Interrupt vectors
- SysTick vector: 0x00000040
- PendSV vector: 0x0000003C
- SVC vector: 0x00000038
- reserved: 0x00000038
- reserved: 0x00000038
- Hard fault vector: 0x00000000C
- NMI vector: 0x000000008
- Reset vector: 0x000000004
- Initial MSP value: 0x00000000

Program code

Vector table

Program image

0x00000000

0x000000000
On the ARM Cortex-M0 SP and PC are loaded from the code (.text) segment

**Initial stack pointer**
- LOC: 0x00000000
- SP mem(0x00000000)

**Interrupt vector table**
- *Initial* base: 0x00000004
- Vector table is relocatable
- Entries: 32-bit values
- Each entry is an address
- Entry #1: reset vector
  - LOC: 0x00000004
  - PC mem(0x00000004)

**Execution begins**
Bootloader

- Initialize the “core”, system clock, I/O clock
- Initialize I/Os (considering GPIO)
  - Clock enabling
  - Functional configuration
  - Enable interrupt
Nested Vectored Interrupt Controller (NVIC)

- Flexible interrupt management include enable/disable, priority configurations
- Hardware nested interrupt support
- Vectored exception entry
- Interrupt masking
## Interrupt enable/disable

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Type</th>
<th>Reset Value</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| 0xE000E100 | SETENA | R/W  | 0x00000000 | Set enable for interrupt 0 to 31. Write 1 to set bit to 1, write 0 has no effect.  
Bit[0] for Interrupt #0 (exception #16)  
Bit[1] for Interrupt #1 (exception #17)  
...  
Bit[31] for Interrupt #31 (exception #47)  
Read value indicates the current enable status |
| 0xE000E180 | CLRENA | R/W  | 0x00000000 | Clear enable for interrupt 0 to 31. Write 1 to clear bit to 0, write 0 has no effect.  
Bit[0] for Interrupt #0 (exception #16)  
...  
Bit[31] for Interrupt #31 (exception #47)  
Read value indicates the current enable status |
Interrupt enable/disable

in C,

```c
*((volatile unsigned long *)(0xE000E100))=0x4; //Enable interrupt #2
```

in assembly,

```
LDR    R0, =0xE000E100          ;Setup address in R0
MOVS   R1, #0x4                  ;interrupt #2
STR    R1,[R0]                   ;write to set interrupt enable
```
Interrupt enable/disable

in C,

```c
*((volatile unsigned long *)(0xE000E180))=0x4;//Disable interrupt #2
```

in assembly,

```
LDR R0, =0xE000E180 ;Setup address in R0
MOVS R1, #0x4 ;interrupt #2
STR R1,[R0] ;write to set interrupt disable
```
Interrupt pending

in C,

```
*((volatile unsigned long *)(0xE000E200))=0x4;//Set interrupt #2 pending
```

in assembly,

```
LDR      R0, =0xE000E200         ;Setup address in R0
MOVS   R1, #0x4                      ;interrupt #2
STR      R1,[R0]                        ;write to set interrupt #2 pending
```
Interrupt pending

in C,

\ star((volatile unsigned long \*) (0xE000E280))=0x4; //Clear interrupt #2 pending

in assembly,

LDR R0, =0xE000E280 ;Setup address in R0
MOVS R1, #0x4 ;interrupt #2
STR R1,[R0] ;write to clear interrupt #2 pending
Interrupt Priority

<table>
<thead>
<tr>
<th>Bit</th>
<th>31</th>
<th>30</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xE000E41C</td>
<td>31</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>0xE000E418</td>
<td>27</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>0xE000E414</td>
<td>23</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>0xE000E410</td>
<td>19</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>0xE000E40C</td>
<td>15</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>0xE000E408</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>0xE000E404</td>
<td>7</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>0xE000E400</td>
<td>IRQ 3</td>
<td>IRQ 2</td>
<td>IRQ 1</td>
<td>IRQ 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The ARM® Cortex™ Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series. The CMSIS enables consistent and simple software interfaces to the processor and the peripherals, simplifying software re-use, reducing the learning curve for new microcontroller developers and reducing the time to market for new devices. Creation of software is acknowledged as a major cost factor by the embedded industry. By standardizing the software interfaces across all Cortex-M silicon vendor products, this cost is significantly reduced, especially when creating new projects or migrating existing software to a new device.
# CMSIS Interrupt Support

<table>
<thead>
<tr>
<th>Function definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void NVIC_SystemReset (void)</code></td>
<td>Resets the whole system including peripherals.</td>
</tr>
<tr>
<td><code>void NVIC_EnableIRQ(IRQn_Type IRQn)</code></td>
<td>Enables the interrupt IRQn.</td>
</tr>
<tr>
<td><code>void NVIC_DisableIRQ (IRQn_Type IRQn)</code></td>
<td>Disables the interrupt IRQn.</td>
</tr>
<tr>
<td><code>void NVIC_SetPriority (IRQn_Type IRQn, int32_t priority)</code></td>
<td>Sets the priority for the interrupt IRQn.</td>
</tr>
<tr>
<td><code>uint32_t NVIC_GetPriority (IRQn_Type IRQn)</code></td>
<td>Returns the priority for the specified interrupt.</td>
</tr>
<tr>
<td><code>void NVIC_SetPendingIRQ (IRQn_Type IRQn)</code></td>
<td>Sets the interrupt IRQn pending.</td>
</tr>
<tr>
<td><code>IRQn_Type NVIC_GetPendingIRQ (IRQn_Type IRQn)</code></td>
<td>Returns the pending status of the interrupt IRQn.</td>
</tr>
<tr>
<td><code>void NVIC_ClearPendingIRQ (IRQn_Type IRQn)</code></td>
<td>Clears the pending status of the interrupt IRQn, if it is not already running or active.</td>
</tr>
</tbody>
</table>
Why CMSIS

LDR R0,=0xE000E400;   // Setup address in R0
LDR R1,[R0];          // PRIORITY0
MOVS R2, #0xFF;       // Byte mask
LSLS R2, R2, #16;     // Shift mask to interrupt #2’s position
BICS R1, R1, R2;      // R1 = R1 AND (NOT(0x00FF0000))
MOVS R2, #0xC0;       // New value for priority level
LSLS R2, R2, #16;     // Shift left by 16 bits
ORR SR1, R1, R2;      // Put new priority level
STR  R1,[R0];         // write back value

void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority);  // Set the priority level of an interrupt or a system exception
ISA

- A “contract” between architects and programmers
- Instruction set
- Register set
- Memory and addressing modes
- Word sizes
- Data formats
- Operating modes
- Condition codes
- Calling conventions
ISA

- Branching
- Data processing
- Load/store
- Exceptions
- Miscellaneous