Quick Reference

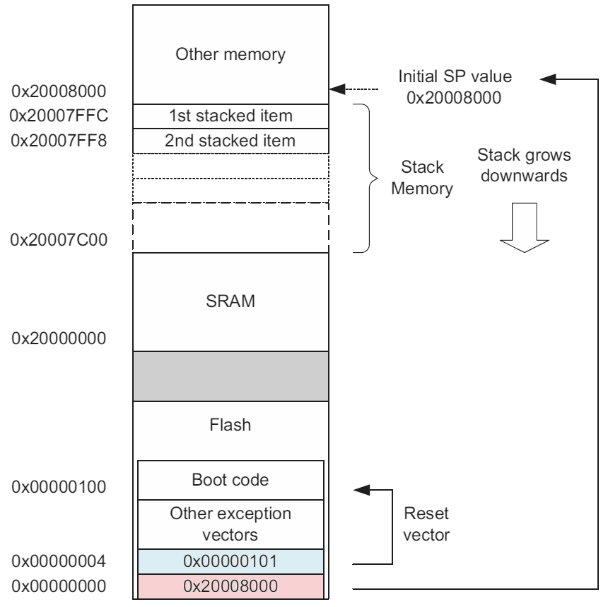
Lars Kamm, Jan Wendler, Martina Knobel

# ARM Cortex-M Memory Organization

## ARM Cortex-M Memory-Map



## Vector-Table & Reset-Sequence



Init. Programm Counter

## Register-Set & Program Status Register

N Negativ Flag

Z Zero Flag

C Carry/Borrow Flag

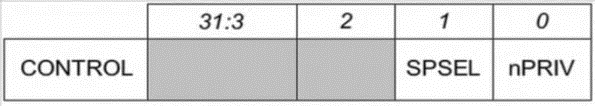
V Overflow Flag

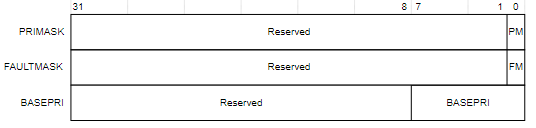
Q Sticky Saturation Flag

ICI/IT Interrupt-Continuable Instruction (ICI) bits, IF-THEN instruction  
status bit

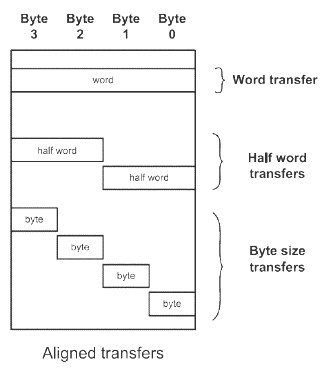
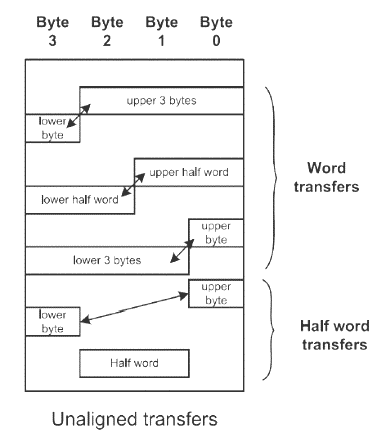
T Thumb state, always “1”; trying to clear this bit will cause a fault  
exception

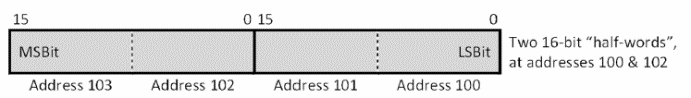
Exception# indicates which exception the processor is handling



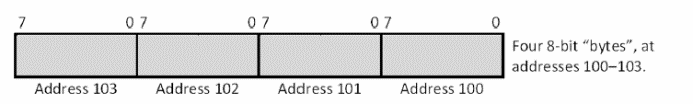


## xPSR_and_FlagsAlignement



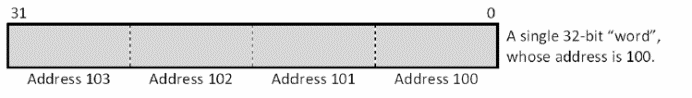


**Word (32Bit):** 0x0, 0x4, 0x8 0xC

****

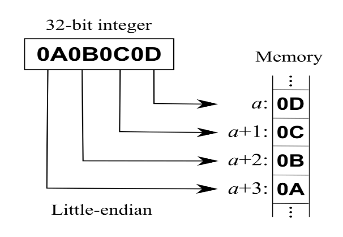
**Halfword (16Bit):** 0x0, 0x2, 0x4, 0x6

0x8, 0xA, 0xC, 0xE



**Byte (8Bit):** 0x0, 0x1, …., 0xF

## Endianess



Endianess betrifft nur Zahlenformate die grösser als ein Byte sind (Halfword, Word, Doubleword).

**Little-Endian:** Least Significant Byte befindet sich an der tiefsten Adresse.

**Big-Endian:** Most Significant Byte befindet sich an der tiefsten Adresse.

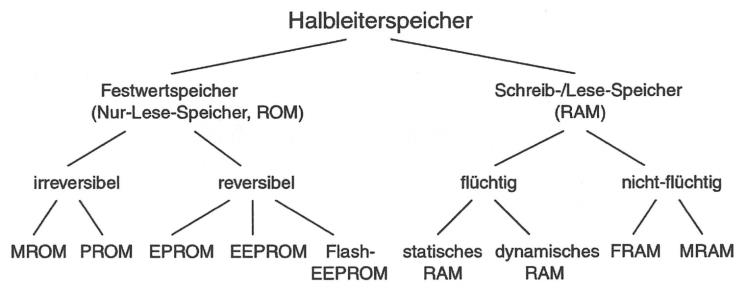
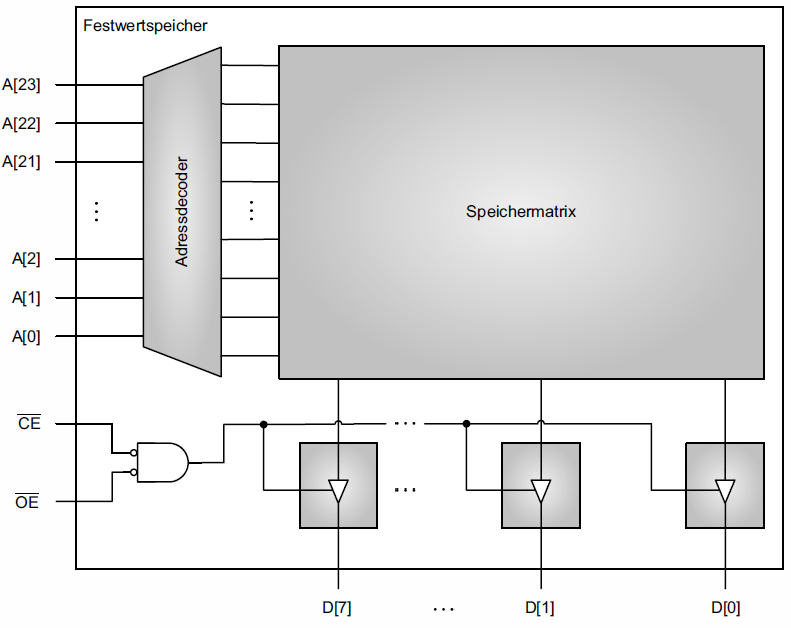
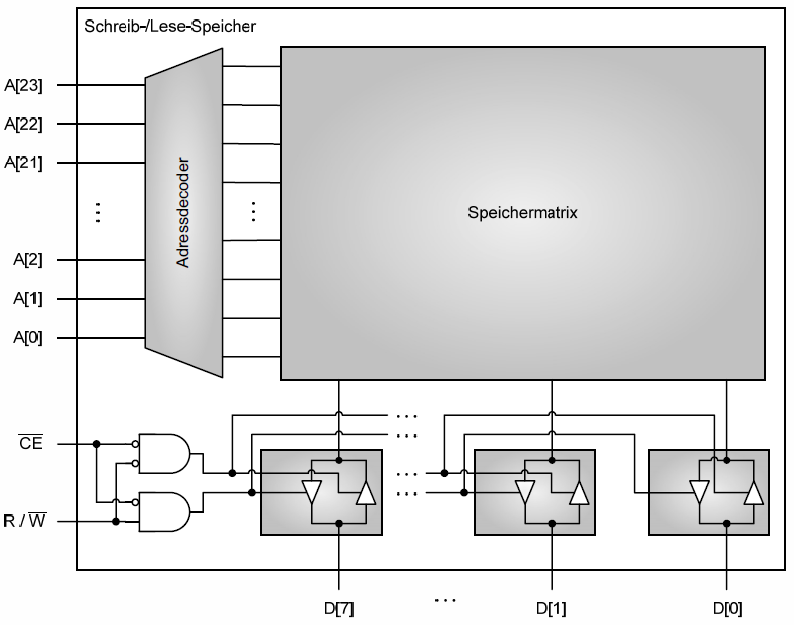
## Speichergrössen

Zur Angabe der Speicherkapazität als Anzahl von Bits, Bytes oder Codewörtern verwendet man in der Informatik in Anlehnung an die Physik die Bezeichnungen Kilo (K), Mega (M), Giga (G) und Tera (T). Diese beziehen sich auf das Binärzahlensystem:

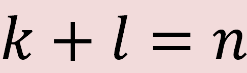
SpyKap = 2#Adressleitungen · #Datenleitungen

Evtl. aufrunden auf nächste 2er-Potenz

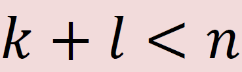
## Speicherarten



## Adressdekodierung









# Kontroll- & Selektionsstrukturen

## Nassi Schneidermann Diagramme

Merkmal: Temporäre Index-Variable mit Startwert; wird nur bei bedingter Erfüllung angepasst



nicht abweisend

abweisend



**If Then Else**

+ Variablen auf viele Arten vergleichen (==, <=, usw.)

+ Vergleichen mehrerer Var pro „if“ möglich.

+ Floating-Point-Variablen evaluierbar

**Switch Case**+ Lookup-Tab. 🡪 schneller als bei   
 verschachtelten If-Then-Else-Anweisungen

+ Übersichtlicher (weniger Codezeilen für Case)

+ Zeit für Erreichen eines Cases immer gleich,

nicht vom Case-Label abhängig.

## Entscheidungen treffen

// a 🡪 R0

// b 🡪 R3

if(a != 1)

{

b = 1;

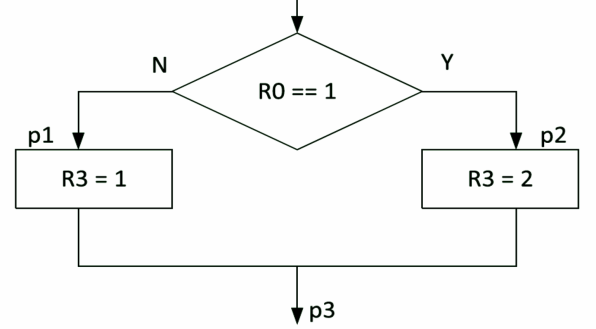
}

else

{

b = 2;

}



;compare r0 to 1

;if equal, go to p2

;then r3 = 1

;go to end (p3)

;label p2

;else r3 = 2

;label p3

;other operations

**CMP** r0, #1

**BEQ** p2

**MOVS** r3, #1

**B** p3

p2:

**MOVS** r3, #2

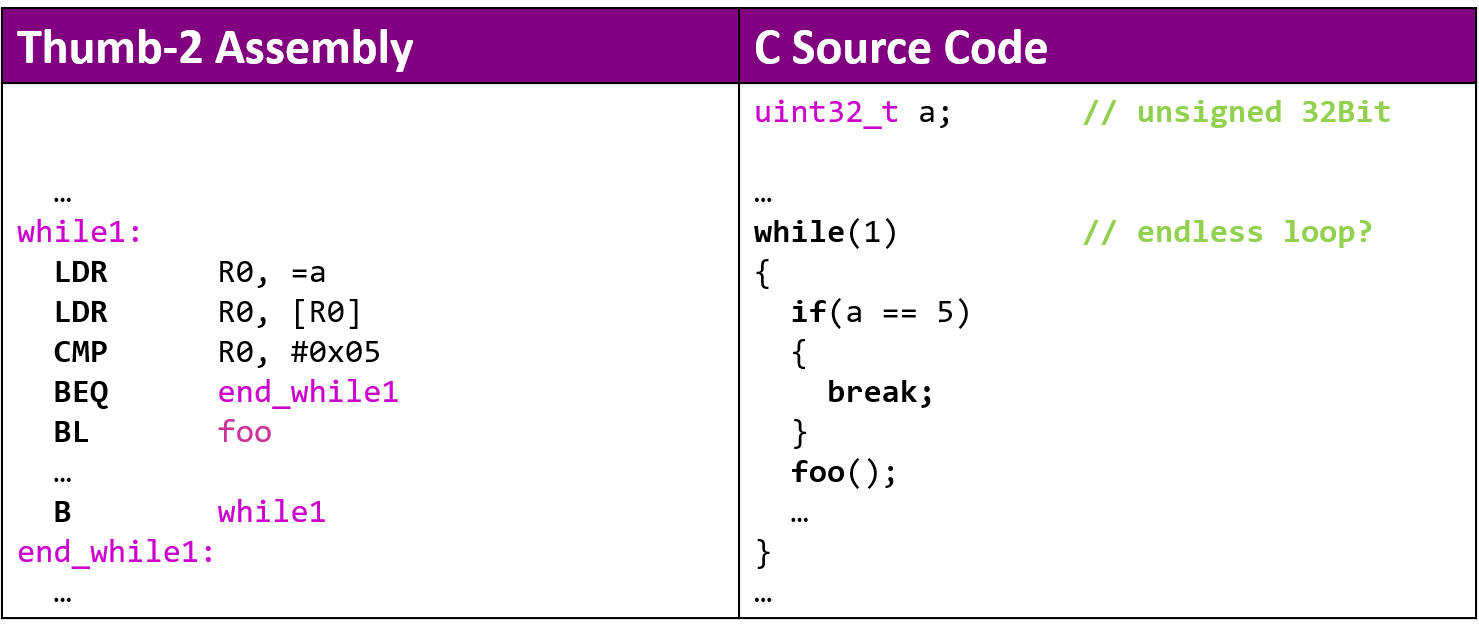
p3:

…

**In Assembler-Sprache ist eine Entscheidung praktisch immer ein 2-stufiger Ablauf**

1. **Benötigte Flags ermitteln**Vergleich (**Compare**) vornehmen, indem zwei Werte voneinander subtrahiert werden. Bei der Differenzbildung ist das Resultat nicht wichtig, sondern nur dessen Eigenschaften (**Flags**).
2. **Zugehörige Sprünge ausführen**Im zweiten Schritt werden mit den Flags bedingte Sprünge (**B{cond}**)ausgeführt

## Umsetzung von C/C++ Strukturen



***Adrdress of variable a***

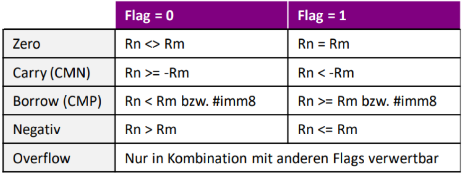
# ARM CORTEX-M0(+) Instruction-set

## Symbole & Appendix

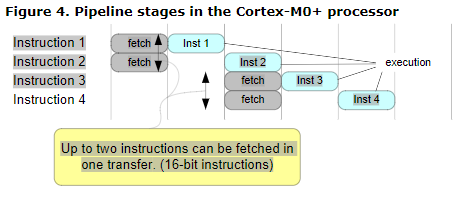
|  |  |
| --- | --- |
| **Symbols** | **Meaning** |
| **Rd Rm Rn** | represent 32-bit registers |
| **{Rd,}** | if Rd is present Rd is destination, otherwise Rn |
| **{S}** | if S is present, instruction will set condition codes (<S> setzt Flags) |
| **{cc}** | optional logical condition, condition code suffix |
| **#imm3** | any value in the range: 0...7 |
| **#imm5** | any value in the range: 0…31  N: wenn Bit31 logisch «1» Z: wenn alle Bits logisch «0» evtl. weitere Flags |
| **#imm7** | any value in the range: 0..127 |
| **#imm8** | any value in the range: 0..255 |
| **#imm10** | any value in the range: 0..1023 |
| **#imm11** | any value in the range: 0..2047 |
| **label** | any address within the ROM of the microcontroller; offset range relative to PC:  B *label* −2 KB to +2 KB B<cc> *label* −256 bytes to +255 bytes  BL *label* −16 MB to +16 MB |
| **{reglist}** | List of registers, sequence is not relevant |

## Bedingte Ausführung & Code-Zusätze {cc}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **cc** | **Meaning** | **Flags (APSR) / Requirements** | **Binary** | **uint** | | **int** | |
| **EQ** | Equal | Z = 1 | 0000 | X | == | X | == |
| **NE** | Not equal | Z = 0 | 0001 | X | != | X | != |
| CS or **HS** | Carry set, Unsigned ≥ | C = 1 | 0010 | X | >= |  |  |
| CC or **LO** | Carry clear,Unsigned < | C = 0 | 0011 | X | < |  |  |
| **MI** | Minus/negative | N = 1 | 0100 |  |  | X | - |
| **PL** | Positive or zero (non-negative) | N = 0 | 0101 |  |  | X | + |
| **VS** | Overflow | V = 1 | 0110 |  |  | X |  |
| **VC** | No overflow | V = 0 | 0111 |  |  | X |  |
| **HI** | Unsigned > ("Higher") | C = 1 && Z = 0 | 1000 | X | > |  |  |
| **LS** | Unsigned ≤ ("Lower or Same") | C = 0 || Z=1 | 1001 | X | <= |  |  |
| **GE** | Signed ≥ ("Greater than or Equal") | N = V | 1010 |  |  | X | >= |
| **LT** | Signed < ("Less Than") | N ≠ V | 1011 |  |  | X | < |
| **GT** | Signed > ("Greater Than") | Z = 0 && N = V | 1100 |  |  | X | > |
| **LE** | Signed ≤ ("Less than or Equal") | Z = 1 || N ≠ V | 1101 |  |  | X | <= |
| **AL** | Always (unconditional) | any | 1110 |  |  |  |  |

**B** = Branch, **BL** = Branch and Link (für Unterprogramme)

## Pipelining



Ein Bild, das Diagramm enthält.

Automatisch generierte Beschreibung

💠Vorteil: Es wird in jedem Cycle ein Befehl ausgeführt  
💠Nachteile: Bei Sprüngen muss Pipeline «geleert» werden, 32Bit Thumb-Befehle blockieren Pipeline

**3-stage Pipeline** (bei jedem Schritt decode (auch bei 0), fetch bereits bei Inst1)  
2x 16Bit Instructions gleichzeitig geladen  
Infinite Loop 3 Cycles

**2-stage Pipeline** (optimal für Von Neumann-Architektur)  
2x 16Bit Instructions gleichzeitig geladen  
Infinite Loop 2 Cycles

## ARM Cortex-M0(+) Thumb Instruktionen



from low register

1. Depends on multiplier implementation
2. 2 if to AHB interface or SCS, 1 if to single-cycle I/O port.
3. N is the number of elements in the list
4. N is the number of elements in the list including PC or LR.



with addition

1. 2 if taken, 1 if not-take
2. Cycle count depends on processor and debug configuration.
3. Excludes time spent waiting for an interrupt or event.
4. Executes as NOP.

# Zahlensysteme

Ein Bild, das Text, Diagramm, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

## Vorzeichenlose Zahlen

**Dualcode (unsigned int)**

**Relevante Flags:** Zero, Carry/Borrow

## Vorzeichenbehaftete Zahlen

**Zweierkomplement (signed int)**

**Relevante Flag:** Zero, Negativ, Overflow

## Zweierkomplement-Negierung

**Vorgehensweise:**

**Beispiel: -310= 1012**

Invertieren: 0102

+1: 0012

------------------------------

0112

=================

Resultat: +310

1. Vorzeichenbehaftete Zahl in Binärcode darstellen
2. Alle Bits einzeln invertieren
3. Binäre Addition mit dem Wert +1
4. Resultat entspricht der negierten Vorzeichenbehaftete Zahl

## Festkomma-Zahlen

**Vorzeichenlose Fixed-Point-Zahlen**

n: Vor-, m: Nachkommastelle(n)

Allgemeine Formel (Dualcode als Basis)

**Wertebereich:**

**Schrittweite:**

**Max. Diskretisierungsfehler:**

Quantisierungsfehler

**Vorzeichenbehaftete Fixed-Point-Zahlen**

Allgemeine Formel (2er-Komplement als Basis)

**Wertebereich:**

**Schrittweite:**

**Max. Diskretisierungsfehler:**

## IQ-Zahlenformate

**Integer/Quotient-Darstellung**

Signed Fixed-Point mit n = I und m = Q.

**Fraktionale Zahlen (oder Spezialfall I1Qx)**

**Wertebereich:**

**Vorzeichen:** Koeffizient

**Genauigkeit:** 2-(Q)

## Zahlenkreise

**Addition:** Zahlenvektoren hintereinander reihen (Anfang auf Ende)

Achtung: Carry invertiert!

**Subtraktion:** Zahlenvektoren stumpf gegeneinanderstellen (Ende auf Ende)

Schrittgrösse = 2n / 16

Anz. Ticks

**Unsigned Integer**

Z C

Relevante Flags: Zero, Carry/Borrow

Zahlenvektoren immer Uhrzeigersinn

**Signed Integer**

N O Z

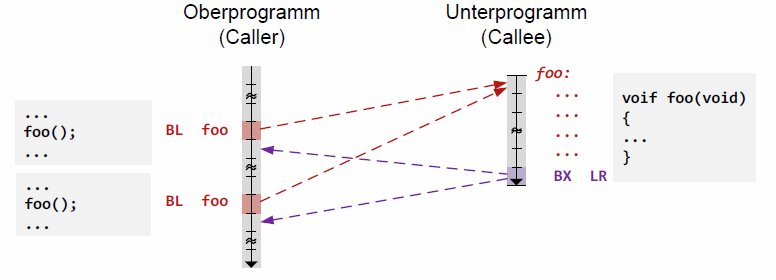
Relevante Flags: Negative, Overflow, Zero

positiv 🡪 Zahlenvektor Uhrzeigersinn  
negativ 🡪 Zahlenvektor Gegenuhrzeiger

# Subroutinen

Bewusster Aufruf durch Programmierer





**Regeln zum ARM Architecture Procedure Call Standard (AAPCS)**

* + Die Register **R0-R3** werden als Parameter verwendet.
  + Weitere Parameter werden auf den **Stack** gelegt.
  + Der Rückgabewert (≤32Bit) wird in Register **R0** übertragen.
  + Ein 64Bit-Rückgabewert wird in den Registern **R0** und **R1** übertragen.
  + Funktionen müssen die Registerinhalte **R4-R11** während der Ausführung sichern und rekonstruieren.
  + Die Register **R0-R3** und **R12** dürf(t)en in der Subroutine ohne Sicherung verändert werden.
  + Es wird stets ein **8-Byte Alignement** auf dem Stack eingehalten.

Regeln Subroutinen: R4..R11 sind immer zu sichern, wenn sie benutzt werden. R0..R3 müss(t)en nicht gesichert werden.

• R0 … R7 = Low Registers (16-Bit Befehle)

• R8 … R12 = High Registers (32-Bit Befehle) Alle Register besitzen eine Breite von 32Bit = 4Byte!

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Verwendungszweck** | **Scratch** | **Sichern** |
| **R0**  **R1**  **R2, R3**  **R4-R7**  **R8-R11**  **R12** | Erster Funktionsparameter, Return-Wert (8-/16-/32-Bit)  Zweiter Funktionsparameter, Upper Word Return-Wert (64-Bit)  Dritter und vierter Funktions-parameter  Registervariablen (temporär)  Compilerabhängige Verwendung  Funktionsinternes temporäres Speicherregister für Sprünge | Ja  Ja  Ja  Ja  Ja  Ja | Ja  Ja |
| **R13**  **R14**  **R15** | Stack-Pointer (SP, Top-of-Stack) (MSP = Main Stack Pointer, PSP = Process Stack Pointer)  Link Register (LR) (Rücksprungadresse für Funktionen und Interrupts)  Programm Counter (PC) (Adresse der auszuführenden Instruktion (Fetch)) | Diese Register werden für spezielle Zwecke verwendet | |

# Interrupts & Exceptions

Keine Parameterübergabe, keine ReturnValues

Interrupt kann zufällig auftreten 🡪 wird über die Vektortabelle gefunden

## Vector Fetch & Stacking

Ein Bild, das Text, Screenshot, Reihe, Zahl enthält.

Automatisch generierte Beschreibung

Sprung in die ISR (Cortex M0+: 15 Cycles):  
🡪 Sicherung R0-R3, R12, LR (= R14), PC (= R15), xPSR (Program Status Register)

**Request:** Pipeline «Execute» noch ausführen und der Pending Status setzen

**Stacking & Vector-Fetch:** Register auf Stack sichern und Sprung via Vektortabelle ausführen und Active Status setzen.

**ISR Handler:** Benutzercode ausführen und am Ende den Active Status zurücksetzen

**Unstacking:** Register wieder herstellen, Stack abbauen und Rücksprung mittels LR

;Example: 64bit logical shift left

**LDR** r2, =data64 ;load address of data64

**LDM** r2, {r0-r1} ;load data64 to r0, r1

**LSLS** r1, r1, #1 ;logical shift left r1 by 1

**MOVS** r3, #0 ;r3 = 0

**LSLS** r0, r0, #1 ;lsl r0 by 1 (carry = b31)

**ADCS** r1, r3 ;r1 = r1 + 0 + carry

**STM** r2!, {r0-r1} ;store r0,r1 in data64



;Example: 64bit Addition

**LDR** r4, =addrOp ;load address of int64 array [op1,op2,res]

**LDM** r4, {r0-r3} ;load op1[r1,r0] and op2[r2,r3]

**ADDS** r0, r0, r1 ;add lower words and set carry

**ADCS** r1, r2, r3 ;add higher words with carry in

**ADDS** r4, #8 ;set r4 to address of res

**STM** r4, {r0-r1} ;store 64bit result from r0, r1 to res

;Example if(a == true)

adds r3, r7, #7 ;Stackaddresse von “a” berechnen.

ldrb r3, [r3, #0] ;“a” aus Stack laden.

cmp r3, #0 ;“a” mit 0 vergleichen.

beq 0x800013e ;Wenn “a” = 0, an Addr. … springen.

;Example Stackreservierung

push {r7, lr} ; lr auf Stack “pushen”

sub sp, #16 ; Erstellung Stackframe (8-Byte Grössen)

# Personal Notes

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Begriff | Infos | | Begriff | | Infos | |
| Addressing Modes (Cortex-M0(+)) | • Register Direkt (Register zu Register)ldr r1, #2 | movs r2, r1  • Register Indirekt (Adresse auf Register) ldr r1, [r3] • Register Indirekt mit Offset (Adresse +/-) ldr r1, [r3, #4]  • PC Relativ ldr r1, [pc, #0]  • SP Relativ • Register Indirekt mit Index (Adresse mit Offset auf anderem  Register) ldr r0, [r1, r2] ;R0 = Wert an Adresse R1+R2 | | **ARM Cortex Familie** | | A: Application R: Real-Time M: Microcontroller | |
| 💠Anzahl **Zugriffe** auf welche Adressen, um einen 32Bit-Wert von der Speicherstelle 0x2000’0003 in ein Register zu laden: 2 Zugriffe auf 0x2000’0000 und auf 0x2000’0004, da der Wert auf 0x2000’0003…4…5 und 6 liegt.  **(Speicher-Zugriff** nur **auf durch 4-teilbaren Adressen!)** 🡪 nur auf diese direkt zugreifbar | | | |
| AMBA  (Advanced Micro-controller Bus Architecture) | Standard Busanbindung von Systemkomponenten (SoC, ARM-Cortex, FPGA, etc.) Hochgeschwindigkeits-Bus (AHB bzw. ASB) und einem Peripherie-Bus (APB) **Advanced High Performance-Lite (AHB-Lite)** Systembus (ADDRESS, DATA, CONTROL) **Amba AHB-Lite Read Transfer** (min. 2 Cycles) • Read Instruction: Adr von PC gesetzt • Read Data: Adr selber setzen (LDR) **Amba AHB-Lite Write Transfer** (min. 2 Cycles) • Write Data: Adr selber setzen (STR) | | **Assembler Syntax** | | • Unified Assembler Language (UAL)  • Einheitlicher Syntax für ARM & Thumb • Label: optionale Adressmarkierung  • Operanden: Parameter des Befehls • Comment: Optionaler Kommentar | |
| **Architekturen (Von-Neumann, Harvard)** | | **Von-Neumann** Nur ein Systembus für Daten und Programm 🡪 einfach, langsam! **Harvard** Separate Systembusse für Daten und  Programm 🡪 schnell, aber komplex! | |
| Bus | **Systembus** (Prozessnahe Systeme (RAM, ROM) Adressbus: Gibt die Adresse im Speicher bzw. I/O vor, an welcher gelesen oder geschrieben wird. Datenbus: Übermittelt die Daten von Speicher (ROM, RAM, I/O) zu CPU oder umgekehrt. Steuerbus: Ermöglicht die fehlerfreie Übertragung der Daten (CLK, ALE, MEMW, MEMR, IOW, IOR) **Peripheriebus** (Anbindung von Peripherie) | | **Compiling** | | Präprozessor 🡪 Compiler 🡪 Binder/Linker 🡪 Loader C/C++: 🡪 Assembly Binder/Linker: Einen Code aus mehreren Codes Loader: Wandelt rel. Adr in abs. Adr, programmiert Speicher Compiling Schritte: 1. Lexikalische Analyse (whitespace, groups) 2. Syntaxanalyse (Symbole werden zusammengefasst) 3. Semantische Analyse (Fehlerüberprüfung) 4. Zwischencode 5. Code Optimierung 6. Code Erzeugung | |
| Cortex M Vergleiche | Zusätzliche Features…  **… M4** **gegenüber M0** • DSP-Extensions (SIMD) • Floating Point Unit (FPU) • Harvard-Architektur  **… M0+ gegenüber M0** • Optimierte 2-stufige Pipeline  (🡪 Verkürzte Ausführungs- und Sprungzeit) • Schnellere Übergangszeiten für Interrupts und  Exceptions (15 statt 16 Cycles) • MPU optional erhältlich • Single-Cycle GPIO’s (für schnelle Auslesung) • Micro Trace Buffer für Debugging (Zusatzspeicher) **M0** • 2Bit-Preemption **Cortex-M3/M4**  • 3bit Preemption & Subpriority  • 4bit Preemption & Subpriority | |  | | | |
| Debug Unit | • Breakpoint Unit  *4 Adress Komperator* • Data Watchpoint Unit (DWT)  *2 Datenadress* Komperator • ROM Table  *Auflistung Debug-Komponenten* • Micro Trace Buffer  *Instruktionsverfolgung SRAM* | | **Exceptions & Interrupts** | | • IRQ’s werden durch On-Chip Peripherien initiiert • NMI über Watchdog-Timer oder Brownout-Detektor • SysTick Timer-Unit wird als Zeitgeber verwendet • Besitzen alle eine eindeutige Nummer, welche beim  Auftreten im xPSR abgelegt wird  • Mit dem Eintrag im xPSR findet der NVIC die richtige Stelle  in der Vektortabelle • Der Eintrag in der Vektortabelle ist ein Pointer auf die  entsprechende ISR | |
| Fault Mechanism | • Schnelle Reaktion auf Fehler, einleiten von Notfallszenarien | | **Hard Faults** | | • immer aktiviert • Bus, Usage & Memory Management Fault | |
| Instruktionssatz | **ARM** (16bit-Befehle)  • mehr Performance als Thumb1 🡪 mehr Speicher benötigt **Thumb** **(1)** (32bit-Befehle) **Thumb-2** (16 & 32bit-Befehle) • Spart Speicherplatz  • ermöglicht schnelle komplexe Rechenoperationen | | **„Infiniter Loop“** | | • Abarbeitung eines Befehles mit den Teilschritten  **Fetch, Decode & Execute** • Loop wird nach der Initialisierung ∞ wiederholt. | |
| Interrupts | Sprung in ISR (Siehe NVIC) Pipeline leeren: M0: 3, M0+: 2 Cycles | | |  |  |  |  | | --- | --- | --- | --- | |  | Blind Cycle | Busy-Wait | Interrupt | | Best-Case-Latency | Fixed | Unpredictable | Best | | HW-Cost | Least | Low | Moderate | | SW Complexity | Low | Low | Moderate | | CPU-Load required | High | High | Minimal | | | | |
| **Tail Chaining** Tiefer priorisierter IRQ tritt während laufendem ISR auf 🡪 Sprung in 2. ISR verkürzt (nur Vector-Fetch, kein Stacking) **Late Arrival** Kommt ein zweiter, höher priorisierter Interrupt während des Stacking vom ersten, wird nach dem Stacking der zweite Interrupt abgearbeitet **POP-Preemption** Tritt ein Interrupt während dem Unstacking ein, wird dieses unterbrochen und der Interrupt ausgeführt. | |
| **1. Software Clearing**  Pending State vor Start ISR mit Software gelöscht (z.B. durch höher priorisierte ISR) **2. Software Clearing mit Reset**  Incl. „1“; Aktive Peripherie setzt den Pending State erneut. **3. ISR Reentry** Pending State durch ISR-Handler (HW) gelöscht **4. Repulsed IRQ** Durch ersten Puls gesetzt. ISR nur einmal. Pending State erneuert mit weiteren Pulsen. **5. Pulsed IRQ** Erneuter IRQ während laufendem ISR. Pending durch zweiten Puls neu gesetzt. Anschliessend ISR nochmals ausgeführt. | |
| I/O Schnittstelle | Datenregister:  zu verarbeitende Daten  Steuerregister: Konfiguration I/O  Statusregister: Status I/O | | **Memory Protection Unit (MPU)** | | • Exception bei unerlaubtem Speicherzugriff • Privileged/User Access | |
| ADD/SUB  I\_(max{ia,ib}+1) und Q\_(max{qa,qb})  Erweitern: integer mit dem Wert vom Vorzeichen. Quotient mit "0" | | | Sättigung: wenn zwei obj ein grösseres  ergeben, dann auf ursprung grösse und max wert angeben | | | |
| Multi-Prozessor-Systeme | • Eng gekoppelt:  Gleicher Systembus,  Memory, I/O • Lose gekoppelt:   vrb über Peripherie-Bus | | **Moore’s Law** | | Die Komplexität (Transistoranzahl) bei integrierter Schaltung verdoppelt sich alle 12-24 Monate. | |
| **Reentrant Code** | | Kann mehrfach/ durch sich selber aufgerufen werden. | |
| **Register** | | 1 Register = 4 Byte e.g. 1.75KByte = 1.75 \* 210  **Anz. Register Inhalte** = Stack-Grösse [in B] / 4 | |
| NVIC (Nested Vector Interrupt Controller), ISR | Generell • 32 Ext Interrupts • 8-256 Prio Levels • Sprung in ISR 15 (M0+) /16 Cycles (M0) | | | Mögliche Interrupts • Low Random: z.B. Taster • Low Periodic: z.B. Sys-Tick (1ms) • High Random: serielle Schnittstelle • High Periodic: ADC Sequenz (>1MHz) | | |
| Funktion • Background Programm wird von PC abgearbeitet bis Interrupt (**ISR**) • Aktuelle Instruktion wird zunächst vollständig abgearbeitet und Pending  Status gesetzt. • Stacking der Register R0-R3, R12, LR, PC und PSR  • Vector-Fetch via ISR-Nummer im PSR und ISR-Adresseintrag in VT  (Sprung zu ISR) • Erste Instruktion der ISR wird geladen (Fetch) • Controller bereit, den Usercode in der ISR abzuarbeiten • Unstacking 🡪 den jeweiligen Registern wird vorheriger Wert wieder  zugewiesen & Prozessor Status wird wiedererstellt | | | | | |
| Settings • Preemption (priority)  *NVIC\_SetPriority (IRQn, Priority)* • Primask:  *1bit-Register, on/off aller Interrupts* • Faultmask:  *1bit-Register, on/off alle Interrupts & Exceptions* • Basepri (Threshold setzbar)  *8bit, on/off aller tiefen Interrupts* | | **Wake-Up** | | Modi • Sleep- & Debug-Mode  *unterbrochen durch Interrupts, Exceptions, Debug Events* • Sleep on Exit  *nach Verlassen des ISR* • Debug Support  *Infos über Nutzung* • Polling Mode  *normaler Modus, 100% CPU Auslastung* | |
| Prozessortypen | **RISC** (Reduced Instruction Set Controller) • Einfacher Instruktionssatz,  viele Register, Load/Store  **CISC** (Complex Instruction Set Controller) • Komplexer Instruktionssatz, wenig Register | **Zu beachten beim Kaufen** • Gehäusegrösse & Peripherie des μP • Bitbreite/ Grösse Adressraum • Spannung/ Leistungsaufnahme • Systemtakt • Kosten & Verfügbarkeit • Entwicklungs-SW | | | | **SPEC-Benchmarks (Marketing)** • # Befehle pro Sekunde (in MIPS) • # erforderliche Takte pro Befehl (in CPI) • # Floating-Point Berechnungen pro Sekunde (FLOPS)  • Gut: MIPS/A |
| Spezialregister | PC enthält die Adresse der nächsten Instruktion 🡪 IR übergibt nächsten Befehl an die CU 🡪 AR ermöglicht  Lesen & Schreiben von Daten 🡪 PSR (Programm Status Register) enthält u.a. die Flags  Ablauf: PC lädt Befehl in IR, Befehl wird decodiert und ins Rechenwerk geladen, wo der Befehl verarbeitet wird.  PC 🡪 Mem 🡪 IR 🡪 Decoder | |  | | | |
| RAM / ROM / Flash | **RAM**: mit CE, kombiniertem R/W, Datensig.: bidirekt. Treiber  **ROM**: mit /CE und /OE, Datensignaltreiber: TriState-Outputs  **Flash**: ähnlich wie ROM | |
| Stack | **Regeln** • Stack wird standardmässig in 8-Byte Schritten (64bit)  alloziert. • Stack muss am Ende einer Funktion wieder abgebaut  werden.  **Aufgaben bei Prozessoren** • Zwischenspeicherung von Registern, Daten,  Datenübergabe • Arbeitsregister sind begrenzt 🡪 Erweiterung mittels Stack • Parameterübergabe wird reentrant | | **System Control Space (SCS)** | | **Funktion** • Prozessorkonfigurationen (*Low Power Modes*)  • Fault Status Informationen (*Low Power Modes)* • Vektortabellen-Verlegung (*VTOR)* • Interrupt Management **Komponenten** • Processor Debug Control • MPU, NVIC, SysTick • System Control Register (VTOR, Low-Power-Control, Fault  Status, etc.) | |
| Stack- vs. RAM-Zugriff | • Beim Stack-Zugriff ist der Pointer (SP) bereits gegeben:  LDR Rx, [SP, #offset] | | • Beim RAM-Zugriff muss zuerst der Pointer geladen werden:  LDR Rx, #addr, LDR Ry, [Rx] | | | |
| SoC (Systems on Chip) | • Komplexe Systeme mit mehreren CPUs • Programm- und Datenspeicher • Vielfältige Funktionseinheiten (analog und digital) | | **SoPC** | | (System-on-Programmable-Chip) SoC einfach mittels programmierbaren HW-Bausteins | |
| μC vs. μP | Bei einem μC sind sämtliche Komponenten eines Rechnersystems auf dem gleichen Chip integriert (μP, ROM, RAM, Takt, Peripherie, Interrupt-Handling, …). **Differenzierung** • Standard-μP 🡪 allg. Einsatz • Hochleistungs-μP 🡪 Computer mit hoher Verarbeitungsleistung • μC 🡪 Embedded Systems • Digitale Signalprozessoren (DSP) 🡪 Audio/Video • Software-Prozessorkern (Soft-Core) für SoC | | **Systick  (System Tick Timer)** | | • 24Bit-Counter • Konstante Zeitbasis für Programmabläufe  Control/Reload/Value | |
| **Vektortabelle** | | Für alle Exceptions & Interrupts müssen Sprungadressen in der Vektortabelle abgelegt sein | |
| **Wake-up Interrupt Controller** | | • Hierarchische Abschaltung von Komponenten • geringerer Stromverbrauch | |
| **Word Size** | | 32Bit-μP🡪32Bit, 16Bit-μP🡪16Bit, 8Bit-μP🡪16Bit | |
| 2er-Potenzen | 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 210, …  1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, … | | **16er-Reihe** | | 1, 2, 3, 4, 5, …  16, 32, 48, 64, 80, … | |
|  |  | |  | | Testen& vergleichen: (Spezial)  CMP SUBS ohne Resultat, Flaggs  CMN ADDS ohne Resultat, Flaggs | |

