Mikrocomputertechnik – Befehlsübersicht THUMB2/Cortex-M3 – Prof. Dr. Peter Raab

Speicherzugriff

Befehl	Parameter	Funktion
LDR	R*x*, =adr	Lade Adresse des Labels „adr“ in Register R*x*
LDR(B)	R*x*, [R*y*]	Lade Inhalt des Wortes (Bytes) an der Speicherzelle mit Adresse R*y* in Register R*x*
LDR(B)	R*x*, [R*y, #const*]	Lade Inhalt des Wortes (Bytes) an der Speicherzelle mit Adresse R*y+const* in Register R*x*
LDR(B)	R*x*, [R*y*, R*z*]	Lade Inhalt des Wortes (Bytes) an der Speicherzelle mit Adresse R*y*+R*z* in Register R*x*
LDR(B)	R*x*, [R*y*, R*z*, LSL #n]	Lade Inhalt des Wortes (Bytes) an Speicherzelle mit Adresse R*y+(*R*z<<n)* in Register R*x*
LDR(B)	Rx, [Ry], #const	Lade Inhalt des Wortes (Bytes) an Speicherzelle mit Adresse *Ry* in Register *Rx* und erhöhe *Ry* um *const*
STR(B)	R*x*, [R*y*]	Speichere Inhalt des Wortes (Bytes) in Register R*x* in die Speicherzelle mit Adresse R*y*
STR(B)	R*x*, [R*y, #const*]	Speichere Inhalt des Wortes (Bytes) in Register R*x* in die Speicherzelle mit Adresse R*y+const*
STR(B)	R*x*, [R*y*, R*z*]	Speichere Inhalt des Wortes (Bytes) in Register R*x* in die Speicherzelle mit Adresse R*y*+R*z*
STR(B)	R*x*, [R*y*, R*z*, LSL #n]	Speichere Inhalt des Wortes (Bytes) in Register R*x* in die Speicherzelle mit Adresse (R*y*+R*z*<<n)
STR(B)	Rx, [Ry], #const	Speichere Inhalt des Wortes (Bytes) in Register Rx in die Speicherzelle mit Adresse *Ry* und erhöhe *Ry* um *const*
PUSH	{Rx, Ry, ...}	Speichere Inhalt der angegebenen Liste von Registern auf dem Stack
POP	{Rx, Ry, ...}	Lade Inhalt der angegebenen Liste von Registern vom Stack

Datenverarbeitung

Befehl	Parameter	Funktion
ADD	R*x*, R*y*, R*z*	R*x* := R*y* + R*z*
ADD	R*x*, #const	R*x* := R*x* + const (0 <= const <= 255)
SUB	R*x*, R*y*, R*z*	R*x* := R*y* – R*z*
SUB	R*x*, #const	R*x* := R*x* – const (0 <= const <= 255)
AND	R*x*, R*y*, R*z*	R*x* := R*y* & R*z (auch mit #const statt Rz)*
ORR	R*x*, R*y*, R*z*	R*x* := R*y* | R*z (auch mit #const statt Rz)*
EOR	R*x*, R*y*, R*z*	R*x* := R*y* ⊕ R*z (auch mit #const statt Rz)*
MOV	R*x*, R*y*	R*x* := R*y*
MOV	R*x*, #const	R*x* := const (0 <= const <= 255)
CMP	R*x*, R*y*	Berechne R*x* – R*y* und setze Flags entsprechend
CMP	R*x*, #const	Berechne R*x* – const und setze Flags entsprechend (0 <= const <= 255)
LSL	R*x*, R*y*, #const	Schiebe Inhalt von R*y* um const Bit nach links und speichere Ergebnis in R*x*: R*x* := R*x* << const (0 <= const <= 31)
LSL	R*x*, R*y*	Schiebe Inhalt von R*x* um *Ry* Bit nach links: R*x* := R*x* << R*y*
LSR	R*x*, R*y*, #const	Schiebe Inhalt von R*y* um const Bit *logisch* nach rechts und speichere Ergebnis in R*x*: R*x* := R*y* $>>_{log}$ const (0 <= const <= 31)
LSR	R*x*, R*y*	Schiebe Inhalt von R*x* um *Ry* Bit *logisch* nach rechts: R*x* := R*x* $>>_{log}$ R*y*
ASR	R*x*, R*y*, #const	Schiebe Inhalt von R*y* um const Bit *arithmetisch* nach rechts und speichere Ergebnis in R*x*: R*x* := R*y* $>>_{arith}$ const (0 <= const <= 31)
ASR	R*x*, R*y*	Schiebe Inhalt von R*x* um *Ry* Bit *arithmetisch* nach rechts: R*x* := R*x* $>>_{arith}$ R*y*
ROR	R*x*, R*y*	Rotiere Inhalt von R*x* um *Ry* Bit nach rechts

Multiplikation und Division

Befehl	Par.	Funktion
MUL	R*x*, R*y*, R*z*	R*x* := R*y* * R*z*
UDIV	R*x*, R*y*, R*z*	R*x* := R*y* / R*z* (Ganzzahl-division ohne Vorzeichen)
SDIV	R*x*, R*y*, R*z*	R*x* := R*y* / R*z* (Ganzzahl-division mit Vorzeichen)

Sprungbefehle

Befehl	Par.	Funktion
B	label	PC := Adresse von label
BX	R*x*	PC := Inhalt von R*x*
BL	label	LR := Rücksprungadresse PC := Adresse von label
B{cond}	label	Bedingter Sprung, PC := Adresse von label, wenn Bedingung „cond“ erfüllt ist (siehe unten)

Bedingungen für Sprungbefehle „B{cond}“

Bedingung „B{cond}“	Erfüllt, wenn Flags...
BEQ	Z = 1**, „gleich“**
BNE	Z = 0, **„ungleich“**
BCS / BHS	C = 1, **„>=“ (vorzeichenlos)**
BCC / BLO	C = 0, **„<“ (vorzeichenlos)**
BMI	N = 1, **„negativ“**
BPL	N = 0, **„positiv“**
BVS	V = 1, **„overflow“**
BVC	V = 0, **„kein overflow“**
BHI	C = 1 & Z = 0, **„>“ (vorz.los)**
BLS	C = 0 & Z = 1, **„<=“ (vorz.los)**
BGE	(N=1 & V=1) | (N=0 & V=0) **„>=“ (vorzeichenbehaftet)**
BLT	(N=1 & V=0) | (N=0 & V=1) **„<“ (vorzeichenbehaftet)**
BGT	**„>“ (vorzeichenbehaftet)**
BLE	**„<=“ (vorzeichenbehaftet)**

Prozessorregister

R0	
R1	
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13 = SP	Stack Pointer
R14 = LR	Link Register
R15 = PC	Programmzähler
xPSR	Prozessor Status-Register

Flags im xPSR: **N Z C V**

N: Negativ, Z: Zero,
C: Carry, V: Overflow

Hinweis: THUMB2 beinhaltet noch weitere Befehle. In dieser Übersicht sind nur die für MCT relevanten aufgeführt.

Mikrocomputertechnik – Register des NXP LPC1778 – Prof. Dr. Peter Raab

Digitale Ein-/Ausgabe-Ports GPIO (Port n=0, 1, 2)

Registername	Funktion
LPC_GPIO**n**_DIR	Richtung der Pins Bit=0: Eingang, Bit=1: Ausgang
LPC_GPIO**n**_PIN	Aktueller Wert der Pins (logisch „0“ oder „1“)
LPC_GPIO**n**_SET	Ausgangsbit setzen Bit=0: unverändert, Bit=1: Ausgang auf „1“ setzen
LPC_GPIO**n**_CLR	Ausgangsbit löschen Bit=0: unverändert, Bit=1: Ausgang auf „0“ setzen

Timer (Timer n=0,...,3)

Registername	Funktion
LPC_TIMER**n**_PR	Prescale Register: Teiler, durch den der Systemtakt geteilt wird, um den Timertakt zu erhalten
LPC_TIMER**n**_PC	Prescale Counter: Aktueller Wert des Prescalers
LPC_TIMER**n**_TCR	Timer Control Register: Aktivieren/Deaktivieren des Timers
LPC_TIMER**n**_TC	Timer Counter: Aktueller Wert des Timers
LPC_TIMER**n**_IR	Interrupt Register: Interrupt-Status abfragen/löschen
LPC_TIMER**n**_MCR	Match Control Register: Legt fest, was bei Match passiert
LPC_TIMER**n**_MR**x**	Match Register (x=0-3): Vergleichswert

Relevante Bits in Timer-Registern

TCR	Funktion
Bit 0	CEN: Counter Enable (1=aktiviert, 0=deaktiviert)
Bit 1	CRST: Counter Reset – TC und PC<=0 (1=Reset, 0=weiter zählen)

IR	Funktion
Bit 0	MR0INT: Interruptstatus für MR0 lesen: 1=Interrupt anstehend, 0=nicht anstehend schreiben: 1=Interrupt löschen, 0=keine Änderung
Bit 1	MR1INT: Interrupt für MR1
Bit 2	MR2INT: Interrupt für MR2
Bit 3	MR3INT: Interrupt für MR3

MCR	Funktion
Bit 0	MR0I: Interrupt bei Match mit MR0 auslösen 1=Interrupt auslösen, 0=nicht auslösen
Bit 1	MR0R: Timer bei Match mit MR0 auf 0 zurücksetzen 1=zurücksetzen, 0=nicht zurücksetzen
Bit 2	MR0S: Timer bei Match mit MR0 anhalten 1=Timer anhalten, 0=nicht anhalten
Bit 3	MR1I (siehe MR0I)
Bit 4	MR1R (siehe MR0R)
Bit 5	MR1S (siehe MR0S)
Bit 6	MR2I
Bit 7	MR2R
Bit 8	MR2S
Bit 9	MR3I
Bit 10	MR3R
Bit 11	MR3S

Serielle Schnittstelle (UART n=0,2)

Name	Funktion
LPC_UART**n**_RBR	Receive Buffer Register
LPC_UART**n**_THR	Transmit Hold Register
LPC_UART**n**_DLL	Baudrate Divisor Low Byte
LPC_UART**n**_DLM	Baudrate Divisor High Byte
LPC_UART**n**_IER	Interrupt Enable Register
LPC_UART**n**_FCR	FIFO Control Register
LPC_UART**n**_LCR	Line Control Register
LPC_UART**n**_LSR	Line Status Register

Relevante Bits in UART-Registern

FCR	Funktion
Bit 0	FIFOEN: FIFO Enable (1=an, 0=aus)
Bit 1	RXFIFORES: Receive FIFO Reset (1=reset)
Bit 2	TXFIFORES: Transmit FIFO Reset (1=reset)

LCR	Funktion
Bits 1:0	WLS: Word Length Select 0x0=5 Bit, 0x1=6 Bit, 0x2=7 Bit, 0x3=8 Bit
Bit 2	SBS: Stop Bit Select 0=1 Stop-Bit, 1=2 Stop-Bits
Bit 3	PE: Parity Enable 0=Parity deaktiviert, 1= Parity aktiviert
Bits 5:4	PS: Parity Select 0x0=odd, 0x1=even, 0x2=fix ‘1’, 0x3=fix ‘0’
Bit 7	DLAB: Divisor Latch Access Bit 0=kein Zugriff, 1=Zugriff auf DLL/DLM

LSR	Funktion
Bit 0	RDR: Receiver Data Ready 0=Empfangsregister leer, 1=nicht leer
Bit 5	THRE: Transmit Hold Register Empty 0=Senderegister nicht leer, 1=leer

IER	Funktion
Bit 0	RBRIE: Receive (RBR) Interrupt Enable 0=deaktiviert, 1=aktiviert
Bit 1	THRIE: Transmit (THR) Interrupt Enable 0=deaktiviert, 1=aktiviert

NVIC (Nested Vectored Interrupt Controller)

Reg.	Funktion
ISER0	Interrupt Set Enable (Bit n=Interrupt n) Interrupts anschalten
ICER0	Interrupt Clear Enable (Bit n=Interrupt n) Interrupts ausschalten

NVIC-Interrupt-Nummern

Nr.	Funktion
1	Timer 0 (Match und Capture)
2	Timer 1 (Match und Capture)
3	Timer 2 (Match und Capture)
4	Timer 3 (Match und Capture)
5	UART 0 (Receive und Transmit)
7	UART 2 (Receive und Transmit)