

# Quick Guide MOP

Detta häfte får användas under tentamen i kursen "Maskinorienterad programmering" under förutsättning att inga egna anteckningar gjorts.

Ev. rättelser bifogas tentamenstes.

Institutionen för Data och informationsteknik  
Chalmers 2024-01-23

## Innehåll:

MINNESDISPOSITION MD407 .....	2
INSTRUKTIONSOVERSKT .....	3
ADRESSERINGSSÄTT .....	3
REGISTERUPPSÄTTNING .....	4
KOMPILATORKONVENTIONER.....	4
REGISTERBESKRIVNINGAR .....	5
TILLDELNINGAR .....	6
UTTRYCKSEVALUERING .....	8
PROGRAMFLÖDESKONTROLL .....	11
SPECIELLA INSTRUKTIONER.....	12
ASSEMBLERDIREKTIV .....	13
PERIFERIBUSS .....	14
PERIFERIKRETSAR.....	17
VEKTORTABELL .....	22
C QUICK REFERENCE GUIDE .....	24

Tillhör: \_\_\_\_\_

# MINNESDISPOSITION MD407

## Periferikretsar

A000 0000 - A000 0FFF      FSMC control reg      AHB3

### Adressrum

Tillverkar-specifikt	FFFF FFFF	
	E010 0000	
	E00F FFFF	
Periferibuss		
	E000 0000	
	DFFF FFFF	
Externa enheter		
	A000 0000	
	9FFF FFFF	
Extern RAM-minne		
	6000 0000	
	5FFF FFFF	
Periferikretsar		
	4000 0000	
	3FFF FFFF	
Block 1 SRAM		
	2000 0000	
	1FFF FFFF	
Block 0 kod		
	0000 0000	

5006 0800 - 5006 0BFF	RNG	AHB1
5006 0400 - 5006 07FF	HASH	
5006 0000 - 5006 03FF	CRYP	
5005 0000 - 5005 03FF	DCMI	
5000 0000 - 5003 FFFF	USB OTG FS	
4004 0000 - 4007 FFFF	USB OTG HS	
4002 B000 - 4002 BBFF	DMA2D	
4002 9000 - 4002 93FF	ETHERNET MAC	
4002 8C00 - 4002 8FFF		
4002 8800 - 4002 8BFF		
4002 8400 - 4002 87FF		
4002 8000 - 4002 83FF		
4002 6400 - 4002 67FF	DMA2	
4002 6000 - 4002 63FF	DMA1	
4002 4000 - 4002 4FFF	BKPSRAM	
4002 3C00 - 4002 3FFF	Flash interface	
4002 3800 - 4002 3BFF	RCC	
4002 3000 - 4002 33FF	CRC	
4002 2800 - 4002 2BFF	GPIOK	
4002 2400 - 4002 27FF	GPIOJ	
4002 2000 - 4002 23FF	GPIOI	
4002 1C00 - 4002 1FFF	GPIOH	
4002 1800 - 4002 1BFF	GPIOG	
4002 1400 - 4002 17FF	GPIOF	
4002 1000 - 4002 13FF	GPIOE	
4002 0C00 - 4002 0FFF	GPIOD	
4002 0800 - 4002 0BFF	GPIOC	
4002 0400 - 4002 07FF	GPIOB	
4002 0000 - 4002 03FF	GPIOA	
4001 6800 - 4001 6BFF	LCD-TFT	APB2
4001 5800 - 4001 4BFF	SAI1	
4001 5400 - 4001 57FF	SPI6	
4001 5000 - 4001 53FF	SPI5	
4001 4800 - 4001 4BFF	TIM11	
4001 4400 - 4001 47FF	TIM10	
4001 4000 - 4001 43FF	TIM9	
4001 3C00 - 4001 3FFF	EXTI	
4001 3800 - 4001 3BFF	SYSCFG	
4001 3400 - 4001 37FF	SPI4	
4001 3000 - 4001 33FF	SPI1	
4001 2C00 - 4001 2FFF	SDIO	
4001 2000 - 4001 23FF	ADC1-ADC2-ADC3	
4001 1400 - 4001 17FF	USART6	
4001 1000 - 4001 13FF	USART1	
4001 0400 - 4001 07FF	TIM8	
4001 0000 - 4001 03FF	TIM1	
4000 7C00 - 4000 7FFF	UART8	APB1
4000 7800 - 4000 7BFF	UART7	
4000 7400 - 4000 77FF	DAC	
4000 7000 - 4000 73FF	PWR	
4000 6800 - 4000 6BFF	CAN2	
4000 6400 - 4000 67FF	CAN1	
4000 5C00 - 4000 5FFF	I2C3	
4000 5800 - 4000 5BFF	I2C2	
4000 5400 - 4000 57FF	I2C1	
4000 5000 - 4000 53FF	UART5	
4000 4C00 - 4000 4FFF	UART4	
4000 4800 - 4000 4BFF	USART3	
4000 4400 - 4000 47FF	USART2	
4000 4000 - 4000 43FF	I2S3ext	
4000 3C00 - 4000 3FFF	SPI3/I2S3	
4000 3800 - 4000 3BFF	SPI2/I2S2	
4000 3400 - 4000 37FF	I2S2ext	
4000 3000 - 4000 33FF	IWDG	
4000 2C00 - 4000 2FFF	WWDG	
4000 2800 - 4000 2BFF	RTC & BKP Reg	
4000 2000 - 4000 23FF	TIM14	
4000 1C00 - 4000 1FFF	TIM13	
4000 1800 - 4000 1BFF	TIM12	
4000 1400 - 4000 17FF	TIM7	
4000 1000 - 4000 13FF	TIM6	
4000 0C00 - 4000 0FFF	TIM5	
4000 0800 - 4000 0BFF	TIM4	
4000 0400 - 4000 07FF	TIM3	
4000 0000 - 4000 03FF	TIM2	

### Block 1 SRAM

2001 FFFF		Monitor/ debugger stack och data
2001 C400		Relokerade avbrottsvektorer
2001 C3FF		
2001 C000		Reserverat för applikationen
2001 BFFF		
2000 0000		

### Periferibuss

E000 EF44		"Floating point unit"
E000 EF30		
E000 EF03		"NVIC"
E000 EF00		
E000 EDB8		"Memory protection unit"
E000 ED90		
E000 ED8B		"Floating point unit, access control"
E000 ED88		
E000 ED3F		"System Control Block"
E000 ED00		
E000 E4EF		"NVIC"
E000 E100		
E000 E01F		"System timer"
E000 E010		

# Instruktionslista ARM Thumb

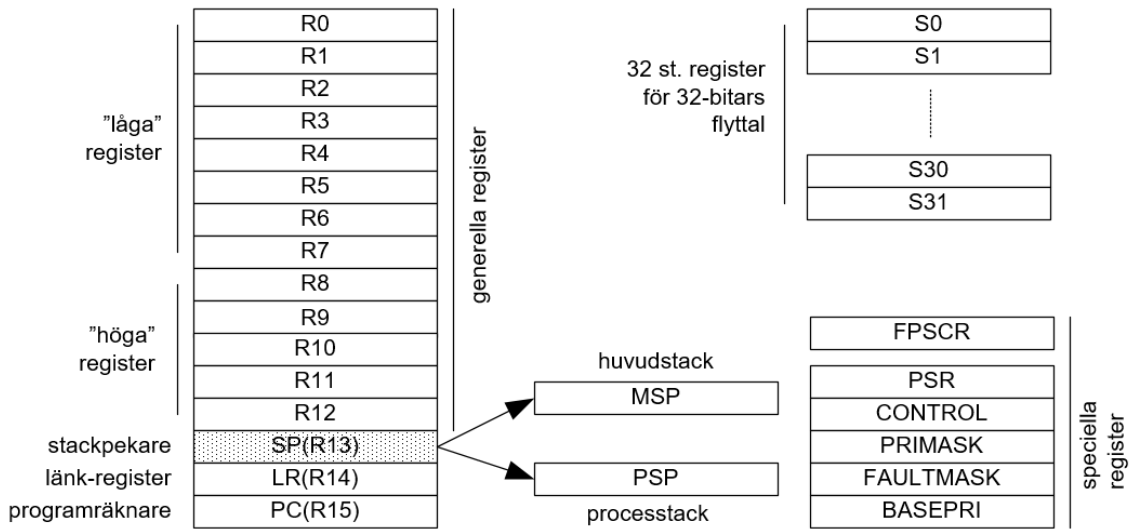
## INSTRUKTIONSOVERSIKT

Instruktion	Storlek	Cortex M0	Cortex M0+	Cortex M1	Cortex M3	Cortex M4	Cortex M7	Ark.
ADC, ADD, (ADR), AND, ASR, B, BIC, BKPT, BLX, BX, CMN, CMP, CPS, EOR, LDM, (LDMIA, LDMFD), LDR, LDRB, LDRH, LDRSB, LDRSH, LSL, LSR, MOV, MUL, MVN, NOP, ORR, POP, PUSH, REV, REV16, REVSH, ROR, RSB, SBC, SEV, STM, (STMIA, STMEA), STR, STRB, STRH, SUB, SVC, SXTB, SXTL, TST, UXTB, UXTH, WFE, WFI, YIELD	16-bit	x	x	x	x	x	x	v6
BL, DMB, DSB, ISB, MRS, MSR	32-bit	x	x	x	x	x	x	
CBNZ, CBZ, IT	16-bit				x	x	x	
ADC, ADD, AND, ASR, B, BFC, BFI, BIC, CDP, CLREX, CLZ, CMN, CMP, DBG, EOR, LDC, LDMA, LDMDB, LDR, LDRB, LDRBT, LDRD, LDREX, LDREXB, LDREXH, LDRH, LDRHT, LDRSB, LDRSBT, LDRSHT, LDRT, MCR, LSL, LSR, MLS, MCRR, MLA, MOV, MOVT, MRC, MRRC, MUL, MVN, NOP, ORN, ORR, PLD, PLDW, PLI, POP, PUSH, RBIT, REV, REV16, REVSH, ROR, RRX, RSB, SBC, SBFX, SDIV, SEV, SMLAL, SMULL, SSAT, STC, STMDB, STR, STRB, STRBT, STRD, STREX, STREXB, STREXH, STRH, STRHT, STRT, SUB, SXTB, SXTL, TBB, TBH, TEQ, TST, UBFX, UDIV, UMLAL, UMLL, USAT, UXTB, UXTH, WFE, WFI, YIELD	32-bit				x	x	x	v7
PKH, QADD, QADD16, QADD8, QASX, QDADD, QDSUB, QSAX, QSUB, QSUB16, QSUB8, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHASX, SHSAX, SHSUB16, SHSUB8, SMLABB, SMLABT, SMLATB, SMLATT, SMLAD, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD, SMLAWB, SMLAWT, SMLSD, SMLSLD, SMMLA, SMMLS, SMMUL, SMUAD, SMULBB, SMULBT, SMULTT, SMULTB, SMULWT, SMULWB, SMUSD, SSAT16, SSAX, SSUB16, SSUB8, SXTAB, SXTAB16, SXTAH, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSAX, UHSUB16, UHSUB8, UMAAL, UQADD16, UQADD8, UQASX, UQSAX, UQSUB16, UQSUB8, USAD8, USADA8, USAT16, USAX, USUB16, USUB8, UXTAB, UXTAB16, UXTAH, UXTB16	32-bit					x	x	v7e (DSP)
VABS, VADD, VCMPE, VCMPE, VCVT, VCVTR, VDIV, VLDM, VLDR, VMLA, VMLS, VMOV, VMRS, VMRS, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VPOP, VPUSH, VSQRT, VSTM, VSTR, VSUB	32-bit					SP FPU	SP FPU	32-b FP
FP- dubbel precision	32-bit						DP FPU	64-b FP

## ADRESSERINGSSÄTT

Namn	Syntax	Exempel	RTN
Register direct	Rx	MOV R0, R1	R0←R1
Direct	Symbol	LDR R0, symbol	R0←M(symbol)
Immediate	#const	MOV R0, #0x15	R0←0x15
Register indirect	[Rx]	LDR R0, [R1]	R0←M(R1)
.. with offset	[Rx, #offset]	LDR R0, [R1, #4]	R0←M(R1+4)
.. with register offset	[Rx, Ri]	LDR R0, [R1, R2]	R0←M(R1+R2)

# REGISTERUPPSÄTTNING



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N Z C V Q																				ISR_NUMBER							APSR				
																ICI/IT							ICI/IT							EPSR	

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																											F	S	P	CONTROL	

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																											M	PRIMASK			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																											M	FAULTMASK			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
																											PRIORITY							BASEPRI

# KOMPILATORKONVENTIONER

Register	Användning			
R15 (PC)	Programräknare			
R14 (LR)	Länkregister			
R13 (SP)	Stackpekare			
R12 (IP)	Dessa register är avsedda för variabler och som temporära register. Om dom används måste dom sparas och återställas av den anropade ( <i>callee</i> ) funktionen			
R11				
R10				
R9				
R8				
R7	Speciellt använder GCC R7 som pekare till aktiveringspost ( <i>stack frame</i> ) Också dessa register är avsedda för variabler och temporärbruk			
R6	Om dom används måste dom sparas och återställas av den anropade ( <i>callee</i> ) funktionen			
R5				
R4				
R3			parameter 4 / temporärregister	Dessa register sparas normalt sett inte över funktionsanrop men om, så är det den anropande ( <i>caller</i> ) funktionens uppgift
R2			parameter 3 / temporärregister	
R1	parameter 2 / resultat 2 / temporärregister			
R0	parameter 1 / resultat 1 / temporärregister			

# REGISTERBESKRIVNINGAR

R0-R12: Generella 32-bitars register.

R13: Stackpekare, i själva verket två olika register där inställningen i CONTROL-registret bestämmer vilket av registren PSP (process stack pointer) eller MSP (master stack pointer)

R14: Länkregister, i detta register sparas återhoppadressen vid BL (Branch and Link)instruktionen.

R15: Programräknaren

PSR: Program Status Register

Är i själva verket tre olika register

APSR (Application Program Status Register) innehåller statusbitar från operationer.

IPSR (Interrupt Program Status Register)

ISR\_NUMBER: Är antingen 0, dvs. inget avbrott, eller indikerar det aktiva avbrottet.

EPSR (Execution Program Status Register)

Registret innehåller biten Thumb state och exekveringstillståndet för antingen:

- *If-Then* (IT) instruktionen
- *Interruptible-Continuable Instruction* (ICI) fält för en avbruten load multiple eller store multiple instruktion.

ICI:

Då ett avbrott inträffar under exekvering av någon av instruktionerna LDM STM, PUSH, POP, VLDM, VSTM, VPUSH, eller VPOP:

- Stoppas instruktionen temporärt
- Skriver värdet för operationens nästa register i EPSR [15:12].
- Efter att ha betjänat avbrottet:
- Fullföljer instruktionen med början på det register som anges av EPSR[15:12]

Bitarna [26:25,11:10] är 0 om processorn är i ICI-tillstånd.

IT:

Ett IT-block utgörs av upp till fyra villkorligt exekverbara instruktioner. Se instruktionslistan för beskrivning at IT-instruktionen.

T: Thumb state

Cortex M4 stödjer enbart exekvering av instruktioner i Thumb-tillstånd. Följande instruktioner kan potentiellt ändra detta tillstånd eftersom de påverkar T-biten.

- Instruktionerna BLX, BX och POP{PC}
- Återställning av av xPSR vid återgång från undantag
- Bit[0] i adressen hos en vektor i avbrottstabellen

Försök att exekvera en instruktion då T är 0 resulterar i undantag (fault) eller att processorn stannar (lockup).

Läsning från EPSR med instruktionen MSR returnerar alltid 0.

Försök att skriva ignoreras. EPSR kan undersökas i en hanteringsrutin genom att EPSR extraheras från det PSR som lagrats på stacken vid undantaget/avbrottet.

CONTROL:

F=0: Flyttalsräknaren används ej

F=1: Flyttalsräknare aktiverad

SPSEL: Aktiv stackpekare

S=0: MSP är aktiv stackpekare

S=1: PSP är aktiv stackpekare

I Handler mode, läses alltid denna bit som 0. Processorn återställer automatiskt rätt bit vid återgång från undantag/avbrott.

nPRIV: Nivå då processorn är i Thread mode.

P=0: Priviligierad, alla instruktioner tillgängliga.

P=1: Icke privilegierad, en privilegierad instruktion exekveras då som NOP.

BASEPRI[7:4] Prioritetsmask för avbrott

0x00: ingen betydelse

Ett värde skilt från noll i dessa bitar anger den basprioritet som gäller. Processorn accepterar inga avbrott med ett prioritetsvärde som är större eller lika med värdet i detta register. Observera att högsta prioritet anges med prioritetsvärdet 0, lägsta prioritet anges med värdet 0xF0.

PRIMASK:

0: Ingen effekt

1: Förhindrar aktivering av undantag/avbrott med konfigurerbar prioritet, dvs *maskerar* avbrott..

FAULTMASK:

0: Ingen effekt

1: Förhindrar aktivering av alla undantag utom NMI.

## TILLDELNINGAR

### ADR – Address to register

Adderar en konstant till värdet i PC och skriver resultatet till destinationsregistret.

*Syntax:*

```
ADD    Rd, PC, #const
ADR    Rd, label
LDR    Rd, =label
```

Observera att bara de två sista accepteras av assemblern.

Rd Destination, (R0-R7).  
label Adressen till en instruktion eller data. Assemblern bestämmer ett värde för offseten så att denna är rättad till WORD-adresser  
const Positiv konstant 0-1020. Endast multiplar av 4 är tillåtna, kodas därför med 8 bitar.

### ADD – Add SP-relative address to register

Adderar en konstant till värdet i SP och skriver resultatet till ett generell register eller SP.

*Syntax:*

```
ADD    Rd, SP, #imm8
ADD    SP, SP, #imm7 eller
ADD    SP, #imm7
Rd     Destination, (R0-R7).
imm8   Positiv konstant 0-1020. Endast multiplar av 4 är tillåtna, kodas därför med 8 bitar.
imm7   Positiv konstant 0-508. Endast multiplar av 4 är tillåtna, kodas därför med 7 bitar.
```

### SUB – SP-relative address to register

Subtraherar en konstant från värdet i SP och skriver resultatet till SP.

*Syntax:*

```
SUB    SP, SP, #imm7 eller
SUB    SP, #imm7
Rd     Destination, (R0-R7).
imm7   Positiv konstant 0-508. Endast multiplar av 4 är tillåtna, kodas därför med 7 bitar.
```

### LDR – Load immediate

En basadress bestäms genom att en konstant offset adderas till innehållet i ett basregister. Därefter kopieras ett WORD, från denna adress, till destinationsregistret. Om konstanten är 0 kan den utelämnas.

*Syntax:*

```
LDR    Rt, [Rn, +/- #imm5]
LDR    Rt, [Rn]
LDR    Rt, [SP, +/- #imm8]
LDR    Rt, [SP]
Rt     Destination, (R0-R7).
Rn     Basregister för adressberäkningen, (R0-R7).
+/-    Anger positiv eller negativ offset. Om tecken utelämnas tolkas detta som positiv offset.
imm5   Positiv konstant 0-124. Endast multiplar av 4 är tillåtna, kodas därför med 5 bitar.
imm8   Positiv konstant 0-1020. Endast multiplar av 4 är tillåtna, kodas därför med 8 bitar.
```

### LDR – Load literal

En basadress bestäms genom att en konstant adderas till innehållet i PC. Därefter kopieras ett WORD, från denna adress, till destinationsregistret

*Syntax:*

```
LDR    Rt, label
LDR    Rt, [PC, imm]
Rt     Destination, (R0-R7).
label  Symbolisk adress till data som kopieras till Rt.
imm    Positiv konstant 0-1020. Endast multiplar av 4 är tillåtna, kodas därför med 8 bitar.
```

### LDR – Load register

En basadress bestäms genom att innehållet i ett indexregister adderas till innehållet i basregistret. Därefter kopieras ett WORD, från denna adress, till destinationsregistret

*Syntax:*

```
LDR    Rt, [Rn, Rm]
Rt     Destination, (R0-R7).
Rn     Basregister för adressberäkningen, (R0-R7).
Rm     Indexregister för adressberäkningen, (R0-R7).
```

### LDRB – Load byte immediate

En basadress bestäms genom att en konstant adderas till innehållet i ett basregister. Därefter kopieras en BYTE från denna adress, utvidgas med nollor till ett WORD och placeras i destinationsregistret.

*Syntax:*

```
LDRB   Rt, [Rn, +/- #imm]
Rt     Destination, (R0-R7).
Rn     Basregister för adressberäkningen, (R0-R7).
+/-    Anger positiv eller negativ offset. Om tecken utelämnas tolkas detta som positiv offset.
imm    Positiv konstant 0-31.
```

### LDRB – Load byte register

En basadress bestäms genom att innehållet i ett indexregister adderas till innehållet i basregistret. Därefter kopieras en BYTE från denna adress, utvidgas med nollor till ett WORD och placeras i destinationsregistret.

*Syntax:*

```
LDRB   Rt, [Rn, Rm]
Rt     Destination, (R0-R7).
Rn     Basregister för adressberäkningen, (R0-R7).
Rm     Indexregister för adressberäkningen, (R0-R7).
```

### LDRH – Load halfword immediate

En basadress bestäms genom att en konstant adderas till innehållet i ett basregister. Därefter kopieras ett HWORD från denna adress, utvidgas med nollor till ett WORD och placeras i destinationsregistret.

*Syntax:*

```
LDRH   Rt, [Rn, +/- #imm5]
Rt     Destination, (R0-R7).
Rn     Basregister för adressberäkningen, (R0-R7).
+/-    Anger positiv eller negativ offset. Om tecken utelämnas tolkas detta som positiv offset.
imm    Positiv konstant 0-62. Endast multiplar av 2 är tillåtna.
```

### LDRH – Load halfword register

En basadress bestäms genom att innehållet i ett indexregister adderas till innehållet i basregistret. Därefter kopieras ett HWORD från denna adress, utvidgas med nollor till ett WORD och placeras i destinationsregistret.

*Syntax:*

```
LDRH   Rt, [Rn, Rm]
Rt     Destination, (R0-R7).
Rn     Basregister för adressberäkningen, (R0-R7).
Rm     Indexregister för adressberäkningen, (R0-R7).
```

### LDRSB – Load signed byte register

En basadress bestäms genom att innehållet i ett indexregister adderas till innehållet i basregistret. Därefter kopieras en BYTE från denna adress, teckenutvidgas till ett WORD och placeras i destinationsregistret.

*Syntax:*

```
LDRSB  Rt, [Rn, Rm]
Rt     Destination, (R0-R7).
Rn     Basregister för adressberäkningen, (R0-R7).
Rm     Indexregister för adressberäkningen, (R0-R7).
```

**LDRSH – Load signed halfword register**

En basadress bestäms genom att innehållet i ett indexregister adderas till innehållet i basregistret. Därefter kopieras ett HWORD från denna adress, teckenutvidgas till ett WORD och placeras i destinationsregistret.

**Syntax:**

LDRSH Rt, [Rn, Rm]  
 Rt Destination, (R0-R7).  
 Rn Basregister för adressberäkningen, (R0-R7).  
 Rm Indexregister för adressberäkningen, (R0-R7).

**STR – Store immediate**

En basadress bestäms genom att en konstant offset adderas till innehållet i ett basregister. Därefter kopieras ett WORD, från källregistret till denna adress. Om konstanten är 0 kan den utelämnas.

**Syntax:**

STR Rt, [Rn, +/- #imm5]  
 STR Rt, [SP, +/- #imm8]  
 Rt Källa, (R0-R7).  
 Rn Basregister för adressberäkningen, (R0-R7).  
 +/- Anger positiv eller negativ offset. Om tecken utelämnas tolkas detta som positiv offset.  
 imm5 Positiv konstant 0-124. Endast multiplar av 4 är tillåtna, kodas därför med 5 bitar.  
 imm8 Positiv konstant 0-1020. Endast multiplar av 4 är tillåtna, kodas därför med 8 bitar.

**STR – Store register**

En basadress bestäms genom att innehållet i ett indexregister adderas till innehållet i basregistret. Därefter kopieras ett WORD, från källregistret till denna adress.

**Syntax:**

STR Rt, [Rn, Rm]  
 Rt Källa, (R0-R7).  
 Rn Basregister för adressberäkningen.  
 Rm Indexregister för adressberäkningen.

**STRB – Store byte register**

En basadress bestäms genom att innehållet i ett indexregister adderas till innehållet i basregistret. Därefter kopieras en BYTE från källregistret till denna adress.

**Syntax:**

STRB Rt, [Rn, Rm]  
 Rt Källa, (R0-R7).  
 Rn Basregister för adressberäkningen, (R0-R7).  
 Rm Indexregister för adressberäkningen, (R0-R7).

**STRB – Store byte immediate**

En basadress bestäms genom att en konstant adderas till innehållet i ett basregister. Därefter kopieras en BYTE från källregistret till denna adress.

**Syntax:**

STRB Rt, [Rn, +/- #imm]  
 Rt Källa, (R0-R7).  
 Rn Basregister för adressberäkningen, (R0-R7).  
 +/- Anger positiv eller negativ offset. Om tecken utelämnas tolkas detta som positiv offset.  
 imm Positiv konstant 0-31.

**STRH – Store halfword immediate**

En basadress bestäms genom att en konstant adderas till innehållet i ett basregister. Därefter kopieras ett HWORD från källregistret till denna adress.

**Syntax:**

STRH Rt, [Rn, +/- #imm]  
 Rt Källa, (R0-R7).  
 Rn Basregister för adressberäkningen, (R0-R7).  
 +/- Anger positiv eller negativ offset. Om tecken utelämnas tolkas detta som positiv offset.  
 imm Positiv konstant 0-62.

**STRH – Store halfword register**

En basadress bestäms genom att innehållet i ett indexregister adderas till innehållet i basregistret. Därefter kopieras ett HWORD från källregistret till denna adress.

**Syntax:**

STRH Rt, [Rn, Rm]  
 Rt Källa, (R0-R7).  
 Rn Basregister för adressberäkningen, (R0-R7).  
 Rm Indexregister för adressberäkningen, (R0-R7).

**LDMIA – Load multiple**

Kopierar konsekutiva minnesinnehåll från en basadress till en uppsättning register. Om registret som anger adressen inte ingår i listan av register kommer detta att uppdateras med adressen till den sist kopierade positionen +4.

**Syntax:**

LDMIA Rn!, {regs}  
 LDMIA Rn, {regs}  
 Rn Basregister för adressberäkningen, (R0-R7).  
 regs Kommaseparerad lista av ett eller flera register (R0-R7). Kopiering sker mot ökande adresser. Register med lägst index kopieras först.

**STMIA – Store multiple**

Kopierar registerinnehåll till konsekutiva minnesinnehåll från en basadress. Basregistret uppdateras med adressen till den sist kopierade positionen +4.

**Syntax:**

STMIA Rn!, {regs}  
 STMIA Rn, {regs}  
 Rn Basregister för adressberäkningen, (R0-R7).  
 regs Kommaseparerad lista av ett eller flera register (R0-R7). Kopiering sker mot minskande adresser. Register med högst index kopieras först. Rn bör inte ingå i listan eftersom detta kan skapa oförutsedda resultat.

**POP – Pop multiple**

Kopierar minnesinnehåll från stacken till en uppsättning, register. Slutligen uppdateras SP (jfr LDMIA). Om PC ingår i listan av register kommer instruktionen också att fungera som en BRANCH-instruktion. Bit 0 i adressen måste då vara 1 för att bibehålla Thumb-tillstånd, om inte, vidtar undantagshantering.

**Syntax:**

POP {regs}  
 regs Lista av ett eller flera register (R0-R7) och ev. PC. Kopiering sker mot ökande adresser. Register med lägst index kopieras först (R0,R1,R2 ... PC).

**PUSH – Push multiple**

Kopierar en uppsättning, ett eller flera, register till stacken. Slutligen uppdateras SP (jfr STMIA).

Om LR ingår i listan av register kan instruktionen också att komma att fungera som en BRANCH-instruktion. Bit 0 i adressen måste då vara 1 för att bibehålla Thumb-tillstånd, om inte, vidtar undantagshantering då adressen senare återställs till PC.

**Syntax:**

PUSH {regs}  
 regs Lista av ett eller flera register (R0-R7) och ev. LR. Kopiering sker mot minskande adresser. Register med högst index kopieras först (LR,R7,R6 osv.).

**MOV – Move immediate**

Placerar en 8 bitars konstant i destinationsregistret, övriga bitar i registret nollställs. Flaggor uppdateras.

**Syntax:**

MOV Rd, #const  
 Rd Destinationsregister (R0-R12, SP, LR, PC).  
 const Positiv konstant 0-255.

**Flaggor:**

N kopia av bit 31 hos resultatet  
 Z Ettställs om resultatet blev 0, nollställs annars  
 C,V Påverkas ej

**MOV – Move**

Kopierar innehållet i källregistret till destinationsregistret. I den första formen kan alla register användas, då sker ingen flaggpåverkan. I den andra formen kan endast låga register användas, med denna form uppdateras flaggorna.

**Syntax:****Form T1:**

MOV Rd, Rm

Rm Källregister (R0-R12, SP, LR, PC).

Rd Destinationsregister (R0-R12, SP, LR, PC).

**Form T2, uppdaterar flaggor:**

MOVS Rd, Rm

Rm Källregister 1 (R0-R7).

Rdn Källregister 2 och destinationsregister (R0-R7).

**Flaggor:**

N kopia av bit 31 hos resultatet

Z Ettställs om resultatet blev 0, nollställs annars

C,V Påverkas ej

**SXTH – Sign extend halfword**

Teckenutvidgar tal med tecken, från 16-bitar i källregistret till 32 bitar och placerar resultatet i destinationsregistret.

**Syntax:**

SXTH Rd, Rm

Rm Källregister (R0-R7).

Rd Destinationsregister (R0-R7).

**SXTB – Sign extend byte**

Teckenutvidgar tal med tecken, från 8-bitar i källregistret till 32 bitar och placerar resultatet i destinationsregistret.

**Syntax:**

SXTB Rd, Rm

Rm Källregister (R0-R7).

Rd Destinationsregister (R0-R7).

**UXTH – Unsigned extend halfword**

Teckenutvidgar tal utan tecken, från 16-bitar i källregistret till 32 bitar och placerar resultatet i destinationsregistret.

**Syntax:**

UXTH Rd, Rm

Rm Källregister (R0-R7).

Rd Destinationsregister (R0-R7).

**UXTB – Unsigned extend byte**

Teckenutvidgar tal utan tecken, från 8-bitar i källregistret till 32 bitar och placerar resultatet i destinationsregistret.

**Syntax:**

UXTB Rd, Rm

Rm Källregister (R0-R7).

Rd Destinationsregister (R0-R7).

**REV – Byte-reverse word**

Skifta endianordning genom att arrangera om bytes i källregistret till ett 32-bitars ord som placeras i destinationsregistret.

**Syntax:**

REV Rd, Rm

Rm Källregister (R0-R7).

Rd Destinationsregister (R0-R7).

**Detaljer:**

Rd&lt;31:24&gt; = Rm&lt;7:0&gt;;

Rd&lt;23:16&gt; = Rm&lt;15:8&gt;;

Rd&lt;15:8&gt; = Rm&lt;23:16&gt;;

Rd&lt;7:0&gt; = Rm&lt;31:24&gt;;

**REV16 – Byte-reverse packed halfword**

Skifta endianordning genom att arrangera om bytes i de båda halva orden i källregistret, resultatet placeras i destinationsregistret som då innehåller två endianskiftade halvord.

**Syntax:**

REV16 Rd, Rm

Rm Källregister (R0-R7).

Rd Destinationsregister (R0-R7).

**Detaljer:**

Rd&lt;31:24&gt; = Rm&lt;23:16&gt;;

Rd&lt;23:16&gt; = Rm&lt;31:24&gt;;

Rd&lt;15:8&gt; = Rm&lt;7:0&gt;;

Rd&lt;7:0&gt; = Rm&lt;15:8&gt;;

**REVSH – Byte-reverse signed halfword**

Skifta endianordning genom att arrangera om bytes i det lägre halva orden källregistret, därefter teckenutvidga till 32 bitar och placera resultatet i destinationsregistret.

**Syntax:**

REVSH Rd, Rm

Rm Källregister (R0-R7).

Rd Destinationsregister (R0-R7).

**Detaljer:**

Rd&lt;7:0&gt; = Rm&lt;15:8&gt;;

Rd&lt;31:8&gt; = SignExtend(Rm&lt;7:0&gt;);

## UTTRYCKSEVALUERING

### Aritmetik- och logik- operationer

**ADC – Add with carry**

Innehållen i källregister och destinationsregister adderas tillsammans med C-flaggan. Resultatet placeras i destinationsregistret. Flaggor uppdateras.

**Syntax:**

ADC Rd, Rm

ADC {Rd, }Rd, Rm

Rm Källregister.

Rd Destinationsregister.

**Flaggor:**

N kopia av bit 31 hos resultatet

Z Ettställs om resultatet blev 0, nollställs annars

C Carry från additionen

V Tvåkomplementspill

**ADD – Add**

Innehållen i två register adderas. Resultatet placeras i destinationsregistret. Flaggor uppdateras.

**Syntax:****Form T1:**

ADD Rd, Rn, Rm

Rn Källregister 1 (R0-R7).

Rm Källregister 2 (R0-R7).

Rd Destinationsregister (R0-R7).

**Form T2:**

ADD Rdn, Rm

Rm Källregister 1 (R0-R12, SP, LR, PC).

Rdn Källregister 2 och destinationsregister (R0-R12, SP, LR).

**Flaggor:**

N kopia av bit 31 hos resultatet

Z Ettställs om resultatet blev 0, nollställs annars

C Carry från additionen

V Tvåkomplementspill



**ADD – Add immediate**

En konstant adderas till innehållet i källregistret i två register adderas. Resultatet placeras i destinationsregistret.

**Syntax:****Form T1:**

ADD Rd, Rn, #<imm3>  
Rd Destinationsregister (R0-R7).  
Rm Källregister (R0-R7).  
imm3 Positiv konstant 0-7.

**Form T2:**

ADD Rdn, #<imm8>  
Rdn Källregister och destinationsregister (R0-R7).  
imm8 Positiv konstant 0-255.

**Flaggor;**

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C Carry från additionen  
V Tvåkomplementspill

**SBC – Subtract with carry**

Innehållet i källregistret subtraheras från innehållet i destinationsregistret tillsammans med inversen av C-flaggan (*Borrow*). Resultatet placeras i destinationsregistret.

**Syntax:**

SBC Rd, Rm  
SBC {Rd, }Rd, Rm  
Rm Källregister.  
Rd Destinationsregister.

**Flaggor;**

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C Borrow från subtraktionen  
V Tvåkomplementspill

**SUB – Subtract**

Innehållet i källregister 2 subtraheras från innehållet i källregister. Resultatet placeras i destinationsregistret.

**Syntax:**

SUB Rd, Rn, Rm  
Rn Källregister 1 (R0-R7).  
Rm Källregister 2 (R0-R7).  
Rd Destinationsregister (R0-R7).

**Flaggor;**

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C Borrow från subtraktionen  
V Tvåkomplementspill

**SUB – Subtract immediate**

En konstant subtraheras från innehållet i källregistret. Resultatet placeras i destinationsregistret.

**Syntax:****Form T1:**

SUB Rd, Rn, #<imm3>  
Rd Destinationsregister (R0-R7).  
Rm Källregister (R0-R7).  
imm3 Positiv konstant 0-7.

**Form T2:**

SUBS Rdn, #<imm8>  
Rdn Källregister och destinationsregister (R0-R7).  
imm8 Positiv konstant 0-255.

**Flaggor;**

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C Borrow från subtraktionen  
V Tvåkomplementspill

**RSB – Reverse subtract from 0 ("negate")**

Innehållet i källregistret subtraheras från 0. Resultatet placeras i destinationsregistret.

**Syntax:**

NEG Rd, Rm @ GAS syntax  
RSB {Rd, }Rm, #0  
Rm Källregister.  
Rd Destinationsregister.

**Flaggor;**

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C Borrow från subtraktionen  
V Tvåkomplementspill

**MUL – Multiplication**

Innehållen i källregister och destinationsregister multipliceras. Resultatets minst signifikanta 32 bitar placeras i destinationsregistret, resultatet blir det samma oavsett om de ingående operanderna betraktas som tal med eller utan tecken.

**Syntax:**

MUL Rd, Rm  
MUL {Rd, }Rd, Rm  
Rm Källregister.  
Rd Destinationsregister.

**Flaggor;**

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C,V Påverkas ej

**Bitoperationer****AND – Bitwise AND**

Instruktionen utför bitvis OCH mellan innehållen i källregister och destinationsregister. Resultatet placeras i destinationsregistret.

**Syntax:**

AND Rd, Rm  
AND {Rd, }Rd, Rm  
Rm Källregister.  
Rd Destinationsregister.

**Flaggor;**

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C,V Påverkas ej

**BIC – Bit clear**

Instruktionen utför bitvis OCH mellan komplementet av innehållet i källregister och destinationsregister. Resultatet placeras i destinationsregistret..

**Syntax:**

BIC Rd, Rm  
BIC {Rd, }Rd, Rm  
Rd Destinationsregister.  
Rm Källregister.

**Flaggor;**

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C,V Påverkas ej

**EOR – Bitwise Exclusive OR**

Instruktionen utför bitvis Exklusivt ELLER mellan innehållen i källregister och destinationsregister. Resultatet placeras i destinationsregistret.

**Syntax:**

EOR Rd, Rm  
EOR {Rd, }Rd, Rm  
Rm Källregister.  
Rd Destinationsregister.

**Flaggor;**

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C,V Påverkas ej

**ORR – Bitwise OR**

Instruktionen utför bitvis ELLER mellan innehållen i källregister och destinationsregister. Resultatet placeras i destinationsregistret.

*Syntax:*

ORR        Rd, Rm  
 ORR        { Rd, } Rd, Rm  
 Rm        Källregister.  
 Rd        Destinationsregister.

*Flaggor;*

N        kopia av bit 31 hos resultatet  
 Z        Ettställs om resultatet blev 0, nollställs annars  
 C,V      Påverkas ej

#### MVN – Bitwise NOT

Instruktionen bildar komplementet av innehållet i källregistret och placerar detta i destinationsregistret.

*Syntax:*

MVN       Rd, Rm  
 MVN       { Rd, } Rd, Rm  
 Rm        Källregister.  
 Rd        Destinationsregister.

*Flaggor;*

N        kopia av bit 31 hos resultatet  
 Z        Ettställs om resultatet blev 0, nollställs annars  
 C,V      Påverkas ej

## Jämförelse och test

### TST – Test

Instruktionen utför bitvis OCH mellan operander i register.

*Syntax:*

TST        Rn, Rm  
 Rn        Innehåller operand 1.  
 Rm        Innehåller operand 2.

*Flaggor;*

N        kopia av bit 31 hos resultatet  
 Z        Ettställs om resultatet blev 0, nollställs annars  
 C,V      Påverkas ej

### CMP – Compare immediate

En konstant subtraheras från innehållet i källregistret.

*Syntax:*

Form T1:

CMP       Rn, #<imm8>  
 Rn        Källregister och destinationsregister (R0-R7).  
 imm8     Positiv konstant 0-255.

*Flaggor;*

N        kopia av bit 31 hos resultatet  
 Z        Ettställs om resultatet blev 0, nollställs annars  
 C        Carry från additionen  
 V        Tvåkomplementspill

### CMP – Compare

Källoperanden subtraheras från destinationsoperanden.

*Syntax:*

CMP       Rn, Rm  
 Rn        Innehåller operand 1.  
 Rm        Innehåller operand 2.

Form T1: Rn och Rm kan vara R0-R7

Form T2: Ett av registren från R0-R7, det andra registret från R8-R14.

*Flaggor;*

N        kopia av bit 31 hos resultatet  
 Z        Ettställs om resultatet blev 0, nollställs annars  
 C        Borrow från subtraktionen  
 V        Tvåkomplementspill

### CMN – Compare negative

Källoperanden adderas till destinationsoperanden.

*Syntax:*

CMN       Rn, Rm  
 Rn        Innehåller operand 1.  
 Rm        Innehåller operand 2.

*Flaggor;*

N        kopia av bit 31 hos resultatet  
 Z        Ettställs om resultatet blev 0, nollställs annars  
 C        Carry från additionen  
 V        Tvåkomplementspill

## Skiftoperationer

### LSL – Logical shift left immediate

Logiskt vänsterskift, innehållet i källregistret skiftas och placeras i destinationsregistret. En konstant anger antal skift som ska utföras.

*Syntax:*

LSL        Rdn, Rm, #<imm5>  
 Rd        Destinationsregister (R0-R7).  
 Rm        Källregister (R0-R7).  
 imm5     Positiv konstant 0-31.

*Flaggor;*

N        kopia av bit 31 hos resultatet  
 Z        Ettställs om resultatet blev 0, nollställs annars  
 C        Carry från sista skift  
 V        Påverkas ej

### LSL – Logical shift left

Logiskt vänsterskift, innehållet i destinationsregistret skiftas och återförs till destinationsregistret. Ett register (de fem minst signifikanta bitarna i registret) anger antal skift som ska utföras.

*Syntax:*

LSL        Rdn, Rm  
 Rd        Destinationsregister (R0-R7).  
 Rm        Källregister (R0-R7) anger antal skift som ska utföras.

*Flaggor;*

N        kopia av bit 31 hos resultatet  
 Z        Ettställs om resultatet blev 0, nollställs annars  
 C        Carry från sista skift  
 V        Påverkas ej

### LSR – Logical shift right immediate

Logiskt högerskift, innehållet i källregistret skiftas och placeras i destinationsregistret. En konstant anger antal skift som ska utföras.

*Syntax:*

LSR        Rdn, Rm, #<imm5>  
 Rd        Destinationsregister (R0-R7).  
 Rm        Källregister (R0-R7).  
 imm5     Positiv konstant 0-31.

*Flaggor;*

N        kopia av bit 31 hos resultatet  
 Z        Ettställs om resultatet blev 0, nollställs annars  
 C        Carry från sista skift  
 V        Påverkas ej

## PROGRAMFLÖDESKONTROLL

B – Unconditional branch

Ovillkorlig programflödesändring, destinationsadressen placeras i PC.

Syntax:

B label  
label Destination.

B<condition> – Conditional branch

Villkorlig programflödesändring, villkor dikterat av flaggorna N,V,Z,C och kombinationer av dessa evalueras. Om villkoret är uppfyllt placeras destinationsadressen i PC, annars fortsätter exekveringen med nästa sekventiella instruktion..

Syntax:

Bcond label  
label Destination.

Detaljer:

Resultat av flaggsättning från föregående operation (Rn-Rm)

Mnemonic	Funktion	Villkor
<b>Enkla flaggtest</b>		
BCS/BHS	"Hopp" om carry	C=1
BCC/BLO	"Hopp" om ICKE carry	C=0
BEQ	"Hopp" om zero	Z=1
BNE	"Hopp" om ICKE zero	Z=0
BMI	"Hopp" om negative	N=1
BPL	"Hopp" om ICKE negative	N=0
BVS	"Hopp" om overflow	V=1
BVC	"Hopp" om ICKE overflow	V=0

Jämförelse av tal utan tecken

BHI	Villkor: Rn > Rm	C • !Z = 1
BHS/BCS	Villkor: Rn ≥ Rm	C=1
BLO/BCC	Villkor: Rn < Rm	C=0
BLS	Villkor: Rn ≤ Rm	!C + Z = 1

Jämförelse av tal med tecken

BGT	Villkor: Rn > Rm	Z + (N ⊕ V) = 0
BGE	Villkor: Rn ≥ Rm	N ⊕ V = 0
BLT	Villkor: Rn < Rm	N ⊕ V = 1
BLE	Villkor: Rn ≤ Rm	Z + (N ⊕ V) = 1

Samma tabell kan uttryckas med C-operatorer och får då följande utseende:

C-operator	Betydelse	Datotyp	Instruktion
==	Lika	signed/unsigned	BEQ
!=	Skild från	signed/unsigned	BNE
<	Mindre än	signed	BLT
		unsigned	BCC
<=	Mindre än eller lika	signed	BLE
		unsigned	BLS
>	Större än	signed	BGT
		unsigned	BHI
>=	Större än eller lika	signed	BGE
		unsigned	BCS

BL – Branch with link

Subrutinanrop, adressen till nästa sekventiellt placerade instruktion placeras i LR, den minst signifikanta biten i denna adress sätts till 1 för att indikera Thumb-mod. Destinationsadressen placeras i PC.

Syntax:

BL label  
label Destination.

LSR – Logical shift right

Logiskt högerskift, innehållet i destinationsregistret skiftas och återförs till destinationsregistret. Ett register (de fem minst signifikanta bitarna i registret) anger antal skift som ska utföras.

Syntax:

LSR Rdn, Rm  
Rd Destinationsregister (R0-R7).  
Rm Källregister (R0-R7) anger antal skift som ska utföras.

Flaggor;

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C Carry från sista skift  
V Påverkas ej

ASR – Arithmetic shift right immediate

Aritmetiskt högerskift, innehållet i källregistret skiftas och placeras i destinationsregistret. En konstant anger antal skift som ska utföras.

Syntax:

ASR Rdn, Rm, #<imm5>  
Rd Destinationsregister (R0-R7).  
Rm Källregister (R0-R7).  
imm5 Positiv konstant 0-31.

Flaggor;

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C Carry från sista skift  
V Påverkas ej

ASR – Arithmetic shift right

Aritmetiskt högerskift, innehållet i destinationsregistret skiftas och återförs till destinationsregistret. Ett register (de fem minst signifikanta bitarna i registret) anger antal skift som ska utföras.

Syntax:

ASR Rdn, Rm  
Rd Destinationsregister (R0-R7).  
Rm Källregister (R0-R7) anger antal skift som ska utföras.

Flaggor;

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C Carry från sista skift  
V Påverkas ej

ROR – Rotate right

Rotation höger, innehållet i destinationsregistret skiftas och återförs till destinationsregistret. Ett register (de fem minst signifikanta bitarna i registret) anger antal skift som ska utföras. Carryflaggan ingår i skiftet, en utskiftad bit hamnar i Carry, inskiftad bit hämtas från Carry.

Syntax:

ROR Rdn, Rm  
Rd Destinationsregister (R0-R7).  
Rm Källregister (R0-R7) anger antal skift som ska utföras.

Flaggor;

N kopia av bit 31 hos resultatet  
Z Ettställs om resultatet blev 0, nollställs annars  
C Carry från sista skift  
V Påverkas ej

**EX – Branch with exchange**

Destinationsadressen, som finns i ett register, placeras i PC. Instruktionen kan användas för att skifta exekveringstillstånd (Thumb/ARM) hos procesorer som stödjer båda instruktionsuppsättningarna.

**Syntax:**

EX                    Rm  
Rm                    Källregister (R0-R14) innehåller destinationsadressen och den minst signifikanta biten anger hur instruktion på adressen ska tolkas, Thumb eller ARM..

**BLX – Branch with link and exchange**

Subrutinanrop, adressen till nästa sekventiellt placerade instruktion placeras i LR, den minst signifikanta biten i denna adress sätts till 1 för att indikera Thumb-mod. Destinationsadressen, som finns i ett register, placeras i PC. Instruktionen kan användas för att skifta exekveringstillstånd (Thumb/ARM) hos procesorer som stödjer båda instruktionsuppsättningarna.

**Syntax:**

BLX                    Rm  
Rm                    Källregister (R0-R14) innehåller destinationsadressen och den minst signifikanta biten anger hur instruktion på adressen ska tolkas, Thumb eller ARM.

## SPECIELLA INSTRUKTIONER

Se även tillverkarens beskrivning för ytterligare detaljer om dessa instruktioner.

**MSR – Move to special register**

Kopiera från ett generellt register till något speciellt register.

**Syntax:**

MSR                    sreg, Rn  
Rn                    Källa, generellt register.  
sreg                    Destination, kan vara något av:  
APSR, IAPSR, EAPSR, XPSR, IPSR, EPSR,  
IEPSR, MSP, PSP, PRIMASK, CONTROL

**Detaljer:**

Vissa register kan bara skrivas i Privileged mode. IPSR och EPSR är endast läsbara.

**MRS – Move from special register**

Kopiera från ett generellt register till något speciellt register.

**Syntax:**

MRS                    Rd, sreg  
Rd                    Destination, generellt register.  
sreg                    Källa, kan vara något av:  
APSR, IAPSR, EAPSR, XPSR, IPSR, EPSR,  
IEPSR, MSP, PSP, PRIMASK, CONTROL

**Detaljer:**

Vissa register kan bara läsas i Privileged mode. Försök att läsa stackpekare, IPSR eller EPSR returnerar 0.

**CPS – Change processor state**

Instruktionen används för att nollställa/sätta PRIMASK.

**Syntax:**

CPSIE                i @ Interrupt enable  
CPSID                i @ Interrupt disable

**Detaljer:**

Instruktionen kan bara utföras i i Privileged mode.

CPSIE kan användas som alternativ till MSR.

CPSIE                i är samma sak som att sätta PRIMASK=0  
CPSID                i är samma sak som att sätta PRIMASK =1

**BKPT – Breakpoint**

Instruktionen orsakar undantagshantering.

**Syntax:**

BKPT                #imm8  
imm8                En 8-bitars konstant, ignoreras av processorn men kan användas av en debugger för att ange ytterligare information om brytpunkten.

**SVC – Supervisor call**

Instruktionen orsakar undantagshantering och är speciellt avsedd att implementera övergång till privilegierat tillstånd och kontrollerad anrop till en realtidskärna eller ett operativsystem.

**Syntax:**

SVC                    #imm8  
                         .hword    0cDFxx | imm8  
imm8                En 8-bitars konstant, ignoreras av processorn men kan användas i ett operativsystem för att identifiera typ av anrop.

**NOP – No operation**

Instruktionen utför ingenting.

**Syntax:**

NOP

**YIELD – Yield hint**

Detta är en så kallad "hint"-instruktion. Används i tidsdelningssystem för att indikera att programmet för tillfället kan lämna ifrån sig processorn.

**Syntax:**

YIELD

**WFE – Wait for event hint**

Detta är en så kallad "hint"-instruktion. Används i tidsdelningssystem för synkronisering. Se även instruktionen SEV .

**Syntax:**

WFE

**WFI – Wait for interrupt hint**

Instruktionsexekveringen avbryts och processorn inväntar avbrott.

**Syntax:**

WFI

**SEV – Send event hint**

Används i tidsdelningssystem för att signalera en händelse. Se även instruktionen WFE.

**Syntax:**

SEV

**DSB – Data synchronization barrier**

Instruktionen används för att motverka icke önskade effekter av processorns pipeline. Ingen ny instruktion hämtas/utförs innan alla pågående minnesåtkomster är klara.

**Syntax:**

DSB

**DMB – Data memory barrier**

Instruktionen används för att motverka icke önskade effekter av processorns pipeline. Ingen ny LOAD- eller STORE- instruktion utförs innan alla pågående minnesåtkomster är klara.

**Syntax:**

DMB

**ISB – Instruction synchronization barrier**

Instruktionen tömmer processorns pipeline så att nästa instruktion, den omedelbart efter ISB, hämtas från cache eller minne.

**Syntax:**

ISB

**UDF – Permanently undefined**

Instruktionen orsakar undantagshantering.

**Syntax:**

UDF                    #imm8  
imm8                En 8-bitars konstant, ignoreras av processorn men kan användas av en debugger för att ange ytterligare information.

## ASSEMBLERDIREKTIV

Direktiv	Förklaring
L: innebär att en etikett kan, men behöver inte nödvändigtvis, finnas på raden.	
L: .SPACE N	Avsätter N bytes i följd i minnet (initialvärde 0).
L: .BYTE N1,N2..	Avsätter i följd i minnet en byte för varje argument. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
L: .HWORD N1,N2..	Avsätter i följd i minnet ett 16 bitars ord för varje argument. Respektive ord ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
L: .WORD N1,N2..	Avsätter i följd i minnet ett 32 bitars ord för varje argument. Respektive ord ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
L: .QUAD N1,N2..	Avsätter i följd i minnet ett 64 bitars ord för varje argument. Respektive ord ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
.ALIGN	Garanterar att påföljande adress är jämnt delbar med 4 ("word aligned")
L: .ASCII "..."	Avsätter en textsträng i minnet.
L: .FLOAT F1,F2,..	Avsätter en flyttalskonstant (32 bitar) för varje argument, i minnet.
L: .DOUBLE F1,F2,..	Avsätter en flyttalskonstant (64 bitar) för varje argument, i minnet.
L: .ORG N	Följande kod, data placeras på offset N, från början av aktuell sektion.
.EQU sym, val	Definiera symbolen sym med värdet val.

# PERIFERIBUSS

## SysTick

### System Timer (0xE000E010)

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register	
0																																		STK_CTRL
4																																		STK_LOAD
8																																		STK_VAL
0xC	r	r																															STK_CALIB	

#### STK\_CTRL Status och styrregister

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register	
0																																		STK_CTRL

#### Bit 16: (COUNTFLAG):

Biten är 1 om räknaren räknat ned till 0. Biten nollställs då registret läses.

#### Bit 2: (CLKSOURCE) biten är 1 efter RESET:

- 0: Systemklocka/8
- 1: Systemklocka

#### Bit 1: (TICKINT): Aktivera avbrott

- 0: Inget avbrott genereras.
- 1: Då räknaren slår om till 0 genereras SysTick avbrott.

#### Bit 0: (ENABLE): Aktivera räknare

Aktiverar räknaren. Då ENABLE-biten sätts till 1 kopierar räknaren värdet från STK\_LOAD till STK\_VAL och räknar ned. Då STK\_VAL slår om till 0, sätts COUNTFLAG till 1, avbrott genereras beroende på TICKINT. Därefter kopieras värdet från STK\_LOAD igen, hela processen repeteras.

- 0: Räknare inaktiv.
- 1: Räknare aktiv.

#### STK\_LOAD Räknarintervall

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register	
4																																		STK_LOAD

#### Bits 23:0 Värde för räknarintervall

Registret håller räknarintervalllets startvärde. Värdet kan vara i intervallet 0x00000001-0x00FFFFFF. Startvärdet 0 är möjligt men meningslöst eftersom slutet av räknarintervallet detekteras av att räknaren slår om från 1 till 0. För att generera  $N$  cykler fördröjning ska därför värdet i STK\_VAL sättas till  $N-1$ .

#### STK\_VAL Räknarvärde

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register	
8																																		STK_VAL

#### Bits 23:0 Aktuellt räknarvärde

Läsning av registret returnerar räknarens aktuella värde. En skrivning till registret, oavsett värde, nollställer registret såväl som COUNTFLAG i statusregistret.

#### STK\_CALIB Kalibreringsregister

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register	
0xC	r	r																																STK_CALIB

Registret är avsett för kalibreringsändamål och innehåller implementeringsspecifika detaljer. Vi behandlar inte dessa här.

## NVIC

### Nested vectored interrupt controller (0xE000E100)

För varje avbrott kontrollerat av NVIC finns följande funktioner:

- *Interrupt Set Enable Registers*, NVIC\_ISERx
- *Interrupt Clear Enable Registers*, NVIC\_ICERx
- *Interrupt Set Pending Registers*, NVIC\_ISPRx
- *Interrupt Clear Pending Registers*, NVIC\_ICPRx
- *Interrupt Active Bit Registers*, NVIC\_IABRx
- *Interrupt Priority Registers*, NVIC\_IPRx
- *Software Interrupt Trigger Register*, NVIC\_STIR

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic							
0x000																																		SETENA[31:0]	NVIC_ISER0					
0x004																																		SETENA[63:32]	NVIC_ISER1					
0x008																																		Reserverat	SETENA[81:64]	NVIC_ISER2				
0x080																																			CLRENA[31:0]	NVIC_CER0				
0x084																																			CLRENA[63:32]	NVIC_CER1				
0x088																																			Reserverat	CLRENA[81:64]	NVIC_CER2			
0x100																																			SETPEND[31:0]	NVIC_ISPR0				
0x104																																			SETPEND[63:32]	NVIC_ISPR1				
0x108																																			Reserverat	SETPEND[81:64]	NVIC_ISPR2			
0x180																																				CLRPEND[31:0]	NVIC_ICPR0			
0x184																																				CLRPEND[63:32]	NVIC_ICPR1			
0x188																																			Reserverat	CLRPEND[81:64]	NVIC_ICPR2			
0x200																																				ACTIVE[31:0]	NVIC_IABR0			
0x204																																				ACTIVE[63:32]	NVIC_IABR1			
0x208																																			Reserverat	ACTIVE[81:64]	NVIC_IABR2			
0x300																																				IP[3]	IP[2]	IP[1]	IP[0]	NVIC_IPR0
...																																			...	...	...	...	...	
0x320																																				Reserverat	IP[81]	IP[80]	NVIC_IPR20	

#### NVIC\_ISERx: SETENA[81..0] Interrupt set-enable bit:

Skrivning: 0: ingen effekt, 1: möjliggör avbrott  
Läsning: 0: avbrott avstängt, 1: avbrott möjligt

#### NVIC\_ICERx: CLRENA[81..0] Interrupt clear-enable bit:

Skrivning: 0: ingen effekt, 1: omöjliggör avbrott  
Läsning: 0: avbrott avstängt, 1: avbrott möjligt

#### NVIC\_ISEPRx: SETPEND[81..0] Interrupt set pending bit:

Skrivning: 0: ingen effekt, 1: ändrar avbrottstatus till "avvaktande" (pending).  
Läsning: 0: avbrottstatus är inte "avvaktande", 1: avbrottstatus är "avvaktande"

#### NVIC\_ICPRx: CLRPEND[81..0] Interrupt clear pending bit:

Skrivning: 0: ingen effekt, 1: avlägsnar avbrottstatus "avvaktande" (pending)  
Läsning: 0: avbrottstatus är inte "avvaktande" 1: avbrottstatus är "avvaktande"

#### NVIC\_IABRx: ACTIVE[81..0] Interrupt active bit:

Läsning: 0: avbrottstatus är inte "aktiv", 1: avbrottstatus är "aktiv"

#### NVIC\_IPRx: Interrupt priority

Varje prioritetsfält kan ha ett prioritetsvärde, 0-255. Ju lägre värde, desto högre prioritet för motsvarande avbrott. Processorn implementerar bara bitar [7: 4] för varje fält, bitar [3: 0] läses som noll och ignoreras vid skrivning.







# PERIFERIKRETSAR

## USART

Universal synchronous asynchronous receiver transmitter

USART1: 0x40011000

USART2: 0x40004400

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0																																	USART_SR
4																																	USART_DR
8																																	USART_BRR
0xC																																	USART_CR1
0x10																																	USART_CR2
0x14																																	USART_CR3
0x18																																	USART_GTPR

### USART\_SR Statusregister (0x00C0)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0							CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE	USART_SR

Bit 9: CTS: Clear To Send

Denna bit sätts av hårdvara om ETSC=1 när nivån på NCTS ingången ändras. Den återställs av programvara (genom att skriva 0 till biten). Ett avbrott genereras om CTSIE = 1 i registret USART\_CR3.

0: Ingen förändring har skett på NCTS

1: En förändring har skett på NCTS

Bit 8: LBD: Lin Break Detected

Denna bit sätts av hårdvara när LIN avbrott detekteras. Det återställs av programvara (genom att skriva 0 till biten). Ett avbrott genereras om LBDIE = 1 i registret USART\_CR2.

0: LIN avbrott ej upptäckt

1: LIN avbrott upptäckt

Bit 7: TXE: Transmit data register empty

Denna bit sätts av hårdvara när innehållet av TDR registret har överförts till skiftregistret. Ett avbrott genereras om TXEIE bit = 1 i registret USART\_CR1. Det återställs vid skrivning till USART\_DR registret.

0: Dataregistrets sändardel är upptaget med en överföring.

1: Dataregistrets sändardel är klar att användas. Bit 6 TC:

Transmission Complete

Denna bit sätts då sändningen av en ram är komplett och om TXE = 1. Ett avbrott genereras om TCIE = 1 i USART\_CR1 registreret. TC nollställs av en läsning från USART\_SR följt av en skrivning till USART\_DR eller genom att "0" skrivs till biten.

0: Överföring pågår

1: Överföringen är klar

Bit 5 RXNE: Receive data register not empty

Denna bit sätts då innehållet i skiftregister RDR har överförts till USART\_DR, dvs. ett nytt tecken har kommit. Ett avbrott genereras om RXNEIE = 1 i USART\_CR1. Biten nollställs igen vid en läsning från USART\_DR. Biten kan också återställas genom att skriva en nolla till den.

0: Inget nytt innehåll i USART\_DR sedan senaste läsningen

1: Nytt innehåll finns i USART\_DR. Bit 4 IDLE: Idle line detected

Denna bit sätts av hårdvaran när en tom ram, indikerandes att serieledningen är ledig, upptäcks. Ett avbrott genereras om IDLEIE = 1 i USART\_CR1. Det återställs av en läsning från USART\_SR direkt följt av en läsning från USART\_DR.

0: Ingen tom ram har detekterats

1: En tom ram har detekterats

Bit 3 ORE: Overrun Error

Denna bit sätts av hårdvaran om ett nytt tecken anländer samtidigt som det finns ett oläst tecken i dataregistret ("overrun error"). Ett avbrott genereras om RXNEIE = 1 i USART\_CR1. Det återställs av en läsning från USART\_SR följt av en läsning från USART\_DR.

0: Inget förlorat tecken

1: Mottaget tecken är överskrivet (förlorat) Bit 2 NF: Noise detection Flag

Denna bit sätts av hårdvara när störningar i form av brus upptäcks i en mottagen ram. Biten återställs av en läsning från USART\_SR följt av en läsning från USART\_DR.

0: Ingen störning detekterad

1: Störning detekterad

Bit 1 FE: Framing Error

Denna bit sätts av hårdvara när ett ramfel, oftast orsakat av förlorad synkronisering, upptäcks. Biten återställs av en läsning från USART\_SR följt av en läsning från USART\_DR.

0: Inget ramfel upptäcks

1: Ramfel eller BREAK-ram detekterad

Bit 0 PE: Parity Error

Denna bit sätts av hårdvara när ett paritetsfel uppträder hos mottagaren. Biten återställs av en läsning från USART\_SR följt av en läsning från USART\_DR. Programmet måste vänta på att RXNE-biten ettställs innan PE-biten återställs. Ett avbrott genereras om PEIE = 1 i USART\_CR1.

0: Inget paritetsfel

1: Paritetsfel

### USART\_DR Dataregister

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
4							R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	RDR
							T9	T8	T7	T6	T5	T4	T3	T2	T1	T0	TDR

Bits 8: 0 DR [8: 0]

Innehåller tecknet som tas emot eller sänds. USART\_DR har alltså en dubbel funktion (läs och skriv) eftersom det består av två olika fysiska register, ett för sändning (TDR) och en för mottagning (RDR). Vid sändning med paritet aktiverat (PCE bit satt till 1 i register USART\_CR1), är den mest signifikanta biten betydelselös eftersom den ersätts av pariteten för ordet. Vid mottagning med paritet aktiverat är det värde som avlästs i den mest signifikanta biten den mottagna pariteten.

### USART\_BRR Baudrate register (0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
8																	DIV_Mantissal(11:0)
																	DIV_Fraction(3:0)

Baudraten för standard och SPI-mod bestäms av:

Baudrate = fCK / (8 × (2 - OVER8) × USARTDIV)

där fCK=168/2=84 MHz

OVER8, (översampling) bestäms av en bit i USART\_CR1

USARTDIV består av DIV\_Mantissa och DIV\_Fraction

Bits 15:4 DIV\_Mantissa [11: 0]: heltalsdelen av USARTDIV

Bits 3:0 DIV\_Fraction [3: 0]: decimaldelen av USARTDIV, då

OVER8=1 ska DIV\_Fraction [3] sättas till 0.

### USART\_CR1 Control register 1 (0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0xC	OVER8	Res.	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RRM	SBK	USART_CR1

Bit 15 OVER8: Oversampling mode

0: översampling med 16

1: översampling med 8

Not: Översampling med 8 kan inte användas i Smartcard-, IrDA- eller LIN- mod

Bit 14 Reserverad

Bit 13 UE: USART enable

Då denna bit är 0, nollställs USART prescalers och all utmatning stoppas för att reducera strömförbrukning.

0: USART deaktiverad

1: USART aktiverad

Bit 12 M: Word length

Biten bestämmer ordlängden.

0: 1 startbit, 8 databitar, n stoppbit

1: 1 start bit, 9 databitar, n stoppbit

Bit 11 WAKE: Wakeup method

Biten bestämmer uppväckningsmetoden.

0: Idle Line

1: Address Mark

Bit 10 PCE: Parity Control Enable

Biten bestämmer om paritetsbit ska detekteras och kontrolleras.

När paritetsbit används sätts den alltid in som MSB.

0: Ingen paritetsbit

1: Paritetsbit används

Bit 9 PS: Parity Selection

Med biten väljs typen av paritet.

0: Jämn paritet

1: Udda paritet

Bit 8 PEIE: PE Interrupt Enable

Biten sätts och nollställs av programvara.

0: Ingen funktion

1: USART avbrott genereras då PE=1 i USART\_SR, dvs. paritetsfel.

Bit 7 TXEIE: TXE Interrupt Enable

Biten sätts och nollställs av programvara.

0: Ingen funktion

1: USART avbrott genereras då TXE=1 i USART\_SR.

Bit 6 TCIE: TC Interrupt Enable

Biten sätts och nollställs av programvara.

0: Ingen funktion

1: USART avbrott genereras då TC =1 i USART\_SR

Bit 5 RXNEIE: RXNE Interrupt Enable

Biten sätts och nollställs av programvara.

0: Ingen funktion

1: USART avbrott genereras då ORE=1 eller RXNE =1 i USART\_SR

Bit 4 IDLEIE: IDLE Interrupt Enable

Biten sätts och nollställs av programvara.

0: Ingen funktion

1: USART avbrott genereras då IDLE =1 i USART\_SR

Bit 3 TE: Transmitter Enable

Biten sätts och nollställs av programvara.

0: Sändare deaktiverad

1: Sändare aktiverad

Bit 2 RE: Receiver Enable

Biten sätts och nollställs av programvara.

0: Mottagaren är deaktiverad

1: Mottagaren aktiverad

Bit 1 RWU: Receiver wakeup

Biten bestämmer om mottagaren är i mute-mod eller inte. or not.

Biten sätts och nollställs av programvara och kan dessutom nollställas av hårdvaran då en wakeup sekvens uppträder hos mottagaren.

0: Mottagaren aktiv

1: Mottagare i mute-mod

Bit 0 SBK: Send break

Denna bit används för att sända BREAK. Biten sätts av programvara och återställs av hårdvaran då hela tecknet skickats.

0: Passiv

1: Skicka BREAK-tecken

#### USART\_CR2 Control register 2 (0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x10	RES	LINEN	STOP1[0]	CLKEN	CPOL	CPHA	LBCL	RES	LBD[1:0]	RES	ADD[0:0]						USART_CR2

Bit 15 Reserverad

Bit 14 LINEN: LIN mode enable

Biten sätts och nollställs av programvara.

0: LIN mode deaktiverad

1: LIN mode aktiverad

Med LIN-mod aktiverad kan USART skicka LIN Synch Breaks (13 låga bitar) via SBK-biten i USART\_CR1, och dessutom detektera LIN Sync breaks på bussen.

Bit 13:12 STOP: STOP bits

Dessa bitar används för att programmera antalet stoppbitar i protokollet.

00: 1 Stop bit

01: 0.5 Stop bit

10: 2 Stop bits

11: 1.5 Stop bit

Not: 0.5 stoppbitt och 1.5 stoppbitt kan inte användas med UART4 och UART5.

Bit 11 CLKEN: Clock enable

0: SCLK deaktiverad

1: SCLK aktiverad

Not: Kan inte användas med UART4 och UART5

Bit 10 CPOL: Clock polarity

Biten bestämmer polariteten hos klockutgången SCLK i synkron mod. I kombination med CPHA-biten bestämmer denna bit relationen klocka/data hos signalen

0: SCLK är låg utanför sändarfönstret.

1: SCLK är hög utanför sändarfönstret.

Not: Kan inte användas med UART4 och UART5

Bit 9 CPHA: Clock phase

Biten bestämmer fasen hos klockutgången SCLK i synkron mod. I kombination med CPOL-biten bestämmer denna bit relationen klocka/data hos signalen

0: Första klocktransition läser data

1: Andra klocktransition läser data

Not: Kan inte användas med UART4 och UART5

Bit 8 LBCL: Last bit clock pulse

Denna bit avgör om även den sista databiten ska ha en associerad klockpuls.

0: Sista databiten har ingen klockpuls hos SCLK

1: Sista databiten har klockpuls hos SCLK

Not 1: Sista biten är den 8:e eller 9:e biten beroende på M i USART\_CR.

Not 2: Kan inte användas med UART4 och UART5

Bit 7 Reserverad

Bit 6 LBDIE: LIN break detection interrupt enable

0: Inget avbrott genereras då LBD=1 i USART\_SR

1: Avbrott genereras då LBD=1 i USART\_SR

Bit 5 LBDL: LIN break detection length

0: 10-bit break detektering

1: 11-bit break detektering

Bit 4 Reserverad

Bits 3:0 ADD[3:0]: Address of the USART node

Bitfältet ger adressen till USART-noden i multiprocessor-applikationer under tyst mod.

Not: Bitarna CPOL, CPHA och LBCL får inträndras då sändaren är aktiverad.

#### USART\_CR3 Control register 3 (0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x14	RES	ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMA[R]	SCEN	NACK	HDSSEL	IRLP	IREN	ERE				USART_CR3

Bit 11 ONEBIT: One sample bit method enable

Biten avgör avkänningsmetod. NF-biten deaktiveras alltid då 1 avkänning används..

0: Bitnivå bestäms med 3 avkänningar

1: Bitnivå bestäms med 1 avkänning

Bit 10 CTSIE: CTS interrupt enable

0: Avbrott deaktiverat

1: Avbrott genereras då CTS=1 i USART\_SR

Not: Kan inte användas med UART4 och UART5

Bit 9 CTSE: CTS enable

0: CTS handskakning deaktiverad

1: CTS handskakning aktiverad

Not: Kan inte användas med UART4 och UART5

Bit 8 RTSE: RTS enable

0: RTS handskakning deaktiverad

1: RTS handskakning aktiverad

Not: Kan inte användas med UART4 och UART5

Bit 7 DMAT: DMA enable transmitter

0: DMA mode deaktiveras för sändning.

1: DMA mode aktiveras för sändning.

Bit 6 DMAR: DMA enable receiver

0: DMA mode deaktiveras för mottagning

1: DMA mode aktiveras för mottagning

Bit 5 SCEN: Smartcard mode enable

0: Smartcard mode deaktiverad

1: Smartcard mode aktiverad

Not: Kan inte användas med UART4 och UART5

Bit 4 NACK: Smartcard NACK enable

Biten sätts och nollställs av programvara

0: NACK sändning vid paritetsfel deaktiverad

1: NACK sändning vid paritetsfel aktiverad

Not: Kan inte användas med UART4 och UART5

Bit 3 HDSEL: Half-duplex selection

Aktivering av single-wire half-duplex mod

0: Half duplex mod är deaktiverad

1: Half duplex mod är aktiverad

Bit 2 IRLP: IrDA low-power

Biten väljer normal eller lågenergi IrDA mod

0: Normal mod

1: Lågenergi mod

Bit 1 IREN: IrDA mode enable

Biten aktiverar IrDA mod

0: IrDA deaktiverad

1: IrDA aktiverad

Bit 0 EIE: Error interrupt enable

Avbrottsmask för hela USART-kretsen.

0: Inga avbrott genereras

1: Avbrott genereras då DMAR=1 i USART\_CR3 och FE=1 eller

ORE=1 eller NF=1 i USART\_SR.

#### USART\_GTPR Guard time and prescaler register (0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x18	GT[7:0]							PSC[7:0]									USART_GTPR

Bits 15:8 GT[7:0]: Guard time value

Bitfältet anger Guard time uttryckt i baudrate. Används endast i smartcard mode. Transmission Complete sätts efter denna tid

Not: Kan inte användas med UART4 och UART5

Bits 7:0 PSC[7:0]: Prescaler value

– I lågenergi IrDA mod:

PSC[7:0] = IrDA Low-Power Baud Rate

Används för att dividera systemets klockfrekvens för att få

lågenergifrekvensen (prescaler).

00000000: Reserved - do not program this value

00000001: divides the source clock by 1

00000010: divides the source clock by 2

...

– I normal IrDA mod: PSC ska sättas till 00000001.

– I smartcard mode:

PSC[4:0]: Prescaler value

Används för att dividera systemets klockfrekvens för att få

smartcardfrekvensen (prescaler).

Registrets värde(5 signifikanta bitar) multipliceras med 2 för att ge dividenden.

00000: Reserverat – använd inte detta värde

00001: dividerar klockan med 2

00010: dividerar klockan med 4

00011: dividerar klockan med 6

...

Not 1: Bits [7:5] har ingen effekt i smartcard mode.

Not: Kan inte användas med UART4 och UART5

## SYSCFG

### System Configuration 0x40013800

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0																																	SYSCFG_MEMRMP
4																																	SYSCFG_PMC
8																																	SYSCFG_EXTICR1
0xC																																	SYSCFG_EXTICR2
0x10																																	SYSCFG_EXTICR3
0x14																																	SYSCFG_EXTICR4
0x20																																	SYSCFG_CMPCR

#### SYSCFG\_MEMRMP Memory remap register

Registret används för att konfigurera adressutrymmet från adress 0 och uppåt. Normalt sett används de båda BOOT-pinnarna på kretsen för detta men med detta register kan det även göras i mjukvara. För MD407 gäller att FLASH-minnet, med startadress 0x08000000 dubbelavbildas från adress 0 och uppåt.

#### SYSCFG\_PMC Peripheral mode configuration register

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Register
4									R_SEL							ADCxDC2	SYSCFG_PMC
									rw						rw	rw	rw

Används för att konfigurera de fysiska egenskaperna hos Ethernet-gränssnittet (R\_SEL) och välja funktion hos AD-omvandlare.

#### SYSCFG\_EXTICR1 External interrupt configuration register 1

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
8	EXTI3[3:0]			EXTI2[3:0]			EXTI1[3:0]			EXTI0[3:0]			SYSCFG_EXTICR1				
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

#### SYSCFG\_EXTICR2 External interrupt configuration register 2

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0xC	EXTI7[3:0]			EXTI6[3:0]			EXTI5[3:0]			EXTI4[3:0]			SYSCFG_EXTICR2				
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

#### SYSCFG\_EXTICR3 External interrupt configuration register 3

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x10	EXTI11[3:0]			EXTI10[3:0]			EXTI9[3:0]			EXTI8[3:0]			SYSCFG_EXTICR3				
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

#### SYSCFG\_EXTICR4 External interrupt configuration register 4

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x14	EXTI15[3:0]			EXTI14[3:0]			EXTI13[3:0]			EXTI12[3:0]			SYSCFG_EXTICR4				
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 15:0 EXTIx[3:0]:

Dessa bitar (EXTICRx) bestämmer hur en IO-pinne dirigeras till någon av de 16 avbrottslinorna EXTI0..EXTI15 genom att motsvarande fält skrivs med fyra bitar. Undantag: PK[15:8] används ej.

0000: PA[x]	0001: PB[x]	0010: PC[x]	0011: PD[x]	0100: PE[x]	0101: PF[x]
0110: PG[x]	0111: PH[x]	1000: PI[x]	1001: PJ[x]	1010: PK[x]	

#### SYSCFG\_CMPCR Compensation cell control register 4

Används normalt inte men kan aktiveras för att öka stigtiden vid IO-frekvenser över 50MHz.

**EXTI****External Interrupt 0x40013C00**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register	
0																																		EXTI_IMR
4																																		EXTI_EMR
8																																		EXTI_RTSTR
0xC																																		EXTI_FTSR
0x10																																		EXTI_SWIER
0x14																																		EXTI_PR

**EXTI\_IMR Interrupt Mask Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic	
0																																		EXTI_IMR

**Bit IMR[22..0]:**

Avbrottsmask för avbrottslina x.

0: Avbrott är maskerat (deaktiverat)

1: Avbrott är aktiverat.

**EXTI\_EMR Event Mask Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic	
4																																		EXTI_EMR

**Bit EMR[22..0]:**

Eventmask för avbrottslina x.

0: Event är maskerat (deaktiverat)

1: Event är aktiverat.

**EXTI\_RTSTR Rising Trigger Selection Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic	
8																																		EXTI_RTSTR

**Bit RTSTR[22..0]:**

Trigg på positiv flank för avbrottslina x.

0: Trigg på positiv flank är maskerat (deaktiverat)

1: Trigg på positiv flank är aktiverat.

**EXTI\_FTSR Falling Trigger Selection Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic	
0xC																																		EXTI_FTSR

**Bit FTSR[22..0]:**

Trigg på negativ flank för avbrottslina x.

0: Trigg på negativ flank är maskerat (deaktiverat)

1: Trigg på negativ flank är aktiverat.

**EXTI\_SWIER Software Interrupt Event Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic	
0x10																																		EXTI_SWIER

**Bit SWIER[22..0]:**

Om avbrott är aktiverat för lina x, kan programvara aktivera detta genom att skriva '1' till motsvarande bit i detta register. Denna bit återställs genom skrivning till EXTI\_PR.

**EXTI\_PR Pending Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic	
0x14																																		EXTI_PR

**Bit PR[22..0]:**

Motsvarande bit sätts i detta register då ett triggvillkor är uppfyllt.

Biten återställs genom att skrivas med '1'.

0: Ingen Trigg.

1: Trigg har uppträtt

**GPIO****General Purpose Input Output**

GPIO A: 0x40020000

GPIO B: 0x40020400

GPIO C: 0x40020800

GPIO D: 0x40020C00

GPIO E: 0x40021000

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0																																	GPIO_MODER
4																																	GPIO_OTYPER
8																																	GPIO_OSPEEDR
0xC																																	GPIO_PUPDR
0x10																																	GPIO_IDR
0x14																																	GPIO_ODR
0x18																																	GPIO_BSRR
0x1C																																	GPIO_LCKR
0x20																																	GPIO_AFR1
0x24																																	GPIO_AFR2

**MODER Port mode register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic	
0x0																																		GPIO_MODER
																																		GPIO_OTYPER
																																		GPIO_OSPEEDR
																																		GPIO_PUPDR
																																		GPIO_IDR
																																		GPIO_ODR
																																		GPIO_BSRR
																																		GPIO_LCKR
																																		GPIO_AFR1
																																		GPIO_AFR2

För varje portpinne används 2 bitar i MODER för att konfigurera pinnen enligt:

00: ingång

01: utgång

10: alternativ funktion

11: analog

Registret har organiserats så att bitar 31,30 konfigurerar portpinne 15, bitar 29,28 konfigurerar portpinne 14 osv.

**OTYPER Output TYPE Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x04																																	GPIO_OTYPER

Bitar 31:16 är reserverade och ska inte ändras. För varje annan portpinne används en bit för att konfigurera pinnen enligt:

0: push-pull

1: open drain

**OSPEEDR Output SPEED Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x08																																	GPIO_OSPEEDR

Registret används för att kontrollera uppdateringsfrekvensen för en portpinne. Exakta frekvenser anges i processorns datablad.

00: low speed

01: medium speed

10: fast speed

11: high speed

**PUPDR Pull-Up/Pull-Down Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x0C																																	GPIO_PUPDR

För varje portpinne används 2 bitar i PUPDR för att konfigurera pinnen enligt:

00: floating

01: pull-up

10: pull-down

11: reserverad

**IDR Input Data Register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x10																																	GPIO_IDR

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic		
0x14																																			GPIO_ODR

Bitar 16 tom 31 används inte och ska hållas vid sitt RESET-värde, dvs 0. Bitar 0 t.o.m 15 sätter portens nivåer då pinnarna är konfigurerade som utgångar. Bitarna är både skriv- och läsbara, vid läsning ger biten det senast skrivna värdet till samma bit.

**BSSR Bit set reset register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	mnemonic
0x18	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	GPIO_BSRR

**LCKR Port configuration lock register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	mnemonic
0x1C																LCK	GPIO_LCKR

**AFRL Alternate function low register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic		
0x20	AFRL7	AFRL6	AFRL5	AFRL4	AFRL3	AFRL2	AFRL1	AFRL0																											GPIO_AFRL

**AFRH Alternate function high register**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic		
0x24	AFRH15	AFRH14	AFRH13	AFRH12	AFRH11	AFRH10	AFRH9	AFRH8																											GPIO_AFRH

**RCC**

**Reset and Clock control (0x40023800)**

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register	
0																																		RCC_CR
4																																		RCC_PLLCFGR
8																																		RCC_CFGR
0xC																																		RCC_CIR
0x10																																		RCC_AHB1RSTR
0x14																																		RCC_AHB2RSTR
0x18																																		RCC_AHB3RSTR
0x1C																																		
0x20																																		RCC_APB1RSTR
0x24																																		RCC_APB2RSTR
0x28																																		
0x2C																																		
0x30																																		RCC_AHB1ENR
0x34																																		RCC_AHB2ENR
0x38																																		RCC_AHB3ENR
0x3C																																		
0x40																																		RCC_APB1ENR
0x44																																		RCC_APB2ENR
0x48																																		
0x4C																																		
0x50																																		RCC_AHB1LPENR
0x54																																		RCC_AHB2LPENR
0x58																																		RCC_AHB3LPENR
0x5C																																		
0x60																																		RCC_APB1LPENR
0x64																																		RCC_APB2LPENR
0x68																																		
0x6C																																		
0x70																																		RCC_BDCR
0x74																																		RCC_CSR
0x78																																		
0x7C																																		
0x80																																		RCC_SSCGR
0x84																																		RCC_PLLI2SCFGR

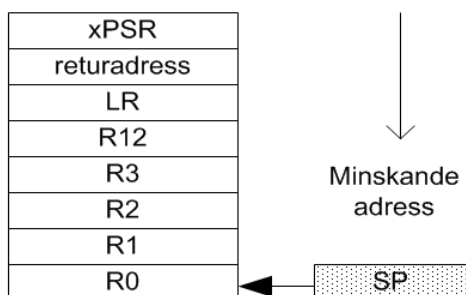
**RCC\_CR Clock control register**

## VEKTORTABELL

IRQ num	priority	name	description	vector offset
-			Reserved for initial stack pointer	0x0000 0000
-	fixed, -3	Reset	Reset	0x0000 0004
-14	fixed, -2	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000 0008
-13	fixed, -1	HardFault	All class of fault	0x0000 000C
-12	settable	MemManage	Memory management	0x0000 0010
-11	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-10	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
			Reserved	0x0000 001C - 0x0000 002B
-5	settable	SVCcall	System service call via SWI instruction	0x0000 002C
-4	settable	Debug Monitor	Debug Monitor	0x0000 0030
			Reserved	0x0000 0034
-2	settable	PendSV	Pendable request for system service	0x0000 0038
-1	settable	SysTick	System tick timer	0x0000 003C
0	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	settable	PVD	PVD through EXTI line detection interrupt	0x0000 0044
2	settable	TAMP_STAMP	Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line	0x0000 004C
4	settable	FLASH	Flash global interrupt	0x0000 0050
5	settable	RCC	RCC global interrupt	0x0000 0054
6	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	settable	EXTI2	EXTI Line2 interrupt	0x0000 0060
9	settable	EXTI3	EXTI Line3 interrupt	0x0000 0064
10	settable	EXTI4	EXTI Line4 interrupt	0x0000 0068
11	settable	DMA1_Stream0	DMA1 Stream0 global interrupt	0x0000 006C
12	settable	DMA1_Stream1	DMA1 Stream1 global interrupt	0x0000 0070
13	settable	DMA1_Stream2	DMA1 Stream2 global interrupt	0x0000 0074
14	settable	DMA1_Stream3	DMA1 Stream3 global interrupt	0x0000 0078
15	settable	DMA1_Stream4	DMA1 Stream4 global interrupt	0x0000 007C
16	settable	DMA1_Stream5	DMA1 Stream5 global interrupt	0x0000 0080
17	settable	DMA1_Stream6	DMA1 Stream6 global interrupt	0x0000 0084
18	settable	ADC	ADC1, ADC2 and ADC3 global interrupts	0x0000 0088
19	settable	CAN1_TX	CAN1 TX interrupts	0x0000 008C
20	settable	CAN1_RX0	CAN1 RX0 interrupts	0x0000 0090
21	settable	CAN1_RX1	CAN1 RX1 interrupt	0x0000 0094
22	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000 0098
23	settable	EXTI9_5 EXTI	Line[9:5] interrupts	0x0000 009C
24	settable	TIM1_BRK_TIM9	TIM1 Break interrupt and TIM9 global interrupt	0x0000 00A0
25	settable	TIM1_UP_TIM10	TIM1 Update interrupt and TIM10 global interrupt	0x0000 00A4
26	settable	TIM1_TRG_COM_TIM11	TIM1 Trigger and Commutation interrupts and TIM11 global interrupt	0x0000 00A8
27	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000 00AC
28	settable	TIM2	TIM2 global interrupt	0x0000 00B0
29	settable	TIM3	TIM3 global interrupt	0x0000 00B4
30	settable	TIM4	TIM4 global interrupt	0x0000 00B8
31	settable	I2C1_EV	I2C1 event interrupt	0x0000 00BC
32	settable	I2C1_ER	I2C1 error interrupt	0x0000 00C0
33	settable	I2C2_EV	I2C2 event interrupt	0x0000 00C4
34	settable	I2C2_ER	I2C2 error interrupt	0x0000 00C8
35	settable	SPI1	SPI1 global interrupt	0x0000 00CC
36	settable	SPI2	SPI2 global interrupt	0x0000 00D0
37	settable	USART1	USART1 global interrupt	0x0000 00D4
38	settable	USART2	USART2 global interrupt	0x0000 00D8
39	settable	USART3	USART3 global interrupt	0x0000 00DC
40	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000 00E0
41	settable	RTC_Alarm	RTC Alarms (A and B) through EXTI line interrupt	0x0000 00E4
42	settable	OTG_FS	WKUP USB On-The-Go FS Wakeup through EXTI line interrupt	0x0000 00E8
43	settable	TIM8_BRK_TIM12	TIM8 Break interrupt and TIM12 global interrupt	0x0000 00EC

44	settable	TIM8_UP_TIM13	TIM8 Update interrupt and TIM13 global interrupt	0x0000 00F0
45	settable	TIM8_TRG_COM_TIM14	TIM8 Trigger and Commutation interrupts and TIM14 global interrupt	0x0000 00F4
46	settable	TIM8_CC	TIM8 Capture Compare interrupt	0x0000 00F8
47	settable	DMA1_Stream7	DMA1 Stream7 global interrupt	0x0000 00FC
48	settable	FSMC	FSMC global interrupt	0x0000 0100
49	settable	SDIO	SDIO global interrupt	0x0000 0104
50	settable	TIM5	TIM5 global interrupt	0x0000 0108
51	settable	SPI3	SPI3 global interrupt	0x0000 010C
52	settable	UART4	UART4 global interrupt	0x0000 0110
53	settable	UART5	UART5 global interrupt	0x0000 0114
54	settable	TIM6_DAC	TIM6 global interrupt, DAC1 and DAC2 underrun error interrupts	0x0000 0118
55	settable	TIM7	TIM7 global interrupt	0x0000 011C
56	settable	DMA2_Stream0	DMA2 Stream0 global interrupt	0x0000 0120
57	settable	DMA2_Stream1	DMA2 Stream1 global interrupt	0x0000 0124
58	settable	DMA2_Stream2	DMA2 Stream2 global interrupt	0x0000 0128
59	settable	DMA2_Stream3	DMA2 Stream3 global interrupt	0x0000 012C
60	settable	DMA2_Stream4	DMA2 Stream4 global interrupt	0x0000 0130
61	settable	ETH	Ethernet global interrupt	0x0000 0134
62	settable	ETH_WKUP	Ethernet Wakeup through EXTI line interrupt	0x0000 0138
63	settable	CAN2_TX	CAN2 TX interrupts	0x0000 013C
64	settable	CAN2_RX0	CAN2 RX0 interrupts	0x0000 0140
65	settable	CAN2_RX1	CAN2 RX1 interrupt	0x0000 0144
66	settable	CAN2_SCE	CAN2 SCE interrupt	0x0000 0148
67	settable	OTG_FS	USB On The Go FS global interrupt	0x0000 014C
68	settable	DMA2_Stream5	DMA2 Stream5 global interrupt	0x0000 0150
69	settable	DMA2_Stream6	DMA2 Stream6 global interrupt	0x0000 0154
70	settable	DMA2_Stream7	DMA2 Stream7 global interrupt	0x0000 0158
71	settable	USART6	USART6 global interrupt	0x0000 015C
72	settable	I2C3_EV	I2C3 event interrupt	0x0000 0160
73	settable	I2C3_ER	I2C3 error interrupt	0x0000 0164
74	settable	OTG_HS_EP1_OUT	USB On The Go HS End Point 1 Out global interrupt	0x0000 0168
75	settable	OTG_HS_EP1_IN	USB On The Go HS End Point 1 In global interrupt	0x0000 016C
76	settable	OTG_HS_WKUP	USB On The Go HS Wakeup through EXTI interrupt	0x0000 0170
77	settable	OTG_HS	USB On The Go HS global interrupt	0x0000 0174
78	settable	DCMI	DCMI global interrupt	0x0000 0178
79	settable	CRYP	CRYP crypto global interrupt	0x0000 017C
80	settable	HASH_RNG	Hash and Rng global interrupt	0x0000 0180
81	settable	FPU	FPU global interrupt	0x0000 0184

Stackens utseende vid undantag:



# C QUICK REFERENCE GUIDE

## Strukturen hos ett C Program

```
#include ...      /* inkludera filer */
#define ...      /* makrodefinitioner */
/* Globala variabler*/
(typ) (identifierare);
/* Definition av funktioner */
```

## Definition av en funktion

```
(returvärde) (identifierare) (parameterlista)
{
(Definition av lokala variabler);
(programkod);
}
```

## Motsvarande funktionsdeklaration:

```
extern
(returvärde) (identifierare) (parameterlista);
```

## Variabeldefinition:

```
(typ) (identifierare {,identifierare} );
```

## Variabeldeklaration:

```
(typ) (identifierare {,identifierare} );
```

## Kommentarer

```
/*( kommentarer ) */
```

## Typdefinition

```
typedef(befintlig typ) (nytt typnamn);
```

*Exempel:*

```
typedef int KILOGRAMS;
```

*Funktionspekare:*

```
typedef(typ) (fp*) (typ {,typ} );
```

## Sammansatta typer

### Union

```
union (tag)
{
    (type) (member name);
    ...
}(variable name);
typedef union (tag)
{
    (type) (member name);
    ...
}(union type name);
```

### Struct

```
struct (tag)
{
    (type) (member name);
    ...
}(variable name);
typedef struct (tag)
{
    (type) (member name);
    ...
}(struct type name);
```

## Operatörer

Företräde	Operator	Assoc.
15	() [] . ->	V
14	! ~ ++ -- + - * & (typ) <b>sizeof</b>	H
13	* / %	V
12	+ -	V
11	<< >>	V
10	< <= > >=	V
9	== !=	V
8	&	V
7	^	V
6		V
5	&&	V
4		V
3	?:	V
2	= *= /= %= += -= &= ^=  = <<= >>=	H
1	,	V

## Aritmetikoperatörer

Operator	Operation	Företräde
+	addition	12
-	subtraktion	12
*	multiplikation	13
/	division (heltalsresultat)	13
%	restdivision	13
+(unärt)	påverkar ej operand	14
-(unärt)	negera operand	14
++	inkrement, addera 1 till operand	14
--	dekrement, subtrahera 1 från operand	14

## Relationsoperatörer

Operator	Operation	Företräde
==	likhet	9
!=	olikhet	9
>	större än	10
<	mindre än	10
>=	större än eller likhet	10
<=	mindre än eller likhet	10

## Bitoperatörer

Operator	Operation	Företräde
&	bitvis OCH	9
	bitvis ELLER	9
^	bitvis EXKLUSIVT ELLER	10
~	bitvis komplement	10
<<	vänsterskift	10
>>	högerskift	10

## Logikoperatörer

Operator	Operation	Företräde
&&	logiskt OCH	5
	logiskt ELLER	4
!	logisk negation	14



**Utmatning och inmatning****Utmatning:**

```
printf(" (formatterare) ", var{, var});
```

**Inmatning:**

```
scanf(" (formatterare) ", &(var){, &(var)});
```

**Formattering för konvertering:**

```
%d decimal
```

```
%u unsigned decimal
```

```
%o octal
```

```
%h hex
```

```
%e exponential
```

```
%f float
```

```
%c char
```

```
%s string
```

**Programflödeskontroll**

*for*-satsen används ofta för att skapa bundna programslingor.

```
for( uttryck1; uttryck2; uttryck3 )
    { satser }
```

*Exempel:*

```
for( i = 0; i < MAX; i++)
    { ... }
```

*while*- satsen skapar en iterativ slinga som upprepas så länge villkoret är sant.

```
while( uttryck )
{
    satser
}
```

*do-while* satsen är en variant, inte alls lika vanlig som *while*, men likafullt lämplig för vissa speciella programkonstruktioner.

```
do
{
    satser
} while( uttryck );
```

*if*-satsen utgör den enklaste flödeskontrollkonstruktionen:

```
if ( uttryck )
{
    satser
}
```

*if-else* satsen utvidgar med ett alternativt val, då villkoret är falskt.

```
if ( uttryck )
{
    satser
}else{
    satser
}
```

Med *switch-case* satsen konstruerar man enkelt flerval-konstruktioner:

```
switch( uttryck )
{
    case n:
    ...
    default:
    ...
}
```