Quick Guide MOP

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

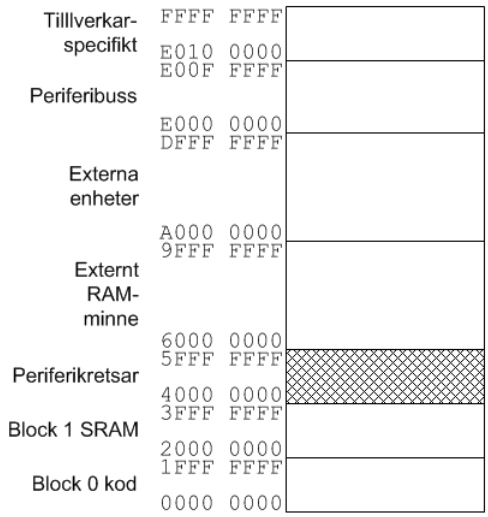
Institutionen för Data och informationsteknik
Chalmers 2018-12-26

Innehåll:

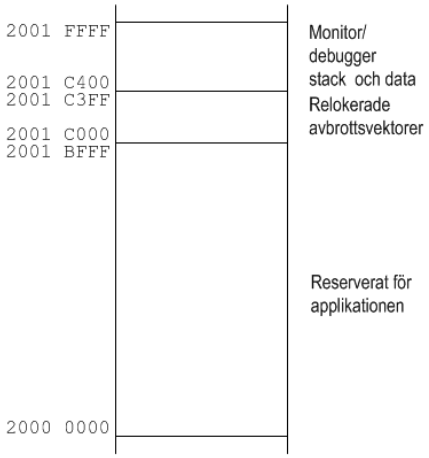
Tillhör:_____

MINNESDISPOSITION MD407

Adressrum



Block 1 SRAM



Periferibuss

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

Periferikretsar

A000 0000 – A000 0FFF	FSMC control reg	AHB3
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 4002 8C00 – 4002 8FFF 4002 8800 – 4002 8BFF 4002 8400 – 4002 87FF 4002 8000 – 4002 83FF	ETHERNET MAC	
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	

Instruktionslista ARM Thumb

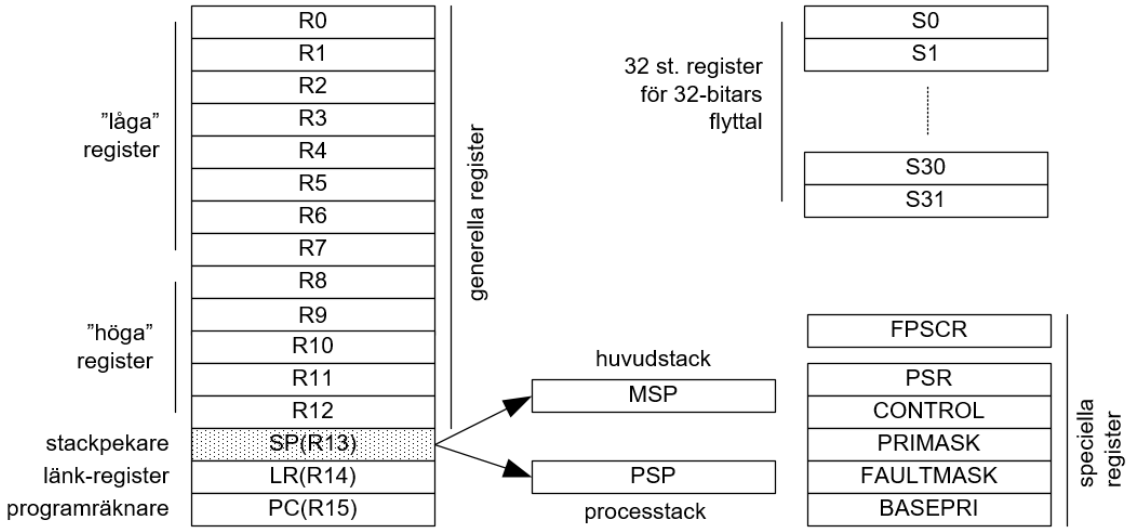
INSTRUKTIONSÖVERSKT

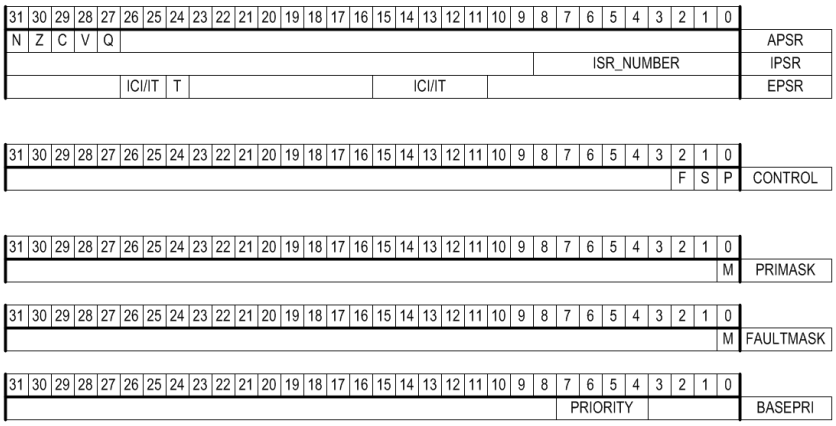
Instruktion	Storlek	Cortex M0	Cortex M0+	Cortex M1	Cortex M3	Cortex M4	Cortex M7	Ark.
ADC, ADD, (ADR), AND, ASR, B, BIC, BKPT, BLX, BX, CMN, CMP, CPS, EOR, LDM, (LDMIA, LDMFD), LDR, LDRB, LDRH, LDRSB, LDRSH, LSL, LSR, MOV, MUL, MVN, NOP, ORR, POP, PUSH, REV, REV16, REVSH, ROR, RSB, SBC, SEV, STM, (STMIA, STMEA), STR, STRB, STRH, SUB, SVC, SXTB, SXTH, 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	v7
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, SXTB, TBB, TBH, TEQ, TST, UBFX, UDIV, UMLAL, UMULL, USAT, UXTB, UXTH, WFE, WFI,YIELD	32-bit				x	x	x	
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, SMLALTTSMLALD, 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, VCMP, VCMPE, VCVT, VCVTR, VDIV, VLDM, VLDR, VMLA, VMLS, VMOV, VMRS, VMSR, 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





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 (*callee*) funktionen	
R11		
R10		
R9		
R8		
R7	Speciellt använder GCC R7 som pekare till aktiveringspost (*stack frame*)	
R6	Också dessa register är avsedda för variabler och temporärbruk	
R5	Om dom används måste dom sparas och återställas av	
R4	den anropade (*callee*) funktionen	
R3	parameter 4 / temporärregister	Dessa register sparas normalt sett inte över funktionsanrop men om, så är det den anropande (*caller*) 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 återhoppsadressen 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 priviligierad, en priviligierad 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 A6-115

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 assemblatorn.

Rd	Destination, (R0-R7).
label	Adressen till en instruktion eller data. Assemblatorn 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 A6-111

Adderar en konstant till värdet i SP och skriver resultatet till ett generellt 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 A6-111

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 A6-139

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 A6-141

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 A6-143

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 A6-144

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 A6-145

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 A6-146

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 A6-147

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 A6-148

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 A6-149

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 A6-177

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 A6-179

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.
Rn	Indexregister för adressberäkningen .

STRB – Store byte register A6-181

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 A6-180

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 A6-182

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 A6-183

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 A6-137

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 A6-175

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 A6-165

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 A6-167

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 A6-154

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 A6-155

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 A6-191

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 A6-191

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 A6-196

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 A6-195

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 A6-168

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<31:24> = Rm<7:0>;
Rd <23:16> = Rm<15:8>;
Rd <15:8> = Rm<23:16>;
Rd <7:0> = Rm<31:24>;

`REV16` – Byte-reverse packed halfword A6-169

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 <31:24> = Rm<23:16>;
Rd <23:16> = Rm<31:24>;
Rd <15:8> = Rm<7:0>;
Rd <7:0> = Rm<15:8>;

`REVSH` – Byte-reverse signed halfword A6-170

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<7:0> = Rm<15:8>;
Rd<31:8> = SignExtend(Rm<7:0>);

UTTRYCKSEVALUERING

Aritmetik- och logik- operationer

`ADC` – Add with carry A6-106

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 A6-109

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 A6-107

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 A6-173

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 A6-187

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 A6-185

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 A6-116

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 A6-121

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 A6-135

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 A6-164

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 A6-161

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 A6-192

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 A6-127

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 A6-129

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 A6-126

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 A6-150

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 A6-151

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 A6-152

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

`LSR` – Logical shift right A6-153

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 A6-117

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 A6-118

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 A6-171

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

PROGRAMFLÖDESKONTROLL

`B` – Unconditional branch A6-119

Ovillkorlig programflödesändring, destinationsadressen placeras i PC.

Syntax:

`B` `label`

`label` Destination.

`B<condition>` – Conditional branch A6-119

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
Enkla flaggtest		
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	Datatyp	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 A6-123

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.

`BX` – Branch with exchange A6-125

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:

```
BX      Rm
```

`Rm` Källregister (R0-R7) innehåller destinationsadressen och den minst signifikanta biten anger hur instruktion på adressen ska tolkas, Thumb eller ARM..

`BLX` – Branch with link and exchange A6-124

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-R7) 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 B4-310

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 B4-308

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 B4-306

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 spm 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 A6-122

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 A6-189

Instruktionen orsakar undantagshantering och är speciellt avsedd att implementera övergång till priviligierat 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 A6-163

Instruktionen utför ingenting.

Syntax:

```
NOP
```

`YIELD` – Yield hint A6-199

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 A6-197

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

Syntax:

```
WFE
```

`WFI` – Wait for interrupt hint A6-198

Instruktionsexekveringen avbryts och processorn inväntar avbrott.

Syntax:

```
WFI
```

`SEV` – Send event hint A6-174

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

Syntax:

```
SEV
```

`DSB` – Data synchronization barrier A6-134

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 A6-133

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 A6-136

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

Syntax:

```
ISB
```

`UDF` – Permanently undefined A6-193

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.

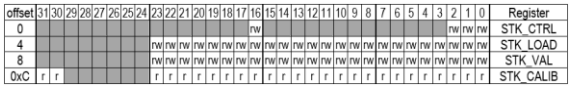
ASSEMBLERDIREKTIV

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.

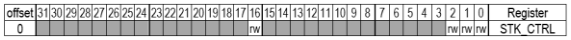
PERIFERIBUSS

SysTick

System Timer (0xE000E010)



STK_CTRL *Status och styrregister*



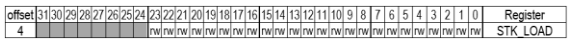
Bit 16: (COUNTFLAG):
Biten är 1 om räknaren räknat ned till 0. Biten nollställs då registret läses.

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.

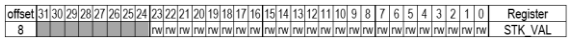
Bit 0: (ENABLE): Aktivera räknare
Aktiverar räknaren. Då ENABLE-biten sätts till 1 kopierar räknaren värdet från STK_LOAD till STK_VAL och räknar ned. Då STK_VAL slår om till 0, sätts COUNTFLAG till 1, avbrott genereras beroende på TICKINT. Därefter kopieras värdet från STK_LOAD igen, hela processen repeteras.
0: Räknare inaktiv.
1: Räknare aktiv.

STK_LOAD *Räknarintervall*



Bits 23:0 Värde för räknarintervall
Registret håller räknarintervallets 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.

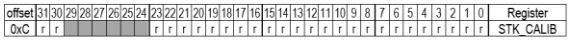
STK_VAL *Räknarvärde*



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 *Kaibreringsregister*



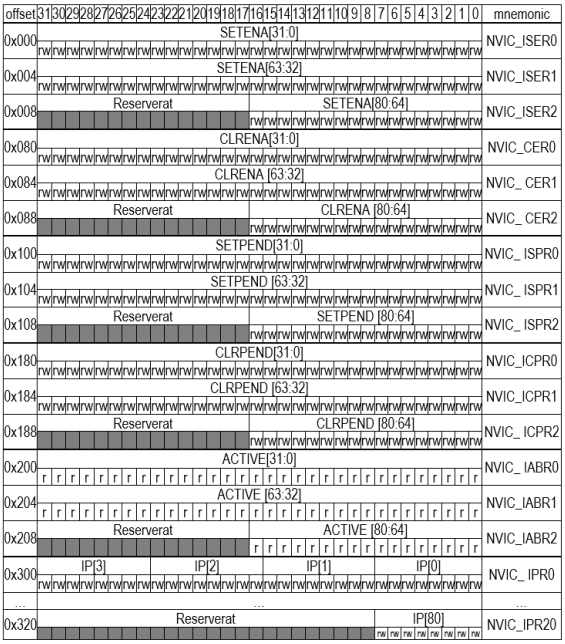
Registret är avsett för kalibreringsändamål och innehåller implementeringsspecifika detaljer. Vi behandlar inte dessa här.

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 Pendig 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



NVIC_ISERx: SETENA[80..0] *Interrupt set-enable bit*:
Skrivning: 0: ingen effekt, 1: möjliggör avbrott
Läsning: 0: avbrott avstängt, 1: avbrott möjligt

NVIC_ICERx: CLRENA[80..0] *Interrupt clear-enable bit*:
Skrivning: 0: ingen effekt, 1: omöjliggör avbrott
Läsning: 0: avbrott avstängt, 1: avbrott möjligt

NVIC_ISEPRx: SETPEND[80..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[80..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[80..0] *Interrupt active bit*:
Läsning: 0: avbrottstatus är inte ”aktivt”, 1: avbrottstatus är ”aktivt”

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.

SCB

System Control Block (0xE000ED00)

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																																	CPUID
4																																	ICSR
8																																	VTOR
0xC																																	AIRCR
0x10																																	SCR
0x14																																	CCR
0x18																																	SHPR1
0x1C																																	SHPR2
0x20																																	SHPR3
0x24																																	SHCSR
0x28																																	CFSR
0x2C																																	HFSR
0x30																																	
0x34																																	MMAR
0x38																																	BFAR
0x3C																																	AFSR

CPUID *Base register*

Innehåller information om produktnummer, revision, variant och tillverkare.

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	Implementer								Variant				Constant				PartNo												Revision				SCB_CPUID
	0	1	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	

Bits 31:24
Implementerares kod (0x41= ARM)
Bits 23:20
Variant nummer (0x0: revision 0)
Bits 19:16
Konstant alltid 0xF
Bits 15:4
ID 0xC24: = Cortex-M4
Bits 3:0
Revisionsnumber (0x1: = patch 1)

ICSR *Interrupt control and state register*

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	mnemonic
4	NMI PEND SET			PEND SV SET	PEND SV CLR	PEND ST SET	PEND ST CLR			ISR PEND ING				VECTPENDING [8:4]			SCB_ICSR
	rw			rw	w	rw	w			r				r	r	r	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	VECTPENDING [3:0]				RET TO BASE			VECTACTIVE[8:0]									
	r	r	r	r	r			rw	rw	rw	rw	rw	rw	rw	rw	rw	

ICSR tillhandahåller

- set-pending bit för NMI
- set-pending och clear pending bitar för PendSV och SysTick avbrott.

indikatorer för:

- om något avbrott är aktivt eller avvaktande
- aktivt undantagstyp
- avvaktande undantags prioritet

VTOR *Vector table offset register*

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

AIRCR *Application interrupt and reset control register*

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	mnemonic
0xC	VECTKEYSTAT[15:0](read)/ VECTKEY[15:0](write)																
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	END IAN ESS					PRIGROUP								SYS RESET REQ	VECT CLR ACT	VECT RESET	SCB_AIRCR
	r					rw	rw	rw						w	w	w	

SCR *System control register*

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	mnemonic
0x10																	SCB_SCR
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
												SEV ON PEND		SLEEP DEEP	SLEEP ON EXIT		
												rw		rw	rw		

CCR *Configuration and control register*

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	mnemonic
0x14																	SCB_CCR
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
							STK ALIGN	BFHF NMIGN				DIV_0 _TRP	UN ALIGN TRP		USER SETM PEND	NON BASE THRD ENA	

SHPR1 *System handler priority register 1*

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
0x18									rw	rw	rw	rw	0	0	0	0	rw	rw	rw	rw	0	0	0	0	rw	rw	rw	rw	0	0	0	0	SCB_SHPR1
	PRI6: Usage fault								PRI5: Bus fault								PRI4: MMU fault																

SHPR2 *System handler priority register 2*

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
0x1C	rw	rw	rw	rw	0	0	0	0																									SCB_SHPR2
	PRI11: SVCall																																

SHPR3 *System handler priority register 3*

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	rw	rw	rw	rw	0	0	0	0	rw	rw	rw	rw	0	0	0	0																	SCB_SHPR3
	PRI15: SysTick								PRI14: PendSV																								

SHCSR *System handler control and state register*

CFSR *Configurable fault status register*

HFSR *Hard fault status register*

MFAR *Mem manage fault address register*

BFAR *Bus fault address register*

AFSR *Auxiliary fault status register*

FPU

Floating point unit coprocessor access control (0xE000ED88)

Floating point unit (0xE000EF30-0xE000EF44)

Adress	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
0xE000ED88									rw	rw	rw	rw																					CPACR
0xE000EF34	rw	rw																							rw	rw	rw	rw	rw	rw	rw	rw	FPCCR
0xE000EF38	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			FPCAR
0xE000EF3C						rw	rw	rw	rw	rw																							FPDSCR

CPACR *Coprocessor access control register*

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
								CP11			CP10																					CPACR

Bits 23:20:
Dessa bitar bestämmer åtkomst av FPU:n.
00: Ingen åtkomst, försök att använda FPU:n resulterar i *NOCP UsageFault*.
01: Endast åtkomst i priviligierat tillstånd. Försök till åtkomst från icke priviligierat tillstånd resulterar i *NOCP UsageFault*.
10: Reserverad kombination, om detta bitmönster används är resultatet inte predikterbart.
11: Full åtkomst, FPU:n kan användas fullt ut.

FPCCR *FP context control register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Register
ASPEN	LSPEN															FPCCR
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
							MON RDY		BF RDY	MM RDY	HF RDY	THREAD		USER	LSPACT	

FPCAR *FP context address register*

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
ADRESS[31:3]																																FPCAR

FPDSCR *FP default status control register*

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
					AHP	DN	FZ	RMode																									FPDSCR

MPU

Memory protection unit (0xE000ED90-0xE000EDB8)

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																																	MPU_TYPER
4																																	MPU_CTRL
8																																	MPU_RNR
0xC																																	MPU_RBAR
0x10																																	MPU_RASR
0x14																																	MPU_RBAR_A2
0x18																																	MPU_RASR_A2
0x1C																																	MPU_RBAR_A3
0x20																																	MPU_RASR_A3

MPU_TYPER *MPU type register*

MPU_CTRL *MPU control register*

MPU_RNR *MPU region number register*

MPU_RBAR *MPU region base address register*

MPU_RASR *MPU region attribute and size*

MPU_RBAR_A1 *MPU region base address register*

MPU_RASR_A1 *MPU region attribute and size*

MPU_RBAR_A2 *MPU region base address register*

MPU_RASR_A2 *MPU region attribute and size*

MPU_RBAR_A3 *MPU region base address register*

MPU_RASR_A3 *MPU region attribute and size*

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 UART_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 dataregister 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 följd 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, upptäckts. 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 ettställts 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	USART_DR
							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_Mantissa[11:0]												DIV_Fraction[3:0]				USART_BRR

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	TXE IE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	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.	LIN EN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBD IE	LBDL	Res.	ADD[3:0]				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 detek¬tera 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 stoppbit och 1.5 stoppbit 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 låser data
1: Andra klocktransition låser 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 intr ändras 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	CTS IE	CTSE	RTSE	DMAT	DMA R	SCEN	NACK	HDSEL	IRLP	IREN	EIE	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 genereas 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_RTSR
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									rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	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									rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	EXTI_EMR

Bit EMR[22..0]:
Eventmask för avbrottslina x.
0: Event är maskerat (deaktiverat)
1: Event är aktiverat.

EXTI_RTSR *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									rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	EXTI_RTSR

Bit RTSR[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									rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	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									rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	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									rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	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_AFRL
0x24																																	GPIO_AFRH

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	pin15		pin14		pin13		pin12		pin11		pin10		pin9		pin8		pin7		pin6		pin5		pin4		pin3		pin2		pin1		pin0		GPIO_MODER
	0x03								0x02								0x01								0x00								

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																	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	GPIO_OTYPER
																	0x05								0x04								

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	pin15		pin14		pin13		pin12		pin11		pin10		pin9		pin8		pin7		pin6		pin5		pin4		pin3		pin2		pin1		pin0		GPIO_OSPEEDR
	0x0B								0x0A								0x09								0x08								

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	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	GPIO_PUPDR
	0x0F								0x0E								0x0D								0x0C								

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																	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	GPIO_IDR
																	0x11								0x10								

Bitar 16 tom 31 används inte och ska hållas vid sitt RESET-värde, dvs 0. Bitar 0 t.o.m 15 avspeglar portens nivåer då pinnarna är konfigurerade som ingångar.

ODR *Output 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
0x14																	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	GPIO_ODR
																	0x15								0x14							

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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0	

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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0	

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[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]				AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]				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[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]				AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]				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_ABP1RSTR
0x24																																	RCC_ABP2RSTR
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

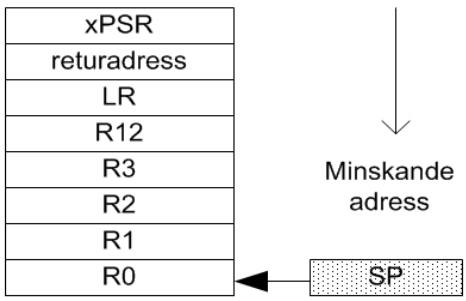
RCC_CR *Clock control register*

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	SVCall	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 interrup	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);
```

Operatorer

Före-träde	Operator	Assoc.
15	`() [] . ->`	V
14	`! ~ ++ -- + - * & (typ) sizeof`	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

Aritmetikoperatorer

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

Relationsoperatorer

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

Bitoperatorer

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

Logikoperatorer

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 flervals-konstruktioner:

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