Inhoud

Algemeen

Vanaf ARM 6 volledig 32 bits PC. Dit betekent:

- PSR heeft een apart register: CPSR
- PSR kan tegelijkertijd met PC worden bewaard bij springen naar andere modus. Elke belangrijke modus heeft nu een eigen SPSR en bevat de oude PSR van de modus.
- Nieuwe instructies voor toegang tot CPSR en SPSR

Door 32 bits PSR zijn er meer processor modi oa: Abort modus en Undefined instruction modus

Processor modi

ARM ingesteld als 32 bits processor levert 10 processor modi:

- User modus: normaal voor applicaties
  of User26 modus: 26 bits versie
- FIQ modus: snelle afhandeling van hardware
  of FIQ26 modus: 26 bits versie
- IRQ modus: voor de standaard interupts
  of IRQ26 modus: 26 bits versie
- SVC modus: beschermde modus voor OS
  of SVC26 modus: 26 bits versie
- Abort (ABT) modus: springt in deze modus bij ophalen data of instructie op ongeldig adres.
- Undefined (UND) modus: springt in deze modus bij een ongedefinieerde instructie.

Registers in diverse modi

User26	**SVC26**	**IRQ26**	**FIQ26**
R0	R0	R0	R0
⋮	⋮	⋮	⋮
R7	R7	R7	R7
R8	R8	R8	R8_FIRQ
R9	R9	R9	R9_FIRQ
R10	R10	R10	R10_FIRQ
R11	R11	R11	R11_FIRQ
R12	R12	R12	R12_FIRQ
R13	R13_SVC	R13_IRQ	R13_FIRQ
R14	R14_SVC	R14_IRQ	R14_FIRQ
..... R15 (PC/PSR)			

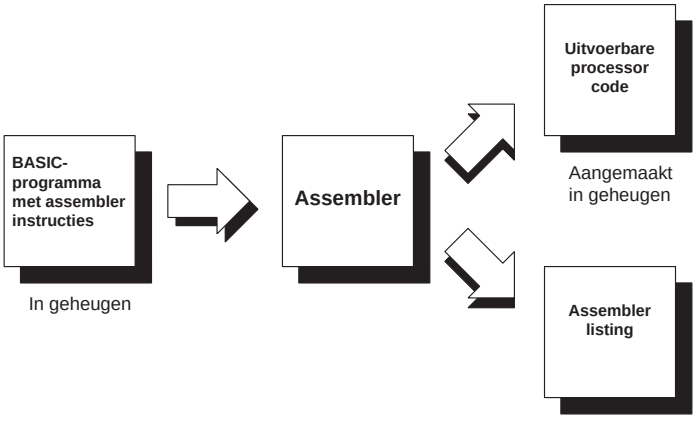
User	**SVC**	**IRQ**	**ABT**	**UND**	**FIQ**
R0	R0	R0	R0	R0	R0
⋮	⋮	⋮	⋮	⋮	⋮
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8_FIRQ
R9	R9	R9	R9	R9	R9_FIRQ
R10	R10	R10	R10	R10	R10_FIRQ
R11	R11	R11	R11	R11	R11_FIRQ
R12	R12	R12	R12	R12	R12_FIRQ
R13	R13_SVC	R13_IRQ	R13_ABT	R13_UND	R13_FIRQ
R14	R14_SVC	R14_IRQ	R14_ABT	R14_UND	R14_FIRQ
..... R15 (PC)					
..... CPSR					
	SPSR_SVC	SPSR_IRQ	SPSR_ABT	SPSR_UND	SPSR_FIRQ

CPSR en SPSR

31	30	29	28	...	7	6	-	4	3	2	1	0	
N	Z	C	V		I	F		M4	M3	M2	M1	M0	
								0	0	0	0	0	User26 modus
								0	0	0	0	1	FIQ26 modus
								0	0	0	1	0	IRQ26 modus
								0	0	0	1	1	SVC26 modus
								1	0	0	0	0	User modus
								1	0	0	0	1	FIQ modus
								1	0	0	1	0	IRQ modus
								1	0	0	1	1	SVC modus
								1	0	1	1	1	ABT modus
								1	1	0	1	1	UND modus

De Assembler

Assembler geïntegreerd in BASIC.



Binair	Hexadecimaal	Assembler
%11100001101000000001000000000010	&E1A01002	MOV **R1,R2**
%11100000100000110000000000000101	&E0831005	ADD **R1,R3,R5**
%11100001010100010000000000000101	&E1510005	CMP **R1,R5**

ARM Assembler-instructies

Assembler-instructies bestaan uit 3 delen:

- type instructie: MOV, ADD SUB etc
- operands waarop de instructie werkt
- conditie

Assembler-instructies

```
MOV  0,3
MOV  R0,R3
MOV  positie,teller
MOV  PC,r14
```


Simpel BASIC Assembler-programma

```
REM Onze eerste BASIC Assembler-programma
DIM code% 31
P%=code%
[
.hallo                    ;label van string
 EQUS   "Hallo allemaal"  ;string in geheugen
 EQUB   0                 ;afsluitende 0
 ALIGN                    ;P% weer word aligned
.programma                ;label Assembler-programma
 ADR    R0,hallo          ;in R0 komt adres van
                          ;onze string
 SWI    "OS_Write0"       ;print string
 MOV    PC,R14            ;stop Assembler-programma
                          ;en terug naar BASIC
]
CALL programma
END
```

BASIC in Assembler

BASIC-operaties kunnen in Assembler-instructies gebruikt worden

Voorbeelden

```
MOV R0,#67
MOV R0,#ASC("C")
MOV R0,fred%
ADD R0,R0,X%*2+5
ADD R0,R0,#%10100101
MOV R0,#&F000
MOV R1,#(start% DIV 256)
MOV R2,#INT(SIN(DEG(60))*100)
```

Opmerking: BASIC-operaties worden uitgevoerd tijdens assembleren. Dus niet bij uitvoeren processor code!

Data transfer van en naar BASIC

Van BASIC naar machine-code:

Register 0 tot en met 7 kan gevuld worden via de variabelen `A%` tot en met `H%`

Van machine-code naar BASIC:

Inhoud van register 0 kan met
`<var> = USR(<adres/label Assembler-programma>)`
teruggegeven worden in variabele `<var>`

Condities

Instructie condities:

- Voorwaarde waaronder een instructie uitgevoerd mag worden
- 4 bits van de instructie: 16 condities
- Condities hangen af van de statusvlaggen: NZCV
- Statusvlaggen hangen af van vorige instructies

Voorbeeld

Conditie code %1011 betekent:

dat:	N vlag	=	SET	en	V vlag	=	CLEAR
of:	N vlag	=	CLEAR	en	V vlag	=	SET
of:	Z vlag	=	SET				

Voer instructie uit als vorige vergelijking: operand 1 $\leq$ operand 2

De conditie codes

In Assembler, conditie van instructie bestaat uit 2 letters

Assembler conditie codes

`EQ`:	Gelijk	`NE`:	Ongelijk
`VS`:	Overflow set	`VC`:	Overflow clear
`AL`:	Altijd	`NV`:	Nooit
`HI`:	Hoger	`LS`:	Lager of gelijk
`PL`:	Positief	`MI`:	Negatief
`CS`:	Carry set	`CC`:	Carry clear
`GE`:	Groter dan of gelijk	`LT`:	Lager dan
`GT`:	Groter dan	`LE`:	Lager dan of gelijk

`SUBPL R0,R1,R2` wordt uitgevoerd als resultaat van een vorige instructie positief was

Conditie EQ/NE

EQ: Gelijk
Conditie: Z vlag = set

Voorbeeld

`CMP`	`R5,R10`	Vergelijk de inhoud van register 5 met 10
`ADDEQ`	`R5,R5,#2`	Als inhoud van R5 en R10 gelijk tel dan 2 op bij de inhoud in R5

NE: Ongelijk
Conditie: Z vlag = clear

Voorbeeld

`CMP`	`R2,R0`	Vergelijk inhoud R2 met R0
`SUBNE`	`R2,R2,R0`	Als 2 en 0 niet gelijk dan trek de inhoud van elkaar af en bewaar resultaat in register 2

Conditie VS/VC

VS: Overflow set
Conditie: V vlag = set

Voorbeeld

`SWI "XOS_Byte"`	Voor de SYS call OS_Byte uit
`BVS error`	Spring naar error routine wanneer OS_Byte een foutmelding geeft

VC: Overflow clear
Conditie: V vlag = clear

Conditie PL/MI

MI: Negatief
Conditie: N vlag = set

Voorbeeld

```
SUBS   R0,R0,#5    Trek 5 af van de inhoud in R0
ADDMI  R0,R0,#5    Als vorig resultaat 'n negatief getal is tel dan er
                   weer 5 bij op
```

PL: Positief
Conditie: N vlag = clear

Conditie: CS/CC

CS: Carry set of HS: hoger of zelfde (unsigned)
Conditie: C vlag = set

Voorbeeld

```
ADDS   R1,R1,#1024    Voeg 1024 toe aan de inhoud van R1
ADDCS  R2,R2,#1       Als carry gezet tel dan 1 op bij inhoud R2
```

CC: Carry clear of LO: lager (unsigned)
Conditie: C vlag = clear

Voorbeeld

```
CMP    R1,#ASC("A")   Vergelijk inhoud R1 met ascii-waarde van
                      karakter A
MOVCC  R1,R1,#0       Als lagere ascii-waarde dan wordt inhoud
                      R1 0
```

Conditie: AL/NV

AL: Altijd
Conditie: Altijd

Voorbeeld

```
ANDAL R0,R1,R2   Voer altijd R0 = R1 AND R2 uit
ADD   R1,R1,#2   Tel altijd 2 op bij R1 (AL is standaard conditie)
```

NV: Nooit
Conditie: Nooit

Instructie wordt met deze conditie nooit uitgevoerd. Maak hier geen gebruik van in je code!

Conditie: HI/LS

HI: Hoger (unsigned)
Conditie: C vlag = set en Z vlag = clear

Voorbeeld

```
CMP   R11,R6   Vergelijk R11 en R6
MOVHI R11,#0   Als R11 > R6 dan R11=0
```

LS: Lager of zelfde (unsigned)
Conditie: C vlag = clear of Z vlag = set

Voorbeeld

```
CMP   R4,R2    Vergelijk R4 met R2
ADDLS R4,R4,#1 Als R4 ≤ R2 dan verhoog R4 met 1
```

Conditie: GE/LT

GE: Groter dan of gelijk (signed)

Conditie: **N vlag = set en V vlag = set**
of: **N vlag = clear en V vlag = clear**
of: **Z vlag = set**

Voorbeeld

```
CMP   R5,R2       Vergelijk R11 en R6
SUBGE R5,R5,#2    Als R5 ≥ R2 dan trek 2 af van R5
```

LT: Lager dan (signed)

Conditie: **N vlag = set en V vlag = clear**
of: **N vlag = clear en V vlag = set**

Conditie: GT/LE

GT: Groter dan (signed)

Conditie: **N vlag = set en V vlag = set**
of: **N vlag = clear en V vlag = clear**
en: **Z vlag = clear**

Voorbeeld

```
CMP   R8,R9            Vergelijk R8 en R6
SWIGT 256+ASC(">")     Als R8 > R9 dan print het > karakter
```

LE: Lager dan of gelijk (signed)

Conditie: **N vlag = set en V vlag = clear**
of: **N vlag = clear en V vlag = set**
of: **Z vlag = set**

Statusvlaggen aanpassen

Met S geef je aan dat het resultaat van de instructie de statusvlaggen mag aanpassen:

- Voordeel: Resultaat van 1 instructie kan je gebruiken om conditioneel meerdere instructies uit te voeren
- Vergelijkingsinstructies zetten altijd de vlaggen: CMP, CMN, TEQ, TST, (SWI)

Voorbeelden

```
.lus              label lus
SUBS   R1,R1,#1   verlaag teller met 1 en zet statusvlaggen
SUB    R5,R2,#5   R5=R2-5, zet geen statusvlaggen
BNE    lus        als teller ≠ 0 dan nog een keer de lus

CMP    R0,#32     vergelijk R0 met ascii karakter spatie
SUBCC  R1,R1,#1   als lager dan 32 verlaag teller met 1
MOVCC  R0,#0      als lager dan 32 R0 = afsluitende 0
```


Overzicht instructies

De dataverwerking instructies

ADD	ADC	SUB	SBC
RSB	RSC	MUL	MLA
MOV	MVN	CMP	CPN
AND	ORR	EOR	
BIC	TST	TEQ	

Formaat Assembler-instructie:

< Type Assembler-instructie > <Doel register>, <Operand 1>, <Operand 2>

Operands

Operand 1: Register 0-15
Operand 2:

1. Een register: ADD R1, R2, **R3**
2. Een directe constante: AND R0, R4, **#%101**
3. Een verschoven register operand:
   - SUB R0, R1, **R2, LSL #3**
   - ADD R0, R1, **R2, LSL R10**
   - EOR R0, R1, **R2, ASR #1**

Bereik directe constantes

Instructie bevat 12 bits voor de directe constante:

- 8 bits voor de nummerieke constante
- 4 bits voor positie in operand

Bit 31 Bit 0	Positie
....76543210	0
10....765432	1
3210....7654	2
543210....76	3
76543210....	4
..76543210....	5
....76543210....	6
......76543210....	7
........76543210....	8
..........76543210....	9
............76543210....	10
..............76543210....	11
................76543210....	12
..................76543210....	13
....................76543210....	14
......................76543210..	15

LSL: logische shift naar links

Syntax: LSL #n of LSL Rx

Voorbeeld LSL #1

Voor:	x	←	b31 b30 b29 . . . b2 b1 b0	←	0
Na:	b31		b30 b29 b28 . . . b1 b0 0		
	Carry		Data word		

ASL werkt hetzelfde als **LSL**

LSR: logische shift naar rechts

Syntax: LSR #n of LSR Rx

Voorbeeld LSR #1

Voor:	0	→	b31 b30 b29 . . . b2 b1 b0	→	x
Na:			0 b31 b30 . . . b3 b2 b1		b0
			Data word		Carry

ASR: Rekenkundige shift naar rechts

Syntax: ASR #n of ASR Rx

Voorbeeld ASR #1

Voor:	b31	→	b31 b30 b29 . . . b2 b1 b0	→	x
Na:			b31 b31 b30 . . . b3 b2 b1		b0
			Data word		Carry

ROR: Roteer naar rechts
Syntax: ROR #n of LSR Rx

Voorbeeld ROR #1

Voor:	→	b31 b30 b29 . . . b2 b1 b0	→	x
Na:		b0 b31 b30 . . . b3 b2 b1		b0
		Data word		Carry

RRX: Roteer rechts met carry
Syntax: RRX

Voorbeeld RRX

Voor:	→	b31 b30 b29 . . . b2 b1 b0	→	x
Na:		x b31 b30 . . . b3 b2 b1		b0
		Data word		Carry

Instructies

ADD: Optellen
Syntax: ADD (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand één + operand twee
Vlaggen: N,Z,C,V

ADC: Optellen met carry
Syntax: ADC (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand één + operand twee + carry
Vlaggen: N,Z,C,V

64 bit optellen

32 bit hoog	32 bit laag
01101010101000111001110011001100	10110101000110001100011100011110
01010111000110100100011001100011	10101010101010011111001101001100 1

11000001101111011110001100110000(1)0101111111000010101011011011011 1

Instructies

SUB: Aftrekken
Syntax: SUB (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand één - operand twee
Vlaggen: N,Z,C,V

SBC: Aftrekken met carry
Syntax: SBC (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand één - operand twee - NOT(carry)
Vlaggen: N,Z,C,V

Als 'geleend' moet worden op bit 31 (0 - 1): carry = clear (0)
Als niet 'geleend' moet worden op bit 31: carry = set (1)

3 juni 2006
32 bits architectuur
BASIC Assembler
ARM instructieset
Formaat dataverwerking instructies
Shift instructies
Dataverwerking instructies

Instructies

RSB: Omgekeerd aftrekken
Syntax: RSB (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand twee - operand één
Vlaggen: N,Z,C,V

SBC: Omgekeerd Aftrekken met carry
Syntax: SBC (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand twee - operand één NOT(carry)
Vlaggen: N,Z,C,V

Instructies

MOV: Verplaats data
Syntax: MOV (S) <doel reg>, <operand2>
Operatie: resultaat = operand twee
Vlaggen: N,Z,C

MVN: Verplaats geïnverteerde data
Syntax: MVN (S) <doel reg>, <operand2>
Operatie: resultaat = NOT operand twee
Vlaggen: N,Z,C

Voorbeelden

`MVN R0,#0`	zet -1 in R0
`MVN R1,#9`	zet -10 in R1
`MVN R6,R7,LSR #1`	R6 = NOT(R7/2)

3 juni 2006
32 bits architectuur
BASIC Assembler
ARM instructieset
Formaat dataverwerking instructies
Shift instructies
Dataverwerking instructies

Instructies

CMP: Vergelijk
Syntax: CMP <operand1>, <operand2>
Operatie: resultaat operand één - operand twee zet vlaggen
Vlaggen: N,Z,C,V

MVN: Vergelijk negatief
Syntax: CMN <operand1>, <operand2>
Operatie: resultaat operand één - (-operand twee) zet vlaggen
Vlaggen: N,Z,C,V

Voorbeelden

`CMN R5,R7`	vergelijk R5 met -R7
`CMN R1,#9`	vergelijk R6 met -9
`CMN R3,R0,LSL #1`	vergelijk R3, met -R0*2

Instructies

AND: logische AND
Syntax: AND (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand één AND operand twee
Vlaggen: N,Z,C

ORR: logische ORR
Syntax: ORR (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand één OR operand twee
Vlaggen: N,Z,C

Voorbeelden

```
AND  R5,R5,#%1111    R5 = R5 AND %1111 (bits 31-4 worden 0)
ORR  R7,R7,#%1100    zet bit 2 en 3 in R7
ORRS R5,R5,#2        R5 = R5 OR 2 (resultaat zet vlaggen)
```


3 juni 2006
32 bits architectuur
BASIC Assembler
ARM instructieset
Formaat dataverwerking instructies
Shift instructies
Dataverwerking instructies

Instructies

EOR: logische exclusieve OR
Syntax: EOR (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand één EOR operand twee
Vlaggen: N,Z,C

BIC: Clear bits
Syntax: BIC (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand één AND(NOT(operand twee))
Vlaggen: N,Z,C

Voorbeelden

```
EOR R7,R7,#%11    zet bit 0 en 1 op 0 in R7
BIC R5,R5,#%11    zet bits 0 en 1 op 0 in R5
BIC R6,R6,R6      zet bit in R6 volgens R6 (R6=0)
```

Instructies

TST: Test bits
Syntax: TST <operand1>, <operand2>
Operatie: resultaat operand één AND operand twee zet vlaggen
Vlaggen: N,Z,C

TEQ: Test gelijkheid
Syntax: TEQ <operand1>, <operand2>
Operatie: resultaat = operand één EOR (NOT(operand twee))
Vlaggen: N,Z,C

Voorbeelden

```
TST   R1,#%1000   controleer of bit 3 is gezet
CMPNE R1,R2       als bit 3 gezet dan vergelijk R1 met R2
TEQEQ R1,R3       als bit 3 niet gezet of R1 is gelijk aan R2
                  dan vergelijk R1 met R3
```


3 juni 2006
32 bits architectuur
BASIC Assembler
ARM instructieset
Formaat dataverwerking instructies
Shift instructies
Dataverwerking instructies

Instructies

MUL: Vermenigvuldigen
Syntax: MUL (S) <doel reg>, <operand1>, <operand2>
Operatie: resultaat = operand één * operand twee
Vlaggen: N,Z en C is onbekend

Restricties:

- Doel register, operand 1 en operand 2 zijn registers
- Operand 2 geen directe constante of verschoven register
- Doel register en operand 1 niet hetzelfde register

MLA: Vermenigvuldig met optellen
Syntax: MLA (S) <doel reg>, <operand1>, <operand2>, som
Operatie: resultaat = (operand één * operand twee) + som
Vlaggen: N,Z en C is onbekend