Inhoud

1. 4 november 2006
   - Hints, tips en trucs
   - Modules

Vermenigvuldigen

```
MOV R0,R0,ASL #n      ;vermenigvuldig met 2^n
RSB R0,R0,R0,ASL #n   ;R0=R0*(2^n-1)
ADD R0,R0,R0,ASL #n   ;R0=R0*(2^n+1)
```

Voorbeelden

```
;vermenigvuldig met 10
MOV R0,R0,ASL #1      ;R0 maal 2
ADD R0,R0,R0, ASL #2 ;R0 maal 5
;vermenigvuldig met 31
RSB R0,R0,R0,ASL #5
```

Delen

```
; getal/deler = quotient met rest
; r_teken = teken rest
; q_teken = teken quotient
ANDS   r_teken,getal,#1<<31        ;zet teken negatief als getal negatief
RSBMI  getal,getal,#0              ;als negatief, maak getal positief
EOR    q_teken,r_teken,deler       ;bepaal teken van quotient
CMP    deler,#0                    ;controleer teken van deler
RSBMI  deler,deler,#0              ;als deler negatief, maak positief
MOV    rest,#0                     ;initialiseer rest
MOV    quotient,#0                 ;initialiseer quotient
MOV    posteller,#1<<31            ;initialiseer positieteller
.delen_loop                        ;verwerk alle alle 32 bits
MOVS   getal,getal,ASL#1           ;shift 1 plaats links
ADC    rest,rest,rest              ;rest maal 2 + bit 31 (carry)
CMP    rest,deler                  ;als rest >= deler
SUBGE  rest,rest,deler             ;dan rest=rest-deler
ORRGE  quotient,quotient,posteller ;dan voeg positiebit toe aan quotient
MOVS   posteller,posteller,LSR#1   ;zet posteller 1 bit naar links
BNE    delen_loop                  ;niet alle bits verwerkt dan volgende bit
CMP    q_teken,#0                  ;check teken quotient
RSBMI  quotient,quotient,#0        ;als negatief dan maak quotient negatief
CMP    r_teken,#0                  ;check teken rest
RSBMI  rest,rest,#0                ;als negatief dan maak negatief
```


Meerdere if constructies

Case When Otherwise structuren in Assembler:

```
 CMP   R0, #(eindtabel-tabel)/4 ;R0=routine nummer
 ADDCC PC, PC, R0, LSL#2          ;R0 niet groter dan aantal
                                  ;opties in tabel dan spring
                                  ;naar juiste procedure
 B      ongeldig                  ;als var groter,
                                  ;dan ongeldig
.tabel
 B      nul
 B      een
 B      twee
.eindtabel
```

Vlaggen aan en uitzetten

Vlaggen wijzigen met 26/32 neutrale code:

V-vlag

Aanzetten	Uitzetten
`CMP R0,#1<<31`	`CMP R0,#1<<31`
`CMNVC R0,#1<<31`	`CMNVS R0,#1<<31`

N-vlag

Aanzetten	Uitzetten
`MVN R0,#0`	`MOV R0,#1`
`TEQ R0,#0`	`TEQ R0,#0`

Z-vlag

Aanzetten	Uitzetten
`TEQ R0,R0`	`TEQ PC,#0`

C-vlag

Aanzetten	Uitzetten
`CMP PC,#0`	`ADDS R0,R0,#0`

Diversen

Laden van signed bytes:

```
LDRB R0, <adres>   ;laad byte
MOV  R0,R0,LSL #24 ;schuif op tot 31..24
MOV  R0,R0,ASR #24 ;zet terug, signed
```

Wisselen inhoud van 2 registers:

```
EOR R0,R0,R1
EOR R1,R0,R1
EOR R0,R0,R1
```

Intervallen

Voorbeeld:Zit getal in interval 1-10 of 100-200

```
CMP     R0,#1       ;vergelijk getal met ondergrens
RSBGES  R1,R0,#10   ;als groter of gelijk dan
                    ;bovengrens-getal, alles binnen
                    ;de grenzen 1-10 geeft een
                    ;positief resultaat, zet vlaggen
CMPLT   R0,#100     ;als getal>10 dan vergelijk met
                    ;ondergrens 100
RSBGES  R1,R0,#200  ;als groter of gelijk dan
                    ;bovengrens-getal, zet vlaggen
MVNGE   R3,#0       ;R3=-1 als getal in 1-10 of
                    ;100-200 ligt
```


Debuggen

Handige debuggingtools:

- Reporter (`http://www.avisoft.force9.co.uk/Reporter.htm`):
  - SWI Report_Registers, Report_Regs
  - SWI Report_Text0, Report_TestS
- ARMAlyser (`http://www.armclub.org.uk/free/`):

Wat is een module

Een module (&FFA) is een stuk software met de volgende eigenschappen:

- Uitbreiding op het OS: SCSIFS, ABCLibrary, Chimemod, Smartmenu
- Code draait in RMA of ROM en moet plaats-onafhankelijk zijn
- Private word (werkruimte) elke submodule heeft z'n eigen private word
- Module header, hiermee communiceert RISC OS met module:
  - Initialisatie, starten en beëindigen van module
  - Modulenaam, helpstring en versie
  - Startpunt (entrypoint) voor commando's inclusief help
  - Startpunt voor SWI's inclusief tabel voor SWI-namen
  - Startpunt voor afhandelen service calls

Werkruimte

Elke module 1 word private werkruimte.
Doel: Opslaan pointer naar werkelijke werkruimte (geheugenblok)

- Gebruik bij initialisatie:
  1. Alloceer geheugenblok tijdens initialisatie (OS_Module 6)
  2. Bewaar pointer in private word met STR R2,[R12]
- Bij elke aanroep van module door OS:
  1. R12 wijst naar private word
  2. Ophalen pointer naar eigen werkruimte met LDR R12,[R12]
- Bij beëindigen (finalise) van module:
  1. Geen code nodig om werkruimte vrij te geven RISC OS doet dit al standaard

Module header formaat

Offset	Type	Inhoud word
&00	offset naar code	Starten van module (taal of applicatie)
&04	offset naar code	Initialisatie-code
&08	offset naar code	Finalise-code
&0C	offset naar code	Afhandelen van service calls
&10	offset naar string	Module titel
&14	offset naar string	Helpstring
&18	offset naar tabel	Tabel commando-namen en help
&1C	nummer	Basisnummer van SWI (optioneel)
&20	offset naar code	SWI-code (optioneel)
&24	offset naar tabel	SWI decodeer-tabel (optioneel)
&28	offset naar code	SWI decodeer-code (optioneel)
&2C	offset naar string	Bestandsnaam Messages-bestand (optioneel)
&30	offset naar word	Module-vlaggen (optioneel)

Starten module

Bij aanroepen

R0 = pointer naar commando string, inclusief modulenaam
R12 = pointer naar private word

Bij verlaten

Stopt niet tenzij er een fout plaatsvindt

Processor modus

USR-modus

Gebruik

Dit startpunt in module wordt gebruikt om een omgeving (BASIC) of wimptaak te starten

Initialisatie

Bij aanroepen

R10 = pointer naar vlaggen, via *RMReInit of OS_Module
R11 = I/O basis of nummer submodule
R12 = pointer naar private word, inhoud = 0
R13 = SVC-stackpointer (FD)

Bij verlaten

R7 - R11 en R13, modus en interrupts onveranderd

Processor modus

SVC-modus

Gebruik

Alloceer werkruimte/DA's, registreer filters/vectors
Bij foutmelding, zet V, R0 -> foutblok en eindig routine
Stop met MOV PC,R14 of LDMFD R13!,{PC}

Finalise

Bij aanroepen

R11 = nummer of submodule
R12 = pointer naar private word
R13 = SVC-stackpointer (FD)

Bij verlaten

R7 - R11 en R13, modus, interrupts onveranderd

Processor modus

SVC-modus

Gebruik

Geef buffers/DA's vrij
Zet V met R0 -> foutblok als module niet gestopt mag worden
Stop met MOV PC,R14 of LDMFD R13!,{PC}

Afhandelen service calls

Bij aanroepen

R1 = nummer service call
R12 = pointer naar private word
R13 = SVC-stackpointer (FD)

Bij verlaten

R1 = 0 als service call door module wordt afgehandeld

R0, R2 - R8 parameter registers

R12 mag worden gewijzigd

Processor modus

SVC-modus of IRQ-modus

Gebruik

Handel ongeïnteresseerde services calls snel af
Stop met MOV PC,R14 of LDMFD R13!,{PC}

Afhandelen service call

Vanaf RISC OS 4 gebruik additionele service-tabel:

- svc-routine onveranderd behalve:

Adres	Inhoud
svc - 4	Offset naar service-tabel
svc	MOV R0,R0 (magische instructie)
svc + 4...	Vervolg svc-routine

- service-tabel:

Adres	Inhoud
tabel + 0	Bit 0 aan dan R1->index in tabel
	Bit 0 uit dan R1 = nummer service
tabel + 4	Offset svc-routine
tabel + 8	1e geïnteresseerde servicenummer
tabel + 12	2e geïnteresseerde servicenummer
tabel + X	Afsluitende 0

Titel en helpstring

Titelstring

Alfanummerieke string afgesloten met 0

Mag geen spaties of control codes bevatten

Helpstring

Alfanummerieke string met afgesloten 0

Geen control codes behalve tab

Vorm: module_naam Tab[Tab] v.vv (DD MMM YYY)

Tabel commando's en help

Tabel bevat sleutelwoorden van commando's en/of pointers naar helptekst en commando code. Tabel is als volgt opgebouwd:

Sleutelwoord commando
ALIGN voor word aligned
Offset naar code gerekend vanaf begin module, of 0 geen code
Informatie word
Offset naar syntax melding of 0 standaard bericht
Offset naar help tekst of 0 geen helptekst

Code offset commando's

Bij aanroepen

R0 = pointer naar commando vlaggen
R1 = aantal parameters gevonden door OSCLI
R12 = pointer naar private word
R13 = SVC-stackpointer (FD)
R14 = terugkeer-adres

Bij verlaten

R0 = bij fout pointer naar foutblok, V gezet

R7 - R11 en R13 onveranderd

Processor modus

SVC-modus

Informatie word commando's

Informatie word bevat informatie over aantal vlaggen en commando-type:

Byte	Inhoud
0	Minimum aantal parameters (0-255)
1	OS_GSTrans selectie voor de 1e 8 parameters
2	Maximum aantal parameters (0-255)
3	Vlaggen

Byte 1 Bit gezet dan parameter via GSTrans, bit 0 parameter 1, bit 1 parameter 2 etc

Bit 31 = 1 commando is een FS-commando zoals *bye, *eject etc. OSCLI vindt dit commando pas als er geen normale variant bestaat voor dit commando

Informatie word commando's

Bit 30 = 1

Commando is configuratie-optie (*Status of *Configure) en R0 is als volgt:

R0 = 0 *Configure is zonder parameters aangeroepen, module drukt syntax af (OS_Write0)

R0 = 1 *Status is aangeroepen, module drukt ingestelde waarde af

Anders *Configure is aangeroepen met parameters R0 -> commando vlaggen

Commando vlaggen incorrect of bij fout, zet V en R0 als volgt:

R0 = 0 Foute configuratie-optie

R0 = 1 Geen nummerieke parameter meegegeven

R0 = 2 Parameter is te lang

R0 = 3 Teveel parameters

R0 > 3 R0 = pointer naar foutblok

4 november 2006 Hints, tips en trucs
Modules

Informatie word commando's

Bit 29 = 1

Offset naar helptekst wijst naar code inplaats van string. De code wordt als volgt aangeroepen:

R0 = pointer naar een buffer

R1 = bufferlengte

R1 - R6 en R12 mogen veranderd zijn

Als aan het eind R0 <> 0 dan R0 = pointer naar string met afsluitende 0. OS drukt vervolgens deze string af.

Basisnummer SWI

Met het basisnummer identificeert RISC OS SWI met module. Het is geen pointer maar een vaste waarde. Neem als bijvoorbeeld hourglass:

&406C0	+0	Hourglass_On
	+1	Hourglass_Off
	+2	Hourglass_Smash
	+3	Hourglass_Start
	+4	Hourglass_Percentage
	+5	Hourglass_LEDs
	+6	Hourglass_Colours

Het basisnummer is &406C0. Module kan in totaal 64 SWI's hebben.

SWI-code

Bij aanroepen

R0 - R8 registers met eventuele opties

R11 = interne SWI-nummer (0-63)

R12 = pointer naar private word

R13 = SVC-stackpointer (FD)

R14 = terugkeer-adres + eventuele vlaggen (26 bit)

Bij verlaten

R0 - R8 teruggegeven waarden, bij fout R0 -> foutblok, V gezet

R10 - R12 mogen veranderd zijn

Gebruik MOV PC,R14 (32 bit), MOVS PC,R14 (26 bit)

Processor modus

SVC-modus

SWI-code voorbeeld

```
.SWIstart
 LDR    R12,[R12]                          ;R12 -> werkruimte
 CMP    R11,#(switabel_eind - switabel)/4  ;vergelijk SWI-nummer met aantal in tabel
 ADDLO  PC,PC,R11,LSL#2                    ;in bereik dan spring naar juiste routine
 B      swi_onbekend                       ;niet in bereik dan kennen we hem ook niet
.switabel
 B      SWInr_0                            ;spring naar SWI routine 1
 B      SWInr_1
 ..............
 B      SWInr_n
.switabel_eind
.swi_onbekend
 STR    R14,[R13,#-4]!                     ;bewaar link op SVC-stack
 ADR    R0,errteken_onbekende_swi
 MOV    R1,#0                              ;gebruike interne messages
 MOV    R2,#0                              ;gebruik buffer messagetrans
 ADR    R4,modulenaam                      ;vervang %0 voor modulenaam
 SWI    "XMessagesTrans_ErrorLookup"       ;R0 -> foutblok
 CMPVC  R1,#1<<31                          ;set V als V ongezet
 LDR    PC,[R13],#4                        ;haal link van stack en stop routine
.errteken_onbekende_swi
 EQUD   &1E6                               ;foutnummer badswi van OS
 EQUD   "BadSWI"                           ;bericht-teken
 EQUB   0
 ALIGN
```


SWI decodeer-tabel

OS gebruikt SWI decodeer-tabel voor conversies tussen nummer en SWI-naam.
Formaat decodeer-tabel:

- SWI-groep prefix
- Naam van SWI 0
- ...
- Naam van SWI n
- Eindig met 0 byte

Decodeer-tabel hourglass met 2 SWI's

```
EQUS "Hourglass"
EQUB 0
EQUS "On"
EQUB 0
EQUS "Off"
EQUB 0
EQUB 0
```

Modules in BASIC Assembler

```
REM Dit programma maakt een module aan
name$ = "Module" : version$ = "0.01 (4 november 2006)" : helpmod$ = name$+CHR$9+version$
size% = 4095 : PROCassemble : SYS"OS_File",10,name$,&FFA,,code%,endcode% : END
DEFPROCassemble
 DIM code% size% : FOR pass% = 12 TO 14 STEP 2 :REM offset assembly
 P% = 0 : O% = code% : L% = O% + size%
 [OPT pass%
  EQUD 0 ;geen startcode
  EQUD initialise
  EQUD 0 ;geen finalise
  EQUD service_call_handler
  EQUD title
  EQUD helptitle
  EQUD 0 ; geen commando's
 .title
  EQUS name$
  EQUB 0
 .helptitle
  EQUS helpmod$
  EQUB 0
  ALIGN
 .initialise
  ..............
 .service_call_handler
  ..............
 ]
 NEXT pass% : endcode%=O% : ENDPROC
```