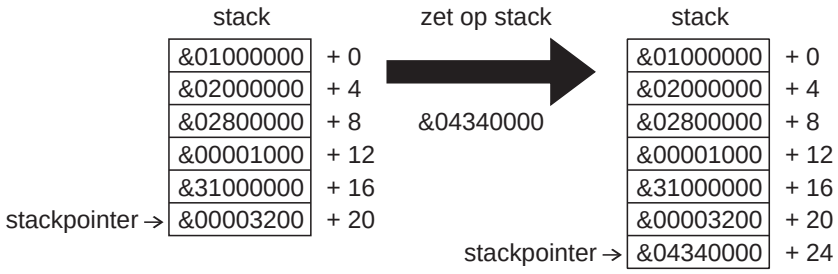
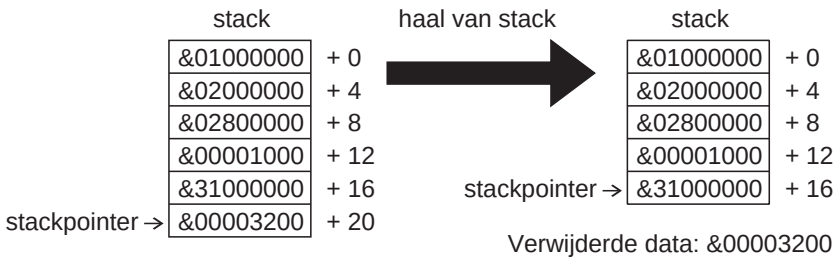
Inhoud

Computer stacks

Ophalen en wegzetten van data:



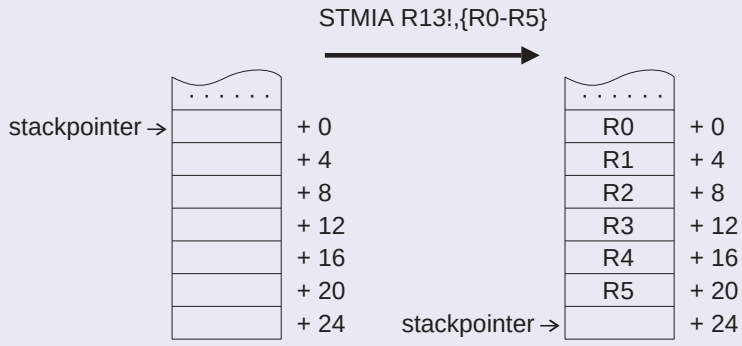


Type stacks

Er zijn 4 type computer stacks:

- FA: vol oplopende stack
- FD: vol aflopende stack
- EA: lege oplopende stack
- ED: lege aflopende stack

Lege oplopende (EA) stack



Stacks met LDM en STM

Registers op stack zetten:

STMDB (stack pointer)!,{register lijst}

Registers van stack halen:

LDMIA (stack pointer)!,{register lijst}

Voorbeelden

`STMFD R13!,{R1,R2,R4}`	Zet R1,R2 en R4 op FD stack
`LDMFD R13!,{R1,R2,R4}`	Laad R1,R2 en R4 van FD stack
`STMIA R13!,{R1-R7}`	Bewaar R1 tm R7 op EA stack
`LDMDA R13!,{R2-R4}`	Laad R2 tm R4 van FA stack
`STMDA R13!,{R0,R14}`	Bewaar R0 en R14 op ED stack

OPT instellingen

De OPT-parameters stellen de BASIC Assembler in, instellingen zijn:

- Bit 0: Print listing tijdens assembleren
- Bit 1: Rapporteer fouten, bijv. niet gevonden labels
- Bit 2: Offset assembly, als code op andere plek in het geheugen moet draaien dan waar het geassembleerd wordt
- Bit 3: Controle assemblage-geheugen, foutmelding als grens van het gealloceerde geheugen is bereikt
- Bit 4: Gebruik van 'nieuwe' instructies (MRS, MSR) mogelijk

7 oktober 2006
Stacks
BASIC Assembler (2)
Sub-routines
Utilities

OPT instellingen

	Nieuwe instr.	Geheugen-controle	Offset assembly	Rapport. fouten	Print listing
OPT 0	nee	nee	nee	nee	nee
OPT 1	nee	nee	nee	nee	ja
OPT 2	nee	nee	nee	ja	nee
OPT 3	nee	nee	nee	ja	ja
OPT 4	nee	nee	ja	nee	nee
OPT 5	nee	nee	ja	nee	ja
OPT 6	nee	nee	ja	ja	nee
OPT 7	nee	nee	ja	ja	ja
OPT 8	*nee*	*ja*	*nee*	*nee*	*nee*
OPT 9	*nee*	*ja*	*nee*	*nee*	*ja*
OPT 10	*nee*	*ja*	*nee*	*ja*	*nee*
OPT 11	*nee*	*ja*	*nee*	*ja*	*ja*
OPT 12	*nee*	*ja*	*ja*	*nee*	*nee*
OPT 13	*nee*	*ja*	*ja*	*nee*	*ja*
OPT 14	*nee*	*ja*	*ja*	*ja*	*nee*
OPT 15	*nee*	*ja*	*ja*	*ja*	*ja*
OPT 16	ja	nee	nee	nee	nee
⋮	⋮	⋮	⋮	⋮	⋮

Fout-controle

Dit geeft normaal een fout bij het assembleren:

```
 TEQ R0,#1
 BNE einde
 SWI 256+7 ;beep
 .einde
 MOV PC,R14
```

Door 2 maal te assembleren kan het wel:

```
FOR pass% = 0 TO 2 STEP 2
P% = code%
[ OPT pass%
 TEQ R0,#1
 BNE einde
 SWI 256+7 ;beep
 .einde
 MOV PC,R14
]
NEXT pass%
```


Geheugen-controle en Offset Assembly

```
size% = 255
DIM code size%
FOR pass% = 8 TO 10 STEP : REM Geen listing en geheugen-controle
P% = code% : L% = P% + size%
[ OPT pass%
 ;assembler code
]
NEXT pass%
```

Offset assembly: code uitvoeren op ander plaats in geheugen dan waar het geassembleerd wordt.

```
DIM code%
FOR pass% = 5 TO 7 STEP 2 : REM Print listing en gebruik offset assembly
P% = 0 : O% = code%
[ OPT pass%
; assembler code
]
NEXT pass%
```

Data in assembler

Assembler instructie	Alternatief	Functie
EQUB	DCB, =var%	reserveert een byte
EQUW	DCW	reserveert een 'halve word' (2 bytes)
EQUD	DCD	reserveert een word (4 bytes)
EQUS	=string$	reserveert een string

ALIGN maakt P% weer word-aligned

Spring met link-adres

Syntax:

BL <adres>

Na sprong bevat R14 adres van instructie na BL (plus de PSR bij 26 bit)

Voorbeeld

```
.hoofd_programma
        :
BL subroutine
        :
einde_programma
.subroutine
        :
MOV PC,R14
```

Bewaren link register en vlaggen

Bewaren link register:

```
STMDB R13!,{R14} ;aan begin van programma/routine
LDMIA R13!,{PC}  ;aan het eind van programma/routine
```

Herstellen link register met vlaggen:

26 bit	32 bit
`STMDB R13!,{R14}`	`STMDB R13!,{R14}`
`.........`	`MRS R14,CPSR`
`.........`	`STMDB R13!,{R14}`
`.........`	`.........`
`.........`	`LDMIA R13!,{R14}`
`.........`	`MSR CPSR_f,R14`
`LDMIA R13!,{PC}^`	`LDMIA R13!,{PC}`

Bewaren link register en vlaggen

26/32 bit neutrale code in hoofd- en sub-routines (ROS 3.1 of hoger)
Deze voorbeeld-routine behoudt ook inhoud van R2.

```
STMDB    R13!,{R2,R14}     ;bewaar originele registers op stack
MRS      R14,CPSR          ;CPSR in R14, NOP-instructie onder ARM2/3
TEQ      R0,R0             ;alleen in USR-modus om Z-vlag te zetten
TEQ      PC,PC             ;EQ als 32 bit NE als 26 bit
STREQ    R14,[R13,#-4]! of STMEQDB R13!,{R14} ;bewaar CPSR op stack
.................          ;de code van de routine
TEQ      R0,R0             ;alleen in USR-modus
TEQ      PC,PC
LDMNEIA  R13!,{R2,PC}^     ;einde in 26 bit modus, behoud registers
                           ;en zet originele PSR terug
LDR      R14,[R13],#4 of LDMIA R13!,{R14} ;haal CPSR op van stack
MSR      CPSR_f,R14        ;zet vlaggen terug in CPSR
LDMIA    R13!,{R2,PC}      ;einde routine en herstel inhoud registers
```

Utility eigenschappen

Een utility (&FFC) heeft de volgende eigenschappen:

- Code draait in RMA en moet plaats-onafhankelijk zijn
- Bij aanroepen code:
  - R0 = pointer naar aanroep van commando
  - R1 = pointer naar commando vlaggen
  - R12 = pointer naar werkruimte
  - R13 = pointer naar einde werkruimte
  - R14 = terugkeer-adres
- Werkruimte 1024 bytes
- Code draait in USR-modus, interrupts aan
- Gebruik SWI's met X optie, zelf fouten afhandelen
- Verlaat code met MOV PC,R14

Utilities in BASIC Assembler

```
REM Dit programma maakt een utility aan
copyright$ = "Dit is een utility"
file$      = "Utility"
size%      = 4095
PROCassemble
SYS"OS_File",10,file$,&FFC,,code%,endcode%
END
DEFPROCassemble
 DIM code% size%
 FOR pass% = 8 TO 10 STEP 2
 P% = code% : L% = P% + size%
 [OPT pass%
  ;Utility code
  ..............
  MOV  PC,R14
  EQUS copyright$
 ]
 NEXT pass%
 endcode%=P%
ENDPROC
```

Afhandelen van fouten

Afhandeling van foutmelding van SWI's:

- gebruik altijd de X-versie SWI
- bij een foutmelding van SWI:
  - is V-vlag gezet
  - R0 is pointer naar een foutblok:
    - R0+0 = foutnummer
    - R0+4 = pointer naar foutmelding, max 255 kars
- Gebruik na SWI: BVS fout_routine, MOVVS PC,R14 of LDMVS R13!,{PC} of iets dergelijks, in ieder geval moet je je code stoppen met V-vlag gezet en R0 pointer naar foutblok

Fatale foutmeldingen in eigen code:

- Maak eigen foutblok
- Zet V-vlag en eindig code met R0 pointer naar foutblok

7 oktober 2006
Stacks
BASIC Assembler (2)
Sub-routines
Utilities

Afhandelen van fouten

Voorbeeld fout-afhandeling van een SWI:

`STMDB`	`R13!,{R0,R14}`	Zet registers op stack
`.....`		Verdere code in routine
`SWI`	`"XOS_File"`	Voor SWI OS_File uit
`STRVS`	`R0,[R13,#0]`	Bij fout zet R0 op stack inplaats van originele R0
`LDMVSIA`	`R13!,{R0,PC}`	Bij fout eindig code en laat OS fout genereren
`.....`		Verdere code in routine
`LDMIA`	`R13!,{R0,PC}`	Eindig routine normaal en zet registers terug