Inhoud

Register 15 in instructies

R15 als verschillende operands in 26 bits mode:

- Als operand 1: alleen bits 2-25, programm counter beschikbaar
- Als operand 2: alle 32 bits, status-, interruptvlaggen, mode bits en programm counter
- Als doel register:
  - standaard: alleen bits 2-25, programm counter wordt gewijzigd
  - met S optie: bit 26-31, interrupt- en statusvlaggen worden ook gewijzigd

In 32 bits modus bevat R15 alleen de PC en is volledig beschikbaar/beschrijfbaar. Dus `MOVS PC,R14` niet gebruiken!

Pipeline en R15

Door pipelining bevat R15 niet het adres van de uitgevoerde instructie: Oftewel `MOV R15,PC` laat de processor 2 instructies (8 bytes) vooruit springen

Voorbeeld

```
&1000   ORRS  R15,PC,#1<<31   zet N-vlag en springt
&1004   SWIMI 256+7           als N-vlag gezet dan biep
&1008   MOV   PC,R14          eindigt routine
```


Wijzigen statusvlaggen in 26 bit modus

Via TEQP kunnen we de PSR wijzigen zonder de PC te wijzigen. Het resultaat van de EOR-operatie wordt weggeschreven in de PSR van R15.

Voorbeelden

```
TEQP R15,#%11       schakelt processor naar SVC-modus
TEQP R15,#1<<31     N-vlag aan, alle andere vlaggen uit
```

Zetten V-vlag, andere vlaggen ongewijzigd

```
MOV  R0,R15         R0 is PC+PSR
ORR  R0,R0,#1<<28   V-vlag wordt gezet in bewaarde PSR
TEQP R0,#0          resultaat EOR wordt opgeslagen in de PSR
```

Wijzigen status vlaggen in 32 bits modus

Via de instructies MSR en MRS kunnen de CPSR en SPSR gewijzigd worden

Voorbeelden

`MSR CPSR_csxf,R0`	kopieer R0 naar CPSR
`MSR SPSR_csxf,R0`	kopieer R0 naar SPSR
`MSR CPSR_f,R0`	kopieer statusvlaggen van R0 naar CPSR
`MSR CPSR_f,#1<<28`	zet alleen V-vlag van de statusvlaggen
`MRS R0,CPSR`	kopieer CPSR naar R0
`MRS R0,SPSR`	kopieer SPSR naar R0

In USR modus kan alleen de statusvlaggen van de PSR gewijzigd worden

Tussen geheugen en registers

Om data van en naar geheugen te kopiëren:

- Voor 1 register:
  - LDR: data vanuit geheugen in een register
  - STR: data vanuit register naar het geheugen
- Voor meerdere registers:
  - LDM: laden van data uit het geheugen in meerdere registers
  - STM: bewaren van meerdere registers in het geheugen

Indirecte adressering: Inhoud van register bepaalt geheugen-locatie

Pre-indexed adressering

Syntax:

LDR/STR <doel reg>, [base{,<offset>}]!

Base: register met startadres voor geheugenblok

Offset: positie in geheugenblok

- Een register
- Een constante
- Een verschoven register

Voorbeelden

`LDR R0,[R1,R2]`	Laad R0 van adres R1+R2
`STR R0,[R1,#-4]`	Bewaar R0 op adres R1-4
`LDR R2,[base,index LSL#2]`	Laad R2 van base+index*4
`LDR R3,[base,#4]!`	Laad R3 van base+4, base=base+4

Post-indexed adressering

Syntax:

LDR <doel reg>, [base],<offset>

Base: register met adres van geheugenblok

Offset: toevoeging aan base adres

- Een register
- Een constante
- Een verschoven register

Voorbeelden

`LDR R0,[R1],R2`	Laad R0 van adres in R1, R1=R1+R2
`STR R5,[R1],#4`	Bewaar R5 op adres in R1, R1=R1+4
`LDR R2,[R5],R2 LSL#2`	Laad R2 van adres in R5, R5=R5+R2*4
`STR R3,[R4],#-20`	Bewaar R3 op adres in R4, R4=R4-20

PC relatief en byte adressering

BASIC Assembler heeft een speciale instructie voor relatieve (directe) adressering:

LDR <doel reg>,adres

Voorbeelden

`LDR R0,&200`	Laad R0 van adres &200
`STR R5,tabel`	Bewaar R5 op adres waar label 'tabel' staat

Voor overdracht van bytes voegen we de B optie toe

Voorbeelden

`LDRB R0,[R1,R2 LSL#6]`	Laad byte van R1+R2*64
`STRB R0,[R5],#1`	Bewaar byte op R5, R5=R5+1

Meerdere registers

Syntax:

STM/LDM<opties> <base> {!}, <register lijst>

Opties

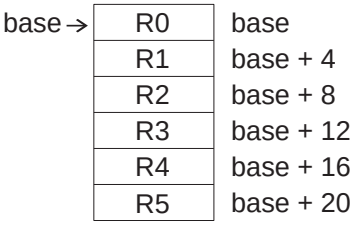
- I: verhogen (met 1 word) van adres na bewaren/laden van elk register
- D: verlagen (met 1 word) van adres na bewaren/laden van elk register
- A: aanpassen base adres na laden/wegschrijven register
- B: aanpassen base adres voor laden/wegschrijven register

Base: register met startadres

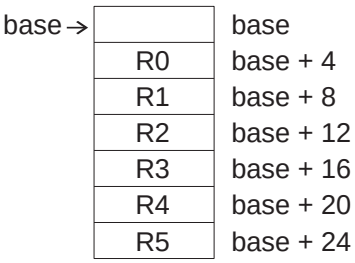
Registerlijst: R1,R4,R6, R1-R8 of R0,R1-R4,R7

STM voorbeelden

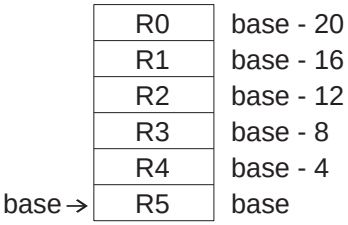
STMIA base,{R0-R5}



STMIB base,{R0-R5}



STMDA base,{R0-R5}



STMDB base,{R0-R5}