

Programma's en de stack

Processoren

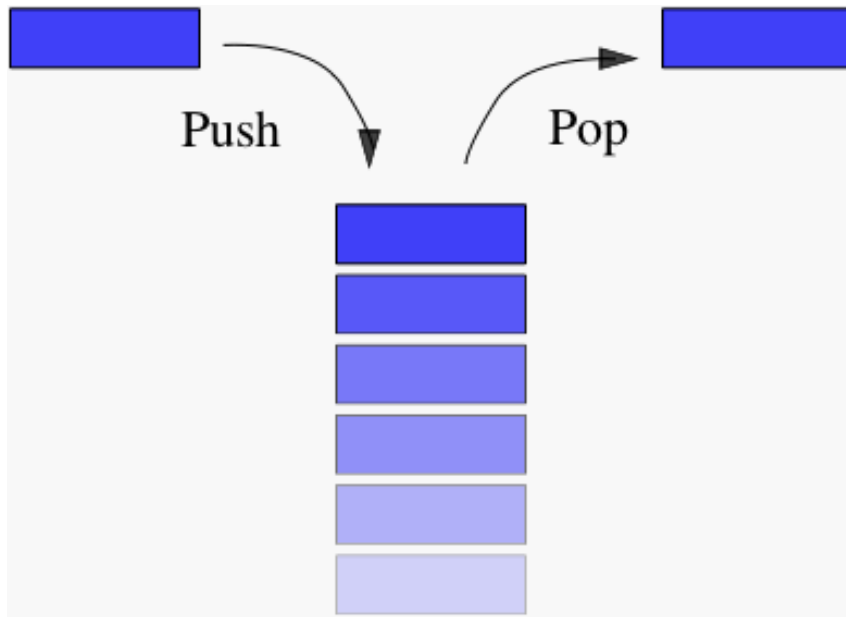
15 december 2014

Stackgeheugen (stapel)

- handige vorm van geheugen
→ bijna elk programma gebruikt het
- gebruiksmogelijkheden:
 - tussenresultaten opslaan
 - functies aanroepen en lokale variabelen

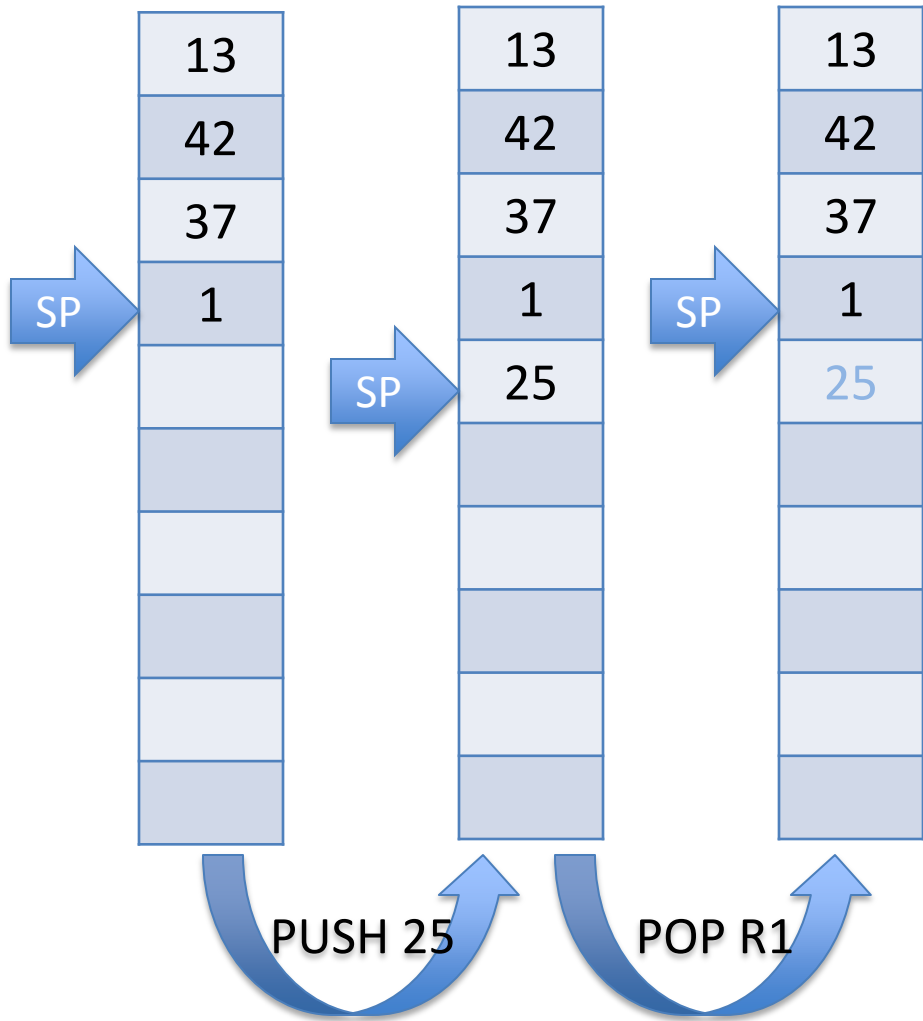


Basis stackoperaties



- PUSH = opslaan:
op de stack zetten
- POP = lezen:
bovenste gegeven
van de stack pakken
- volgorde dus: LIFO =
Last In, First Out

Implementatie van basisoperaties



- gereserveerd stackgeheugen
- speciaal register SP wijst naar top van stack
- PUSH = SP verlagen en op adres [SP] opslaan
- POP = [SP] lezen en SP verhogen

R1
25

Rekenen met de stack

- te weinig registers voor een berekening?
→ gebruik de stack voor tussenresultaten
- voorbeeld:
 - bereken $(a + b) * (c + d)$
 - gebruik alleen R1 en R2

Berekening: voorbeeld

	(bijna) Practicum-assembly
$(a + b) * (c + d)$	MOV a, R2 MOV b, R1 ADD R2, R1, R2 ; nu staat a+b in R2 PUSH R2 ; nu staat a+b op de stack MOV c, R2 MOV d, R3 ADD R2, R3, R2 ; nu staat c+d in R2 POP R1 ; nu staat a+b in R1 MUL R2, R1, R1 ; nu staat het resultaat in R1

Subroutine / Procedure / Functie

- Procedure in assembly:
 - een stuk code
 - meerdere keren uitvoeren
 - in verschillende situaties van (hoofd)programma
 - na einde: doorgaan met hoofdprogramma,
op **het adres** waar je voor de functieaanroep was



stack!

Functies

- Aanroepen:

- parameters opslaan
- terugkeeradres opslaan
- springen naar functie

(meestal) op de stack
of (soms) in registers

- Einde:

- resultaat opslaan
- terugkeeradres terughalen
- springen naar terugkeeradres

meestal in register

Functievoorbeeld: parameters in registers

hoofdprogramma			functie voor deling		
main:	MOV	<i>(dividend)</i> , R2	div:	ORf	R1, R1, R0
	MOV	<i>(divisor)</i> , R1		JMP.Z	error
	CALL	div		...	
	MOV	R1,	
				MOV	<i>(quotient)</i> , R1
				RET	

Functievoorbeeld: parameters op stapel

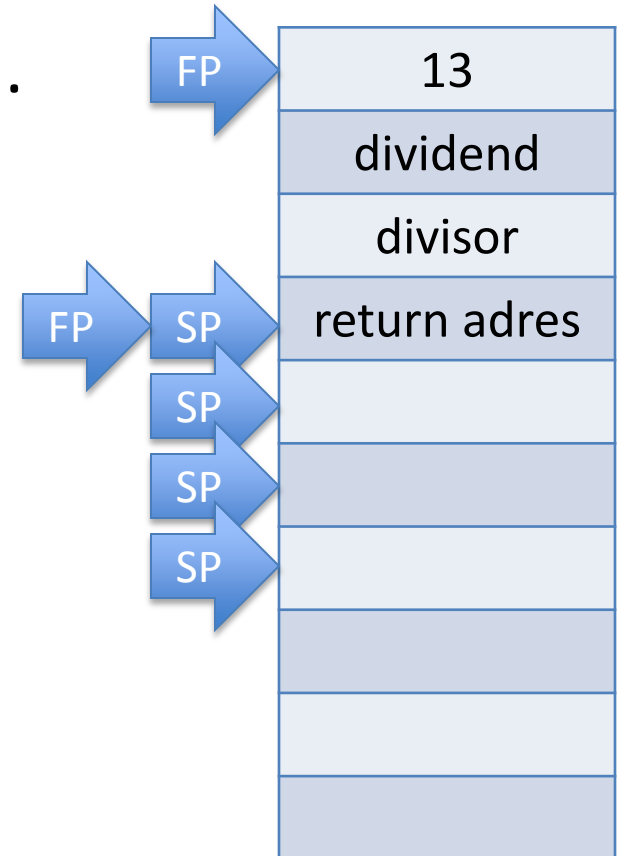
hoofdprogramma			functie voor deling		
main:	MOV	<i>(dividend)</i> , R1	div:	MOV	[SP+4],R1
	PUSH	R1		ORf	R1, R1, R0
	MOV	<i>(divisor)</i> , R5		JMP.Z	error
	PUSH	R5		...	
	CALL	div		...	
	ADD	2*4, SP, SP		MOV	<i>(quotient)</i> , R1
	MOV	R1, ...			

2 parameters die elk
4 bytes groot zijn

- meer code
- maar registers blijven vrij
- groot aantal parameters is mogelijk

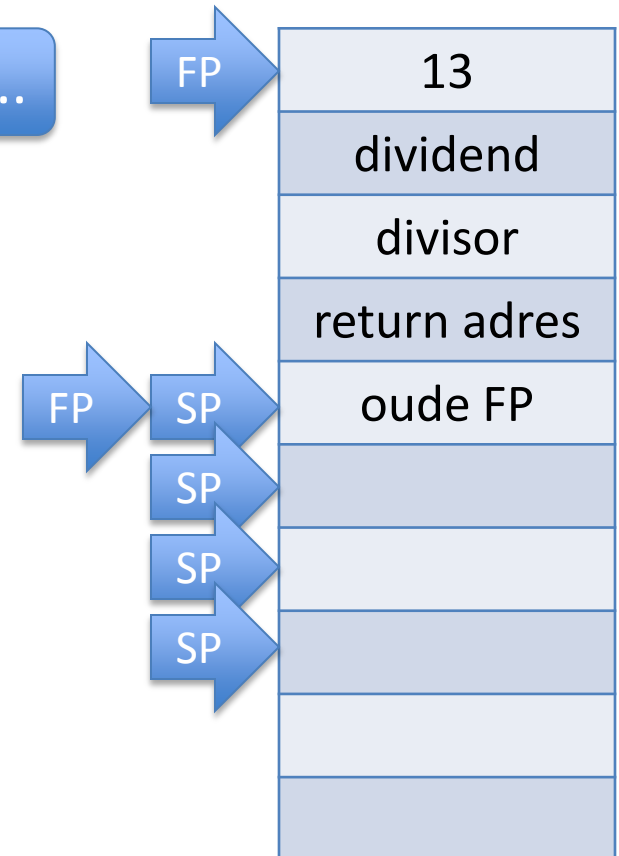
Parameters op de stack

- adres van parameter = $SP + 4, 8, \dots$
- Probleem: SP verandert als je tussenresultaten opslaat!
- Oplossing: nog een register: frame pointer (of base pointer)
 - zet FP op initiële waarde van SP
 - SP kan veranderen, FP niet
 - bij functies binnen functies: oude waarde van FP opslaan



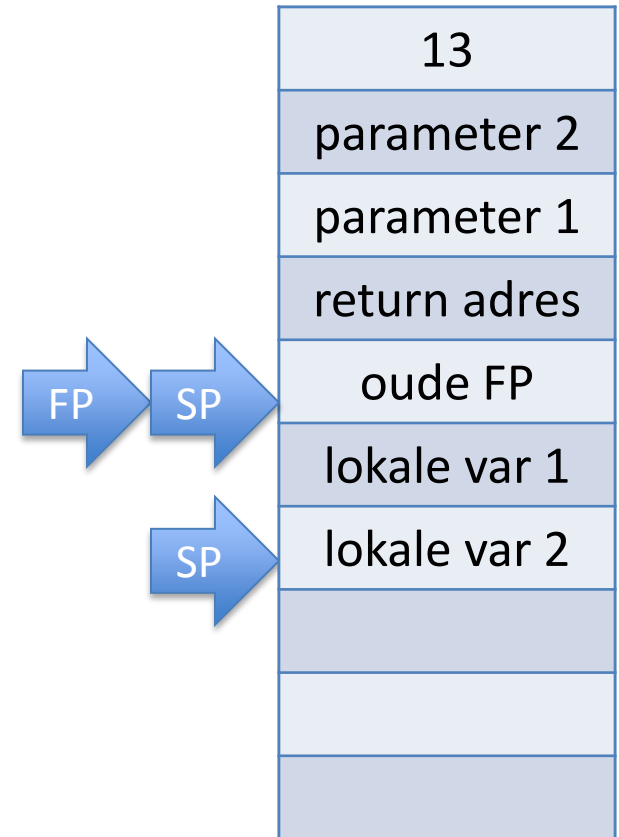
Parameters op de stack

- adres van parameter = $FP + 8, 12, \dots$
- Probleem: SP verandert als je tussenresultaten opslaat!
- Oplossing: nog een register: frame pointer (of base pointer)
 - zet FP op initiële waarde van SP
 - SP kan veranderen, FP niet
 - bij functies binnen functies: oude waarde van FP opslaan



Lokale variabelen van functie

- stack óók voor lokale variabelen
- Idee: reserveer de eerstvolgende plaatsen op stack voor lokale variabelen
- adres van lokale variabele = $FP - 4, 8, \dots$
- voordeel: recursie is mogelijk



Complete administratie van functie

Functiebegin

- oude FP opslaan
- $FP := SP$
- ruimte voor lokale variabelen reserveren

Functieeinde

- lokale variabelen vrijgeven:
 $SP := FP$
- oude FP ophalen
- terugkeeradres ophalen
- parameters vrijgeven
- springen naar terugkeeradres

Administratie op 80x86

	Functiebegin	Functieeinde
8086	func: PUSH BP MOV BP, SP ; BP := SP SUB SP, (<i>lokale variabelen</i>)	MOV SP, BP POP BP RET (<i>parameters</i>)
80286	func: ENTER (<i>lokale variabelen</i>)	LEAVE RET (<i>parameters</i>)

- voorzieningen voor functies:
 - gereserveerd register BP = frame pointer
 - speciale instructies ENTER, LEAVE

Een voorbeeldprogramma...

```
int fibo(int n) {  
    int temp;  
  
    if ( n <= 1 )  
        return 1;  
    temp = fibo(n-1);  
    return fibo(n-2) + temp;  
}
```


Dit voorbeeldprogramma in assembly

```
fibonacci:    PUSH    FP                # temp bevat nu fibo(n-1)
              MOVE    SP,FP          READ    [FP+8],R1
              ADD     -4,SP,SP        ADD     -2,R1,R1
              # [FP+8] = n en [FP-4] = temp.
              READ    [FP+8],R1      PUSH    R1
              ADD     f -1,R1,R1      CALL    fibo
              JUMP.G  endif          ADD     4,SP,SP
              MOVE    1,R1           READ    [FP-4],R2
              MOVE    FP,SP          ADD     R1,R2,R1
              POP     FP              # R1 = het resultaat van fibo
              RET                    MOVE    FP, SP
endif:        PUSH    R1              POP     FP
              CALL    fibo            RET
              ADD     4,SP,SP
              WRITE   R1,[FP-4]
```