Stapelgeheugen

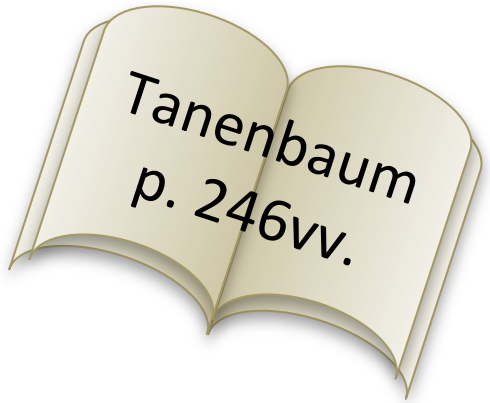
Processoren

6 maart 2012

Programma van komende week

- 7 maart 8.45: extra vragenuur over HADES
- 13 maart 8.45: hoorcollege vervalt,
  maar werkcolleges vinden plaats.

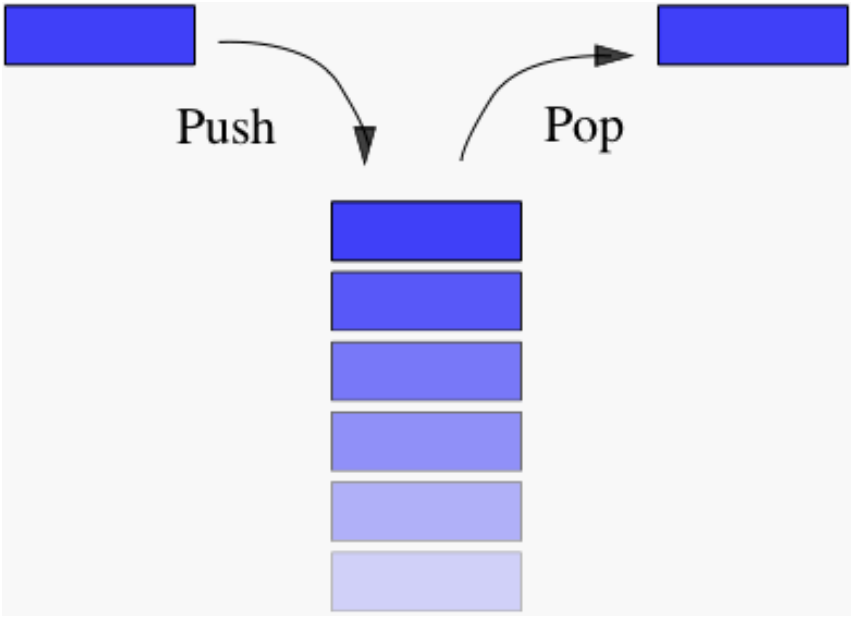
Stapelgeheugen



- handige vorm van geheugen
  → bijna elk programma gebruikt het

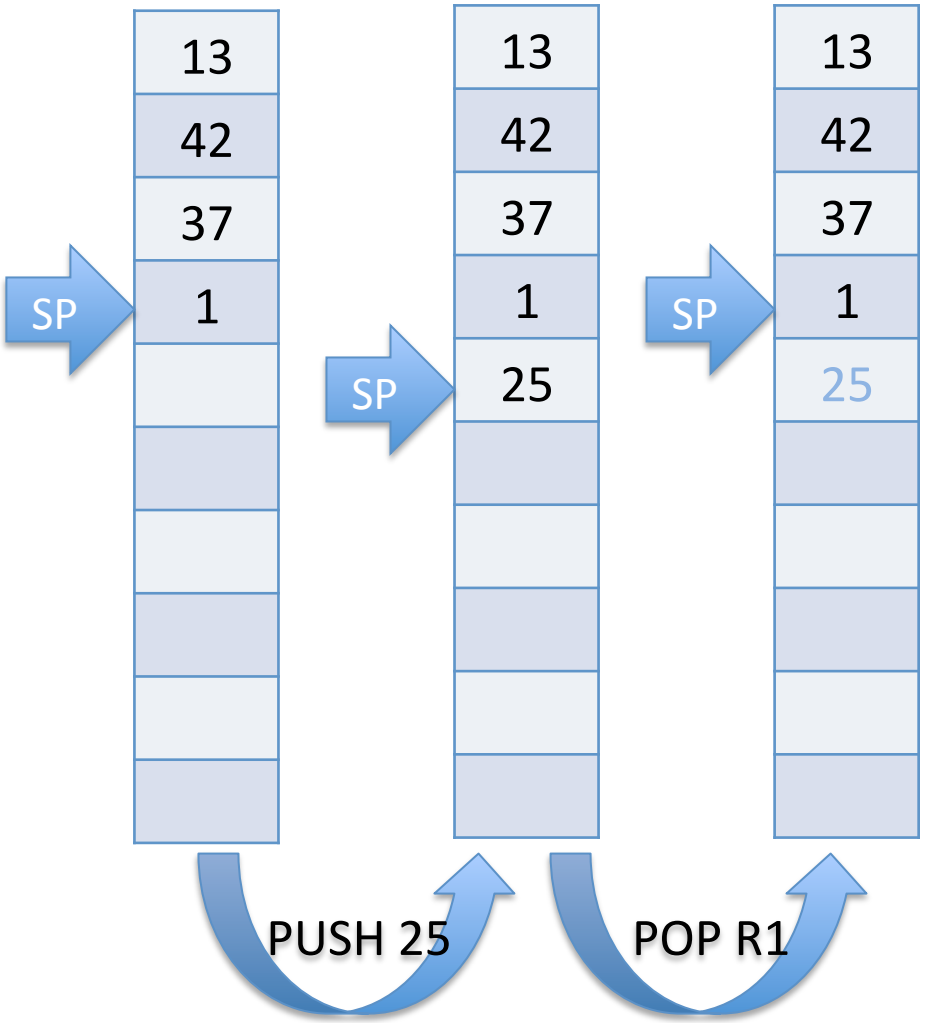
- gebruiksmogelijkheden:
  – tussenresultaten opslaan
  – functies aanroepen en lokale variabelen

Basisoperaties van stapel



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

implementatie van basisoperaties



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

R1
25

Rekenen met de stapel

- te weinig registers voor een berekening?
  → gebruik de stapel 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 stapel
	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

stapel!

Functies

- Aanroepen:
  - parameters opslaan
  - terugkeeradres opslaan
  - springen naar functie
- Einde:
  - resultaat opslaan
  - terugkeeradres terughalen
  - springen naar terugkeeradres

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

meestal in register

Functievoorbeeld: parameters in registers

hoofdprogramma			**functie voor divisie**		
main:	MOV	*(dividend)*, R2	div:	OR	R1, R1, R0
	MOV	*(divisor)*, R1		J.Z	error
	CALL	div		...	
	MOV	R1, ...		...	
				MOV	*(quotient)*, R1
				RET	

Functievoorbeeld: parameters op stapel

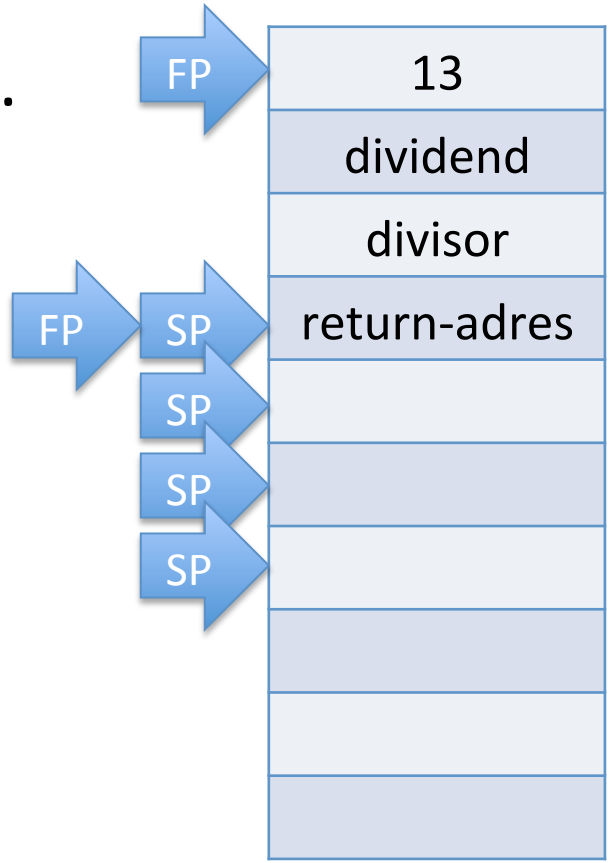
hoofdprogramma			functie voor divisie		
main:	MOV	*(dividend)*, R1	div:	OR	R1, R1, R0
	PUSH	R1		J.Z	error
	MOV	*(divisor)*, R5		...	
	PUSH	R5		...	
	CALL	div		MOV	*(quotient)*, R1
	ADD	2*4, SP, SP		RET	
	MOV	R1, ...			

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

Parameters op de stapel

- adres van parameter = SP + 4, 8, ...
- Probleem: SP verandert als je tussenresultaten opslaat!

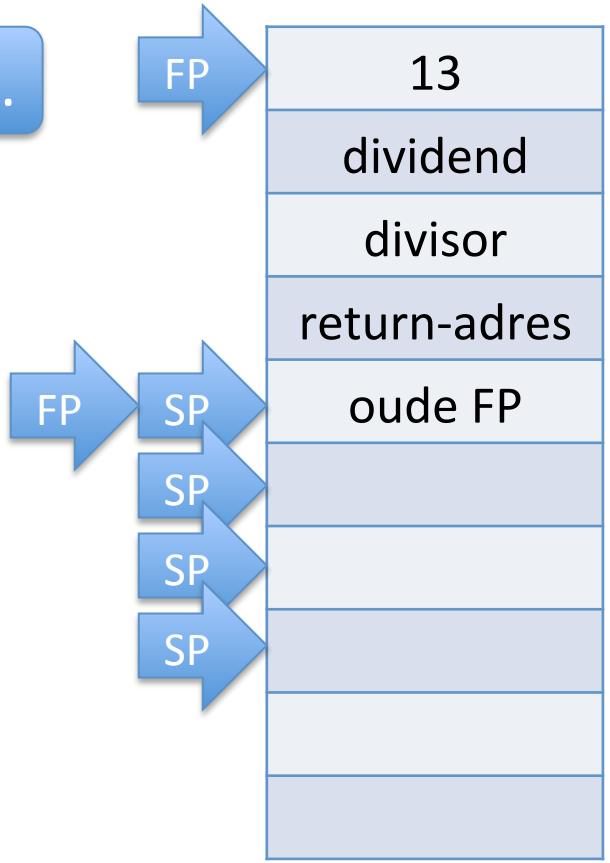
- Oplossing: nog een register: frame pointer (of base pointer)
  - zet FP op initiële waarde van SP
  - SP kan veranderen, FP niet
  - bij functies binnen functies: oude waarde van FP opslaan



Parameters op de stapel

- adres van parameter = FP + 8, 12, ...
- Probleem: SP verandert als je tussenresultaten opslaat!

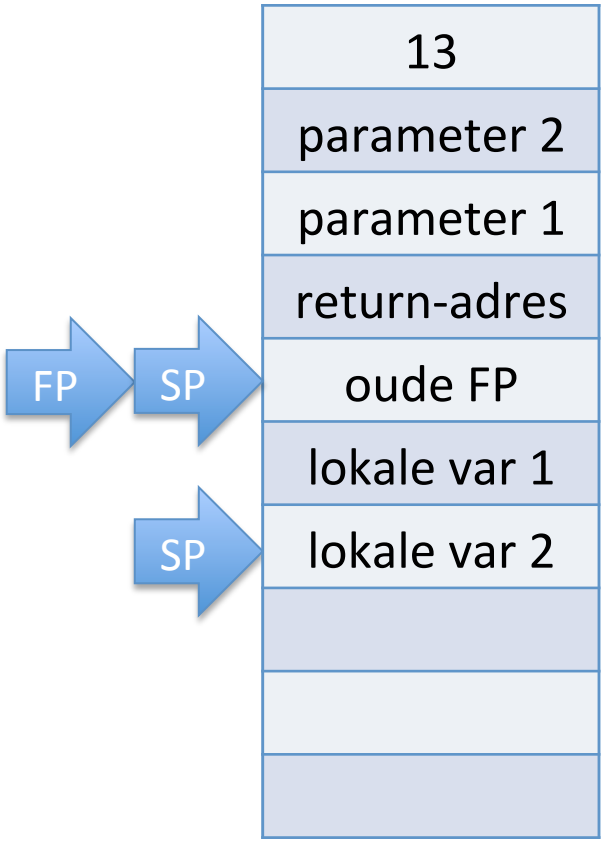
- Oplossing: nog een register: frame pointer (of base pointer)
  - zet FP op initiële waarde van SP
  - SP kan veranderen, FP niet
  - bij functies binnen functies: oude waarde van FP opslaan



Lokale variabelen van functie

- stapel óók voor lokale variabelen
- Idee: reserveer de eerstvolgende plaatsen op stapel voor lokale variabelen
- adres van lokale variabele = FP – 4, 8, ...

- 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, *(lokale variabelen)*	MOV SP, BP POP BP RET *(parameters)*
80286	func: ENTER *(lokale variabelen)*	LEAVE RET *(parameters)*

- 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

```
fibo:   PUSH    FP
        MOV     SP, FP
        ADD     –4, FP, FP
        # nu is [FP+8] = n en [FP–4] = temp.
        READ    [FP+8], R1
        ADD     –1, R1, R1
        J.G     endif
        LOAD    1, R1
        MOV     FP, SP
        POP     FP
        RET
endif:  PUSH    R1
        CALL    fibo
        ADD     4, SP, SP
        WRITE   R1, [FP–4]
        READ    [FP+8], R1
        ADD     –2, R1, R1
        PUSH    R1
        CALL    fibo
        ADD     4, SP, SP
        READ    [FP–4], R2
        ADD     R1, R2, R1
        # nu is R1 = het resultaat van fibo
        MOV     FP, SP
        POP     BP
        RET
```

Hoorcollege gemist?

- Bekijk de video-opname van vorig jaar!

http://icto.science.ru.nl/Podcasts/2011-03-15/ipc010-Deel_1_Processen_en_Processoren-ipod.m4v