Procedure Calls

- Hardware – Software interface

Met dank aan Wouter Koolen-Wijkstra

Doel: Begrip van de implementatie van procedure calls

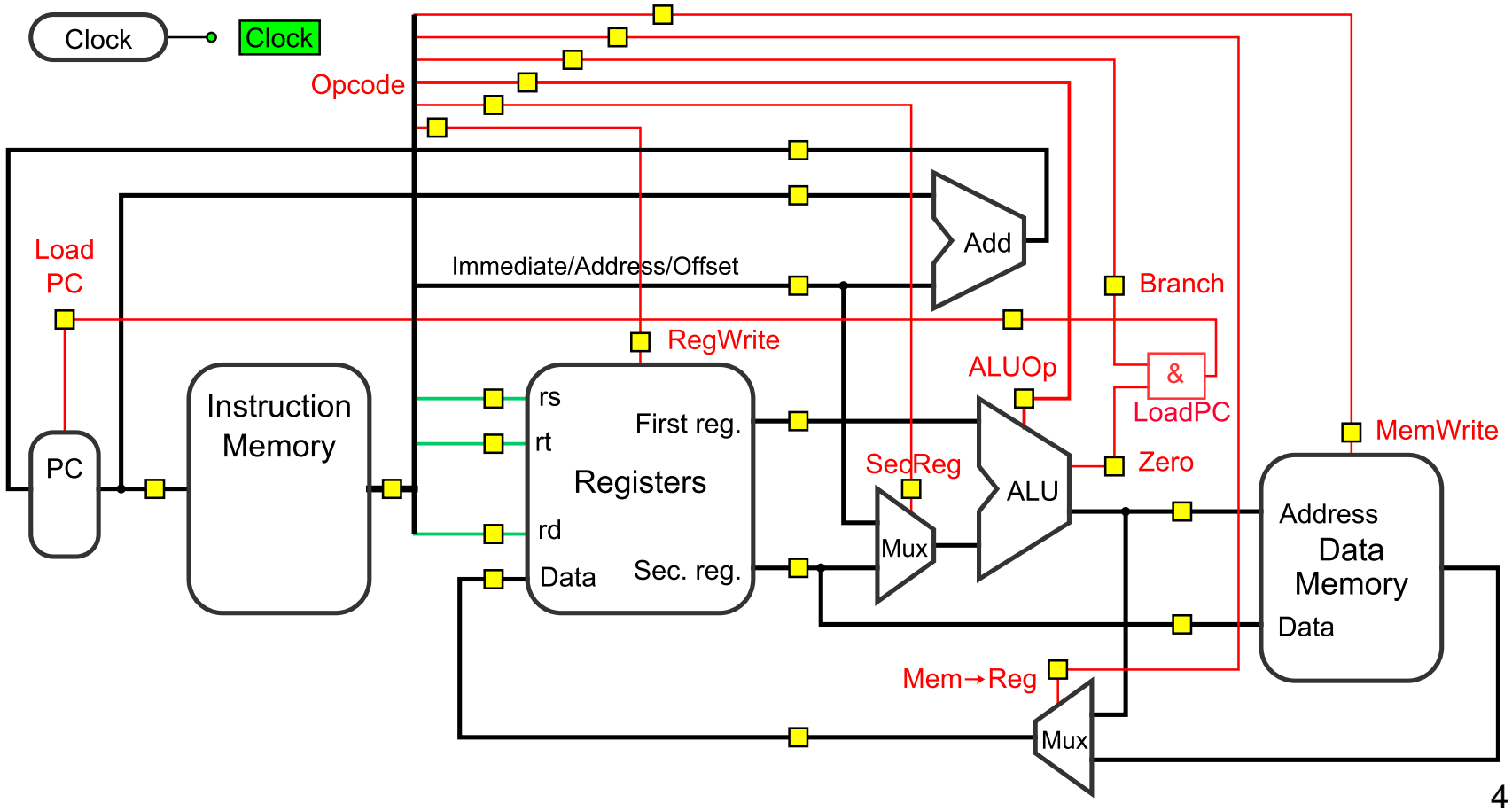
- Recapitulatie:
  - Harvard machine
  - Procedure
  - Stack
- Uitbreiding hardware & instructiset
- Stappen in een procedure aanroep
- Interface & contract

Recapitulatie

- Harvard machine
- Procedure
- Stacks

Recapitulatie Harvard machine

16 bit Harvard Architecture



Recapitulatie procedure

Procedure = functie = subroutine = `static` methode

(r1, . . . , rk) = f(a1, . . . , an)

- "Gewenst gedrag"
  - Bereken waarden (registers)
  - Lees/schrijf geheugen, . . .
- Geparameteriseerd
- Modulair
  - Inzetbaar op meerdere plaatsen in het programma
  - Behoudt toestand voor aanroep zover mogelijk
- Hiërarchisch

Recapitulatie Stack

Datastructuur met operaties:

- push(x) Plaats item bovenop de stack
- v = top() Bekijk bovenste item
- pop() Verwijder bovenste item van de stack

Recapitulatie Stack

Datastructuur met operaties:

- push(x) Plaats item bovenop de stack
- v = top() Bekijk bovenste item
- pop() Verwijder bovenste item van de stack

Een implementatie:

- `M` Array van items
- `i` Index van bovenste item
- push(x) `i = i+1; M[i] = x`
- v = top() `v = M[i]`
- pop() `i = i-1`

Recapitulatie Stack

Datastructuur met operaties:

- push(x) Plaats item bovenop de stack
- v = top() Bekijk bovenste item
- pop() Verwijder bovenste item van de stack

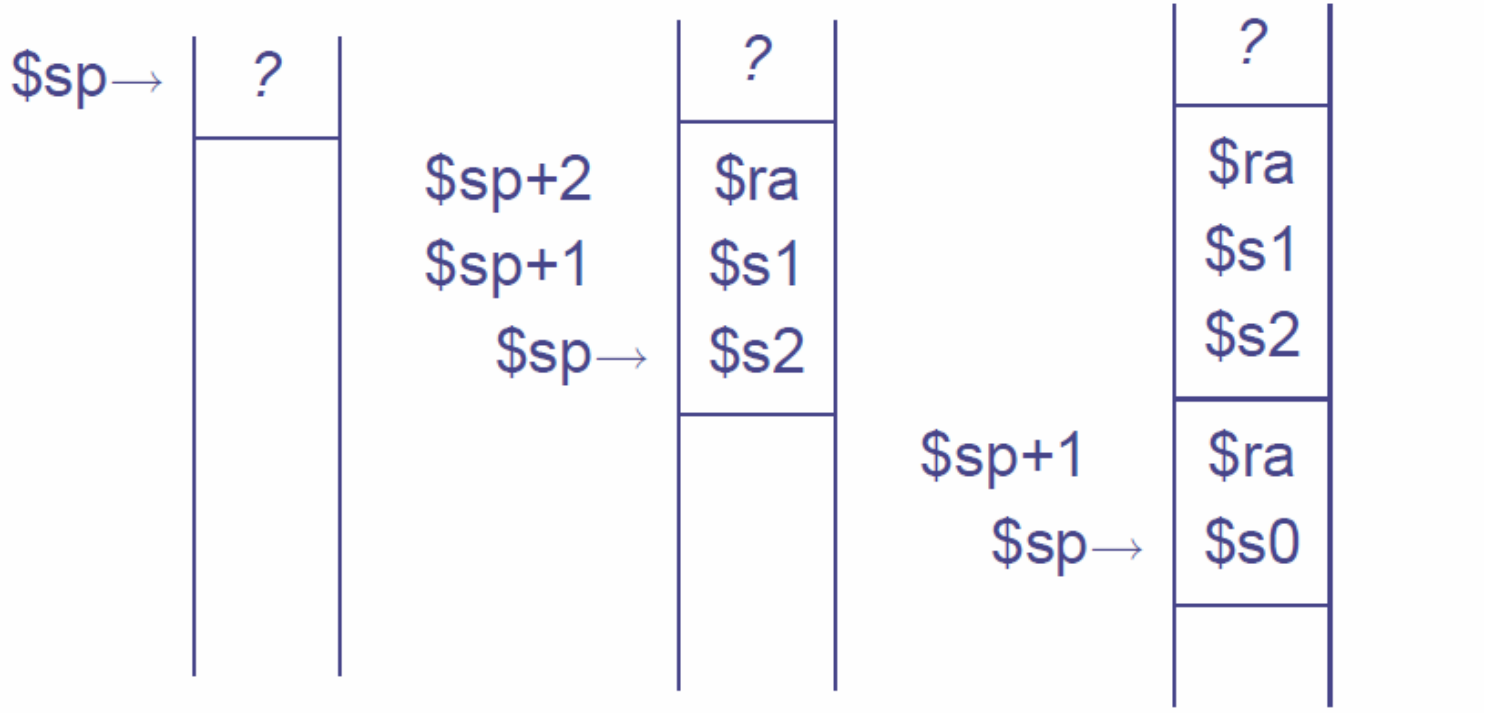
Een implementatie:

- `M` Array van items
- `i` Index van bovenste item
- push(x) `i = i+1; M[i] = x`
- v = top() `v = M[i]`
- pop() `i = i-1`

Wat gebeurt er als `+1` en `-1` worden verwisseld?

Stack frames

Stacks groeien (historisch) van hoge naar lage adressen

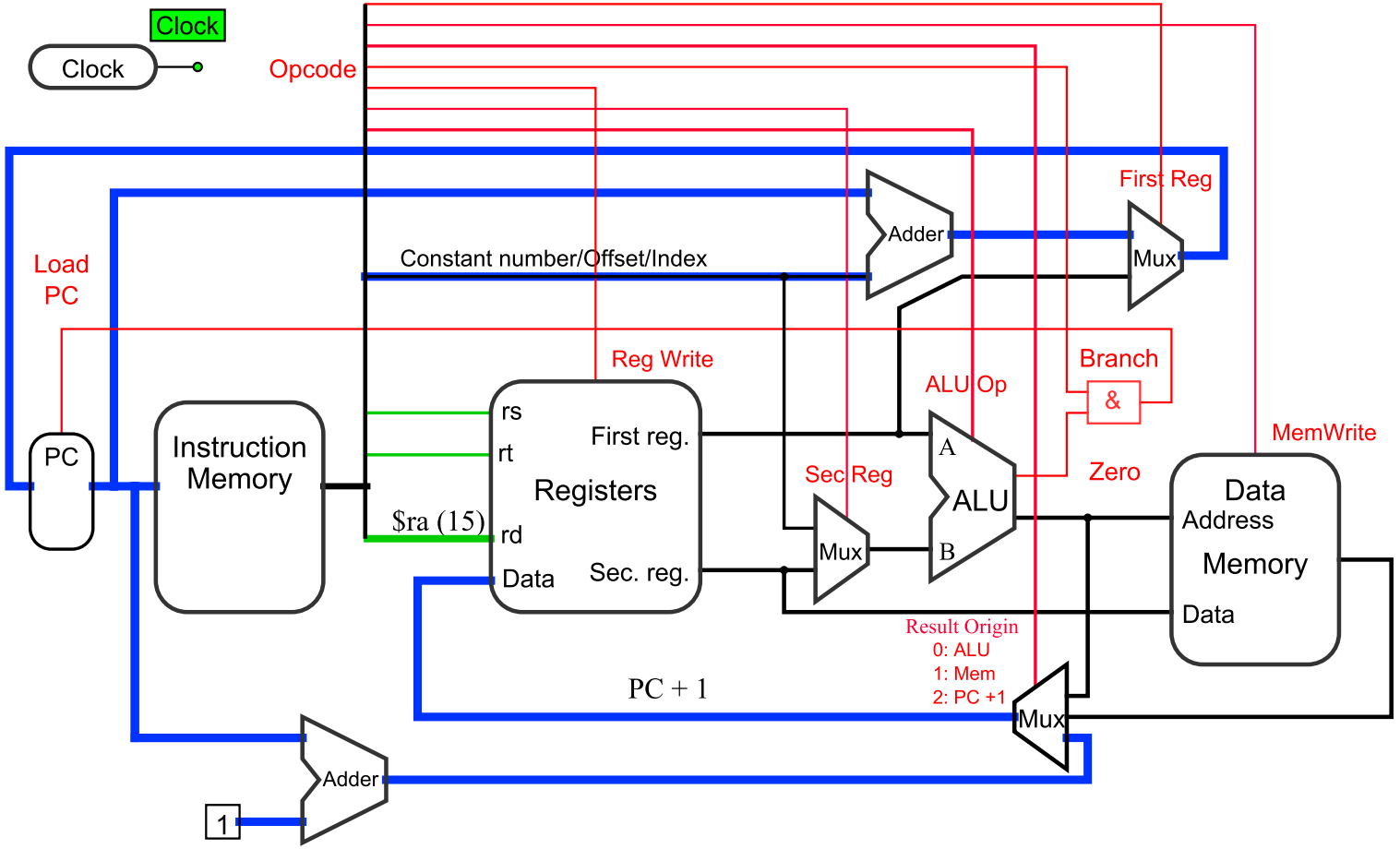


Benodigdheden

Doel: Maak de 16 bit Harvard geschikt voor procedures d.m.v. en twee extra instructies en een minimale uitbreiding van de hardware

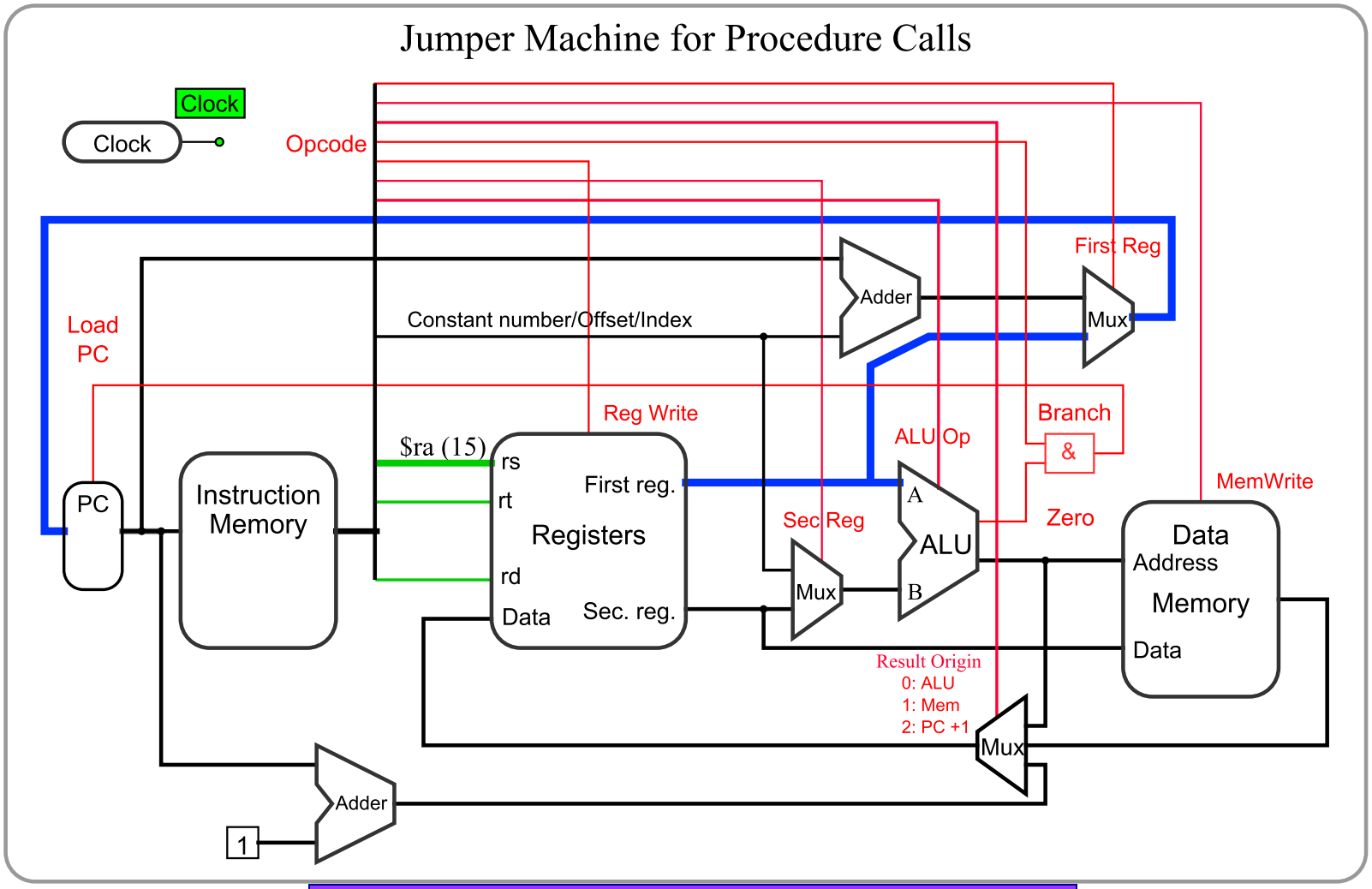
- Aanroep procedure
  - **Instructie: JSR (Jump to SubRoutine)**
  - Transfer PC naar register
  - *Haak in op writeback*
- Terugkeer procedure naar aanroeper
  - **Instructie: RETURN**
  - Transfer register naar PC
  - *Haak in op branch mechanisme*

Jumper Machine for Procedure Calls



JSR (Jump to SubRoutine)

PC ← PC + Offset	# Zelfde als bij BRanch Always
$ra ← PC + 1	# $ra onthoudt terugkeeradres + 1



RETURN from subroutine

PC ← $ra # programma keert terug net naar plek net na aanroep

Stappenplan uitvoeren procedure:

1. Plaats argumenten op een plek zodat ze toegankelijk zijn voor de procedure.
2. Bewaar de toestand van de aanroeper.
3. Draag de uitvoering van het programma over aan de procedure.
4. Voer de procedure uit.
5. Plaats de resultaten op een plek zodat ze toegankelijk zijn voor de aanroeper.
6. Draag de uitvoering van het programma weer over aan de aanroeper, nét na de aanroep.

Implementatie stappenplan

1. Sla argumenten op in daarvoor bestemde registers $arg1 …$arg4.
2. Bewaar het adres van de eerstvolgende instructie ná terugkeer van de procedure in register $ra.
3. Voer de instructie Jump SubRoutine (JSR) uit.
4. Voer de instructies die bij de procedure horen uit.
5. Sla de resultaten van de functieaanroep op in registers $val1 en $val2.
6. Voer instructie RETURN $ra uit.

1. *Plaats argumenten op een plek zodat ze toegankelijk zijn voor de procedure.*
2. *Bewaar de toestand van de aanroeper.*
3. *Draag de uitvoering van het programma over aan de procedure.*
4. *Voer de procedure uit.*
5. *Plaats de resultaten op een plek zodat ze toegankelijk zijn voor de aanroeper.*
6. *Draag de uitvoering van het programma weer over aan de aanroeper, nét na de aanroep.*

Voorbeeld 1:

```
int func(int j, int k, int l, int m)
 {
    int f;
    f = (j+k) - (l+m);
    return f;
 }

 v = func( 10, 20, 3, 4 );
```

```
# main
LOADI $arg1, 10              # step 1 argumenten zijn toegankelijk
LOADI $arg2, 20              # step 1
LOADI $arg3, 3               # step 1
LOADI $arg4, 4               # step 1
JSR func                     # step 2 & 3 stores PC + 1 into register $ra and jumps to func.
HALT                         # the result is now in $val1.

func:   #---------------------  procedure func ----------------------------------
ADD $tmp1, $arg1, $arg2      # step 4 compute j + k and store in temporary register $temp1.
ADD $tmp2, $arg3, $arg4      # step 4 compute l + m.
SUB $tmp3, $tmp1, $tmp2      # step 4 compute f = (j+k) - (l+m).
COPY $val1, $tmp3            # step 5 transfer result f to register $val1; resultaat toegangkelijk
RETURN $ra                   # step 6 return to caller.
```

Nodig voor dit voorbeeld:

- $arg1..$arg4 # vier argument registers om parameters in op te slaan.
- $val1 # register bestemd voor return values.
- $ra # hier wordt het terugkeeradres in opgeslagen.
- $tmp1..$tmp3 # registers voor tijdelijk gebruik

Contract voor gebruik van de 16 registers

Index	Abbreviation	Function	Preserved
0	`$zero`	Always zero	implicit
1. .2	`$val1..$val2`	Return values	no
3. .6	`$arg1..$arg4`	Procedure parameters	no
7. .10	`$tmp1..$tmp4`	Temporary	no
11. .13	`$s1...$s3`	Saved registers	yes
14	`$sp`	Stack pointer	yes
15	`$ra`	Return address	yes

Voorbeeld 2: Registers op stack bewaren

```
.data MyData : REGISTERS
@include "register_constantsLeaf.wasm"
# $s1 = 0x001E; $s2 = 0x0028; $s3 = 0x0032
# these values must be saved and restored

.code MyCode : REGSTACK, MyData
# this is the main program
# v = leaf_example(10,20,3,4);
LOADI $arg1, 10
LOADI $arg2, 20
LOADI $arg3, 3
LOADI $arg4, 4
JSR leaf_example # stores PC in register $ra
HALT # the result is now in $val1
```

```
#---------- procedure leaf_example ----------
leaf_example:
SUBI $sp, $sp, 3        # make space for 3 items on stack
SW $s1, 2, $sp          # save $s1 for use afterward
SW $s2, 1, $sp          # save $s2 for use afterward
SW $s3, 0, $sp          # save $s3 for use afterward
ADD $s1, $arg1, $arg2    # compute g + h
ADD $s2, $arg3, $arg4   # compute i + j
SUB $s3, $s1, $s2       # compute (g+h) - (i+j)
COPY $val1, $s3         # transfer result to return value reg.
LW $s1, 0, $sp          # restore $s1
LW $s2, 1, $sp          # restore $s2
LW $s3, 2, $sp          # restore $s3
ADDI $sp, $sp, 3        # remove 3 items from top stack
RETURN $ra              # return to caller
```

Voorbeeld 3: Procedure die procedure aanroept

ADD5(j, k, m, n, o) = Add3(Add3(j, k, m), n, o)