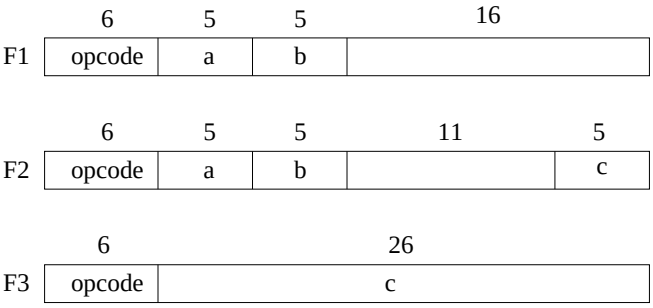
Eenvoudige RISC masjien

- Geheue bestaan uit 32-bis woorde, adresseerbaar as grepe (8 bisse per greep)
- Registers (32, elk 32 bisse wyd, R0–R31)
- Register R0 bevat altyd die waarde 0
- Instruksies om te spring na subroetines stoor die terugkeeradres in R31
- Alle instruksies beslaan 32 bisse

Instruksieformate



Voorbeelde:

```
F1: ADD a, b, c   (R.a := R.b + R.c)
F2: ADDI a, b, c   (R.a := R.b + c)
F3: JSR c   (save PC in R31, then jump
             to absolute address c)
```


Register instruksies (F1)

```
ADD  a, b, c     R.a := R.b + R.c
SUB  a, b, c     R.a := R.b - R.c
MUL  a, b, c     R.a := R.b * R.c
DIV  a, b, c     R.a := R.b DIV R.c
MOD  a, b, c     R.a := R.b MOD R.c
CMP  a, b, c     R.a := R.b - R.c
CHK  a, c        0 <= R.a < R.c
AND  a, b, c     R.a := R.b & R.c
OR   a, b, c     R.a := R.b OR R.c
LSH  a, b, c     R.a := LogicShift(R.b, R.c)
ASH  a, b, c     R.a := ArithShift(R.b, R.c)
```

Positiewe waarde R.c beteken skuif na links (negatief—skuif na regs) in LSH en ASH instruksies.

Register instruksies (F2)

Operand c direk gespesifiseer as deel van die volgende instruksies:

```
ADDI  a, b, c     R.a := R.b + c
SUBI  a, b, c     R.a := R.b - c
MULI  a, b, c     R.a := R.b * c
DIVI  a, b, c     R.a := R.b DIV c
MODI  a, b, c     R.a := R.b MOD c
CMPI  a, b, c     R.a := R.b - c
CHKI  a, c        0 <= R.a < c
ANDI  a, b, c     R.a := R.b & c
ORI   a, b, c     R.a := R.b OR c
LSHI  a, b, c     R.a := LogicShift(R.b, c)
ASHI  a, b, c     R.a := ArithShift(R.b, c)
```

Geheueverwysings-instruksies (F1)

```
LDW  a, b, c   R.a := Mem[R.b + c] (load word)
LDB  a, b, c   R.a := Mem[R.b + c] (load byte)
STW  a, b, c   Mem[R.b + c] := R.a (store word)
STB  a, b, c   Mem[R.b + c] := R.a (store byte)
POP  a, b, c   R.a := Mem[R.b]; R.b := R.b + c
PSH  a, b, c   R.b := R.b - c; Mem[R.b] := R.a
```


Instruksies vir kontrolevloei (F1)

Adresse in die volgende instruksies is PC-relatief:

```
BEQ  a, c   Branch to c if R.a = 0
BNE  a, c   Branch to c if R.a # 0
BLT  a, c   Branch to c if R.a < 0
BGE  a, c   Branch to c if R.a >= 0
BGT  a, c   Branch to c if R.a > 0
BLE  a, c   Branch to c if R.a <= 0
BSR  c      Save PC in R31, then branch to c)
```

Instruksies met absolute adresse:

```
JSR  c   Save PC in R31, then jump to c
         (F3, address absolute)
RET  c   Jump to address R.c
         (F2, address absolute)
```


Interpreter structure

```
MODULE RISC;
CONST MemSize = 4096; (* bytes *)
  ADD = 0; SUB = 1; ...
VAR PC, IR: LONGINT;
  R*: ARRAY 32 of LONGINT;
  M*: ARRAY MemSize DIV 4 OF LONGINT;
PROCEDURE Execute*(pc0: LONGINT);
  VAR opc, a, b, c, nxt: LONGINT;
BEGIN
  LOOP ...
  CASE opc OF
    ADD, ADDI: R[a] := R[b] + c
  | SUB, SUBI: ...

  END; PC := nxt
END Execute;

PROCEDURE Load* (* load code into memory *)

BEGIN
END RISC.
```


Opmerkings oor interpreteerder

- Instruksies RD, WRD, WRH en WRL word verskaf om eenvoudige toevoer en afvoer moontlik te maak. Sulke instruksies word gewoonlik nie ondersteun deur regte verwerkers nie.
- Die instruksies LDB en STB gaan nie regtig gebruik word nie omdat Oberon-0 nie die hantering van karakters ondersteun nie.
- Instruksies PSH en POP ondersteun hantering van prosedures met parameters.
- Instruksies CHK en CHKI word gebruik om te verseker dat indekse vir skikkings binne grense is.

Kode vir toekenningsopdragte

u := x + y * z

Adresse van veranderlikes word aangedui deur die naam van die veranderlike in onderstaande kode:

```
LDW 1,0,x
LDW 2,0,y
LDW 3,0,z
MUL 2,2,3
ADD 1,1,2
STW 1,0,u
```


Effektiewe kode

Oneffektiewe, maar korrekte kode vir `x := x + 1`:

```
LDW   1,0,x   R1 := x
ADDI  2,0,1   R2 := 1
ADD   1,1,2   R1 := R1 + R2
STW   1,0,x   x := R1
```

Meer effektiewe kode:

```
LDW   1,0,x
ADDI  1,1,1
STW   1,0,x
```


Toekenning van registers

- Gebruik een woord (32 bisse) om tred te hou van registers wat in gebruik is. (Byvoorbeeld, stel bis 3 = 1 as R3 in gebruik is.)
- Wanneer register R3 se waarde nie langer benodig word nie, stel bis 3 = 0 om aan te dui dat die register weer beskikbaar is.

Indeksering

- Laai adres van vektor X in register (byvoorbeeld R1)
- Laai waarde van indeks in register (byvoorbeeld R2)
- Grootte van element is beskikbaar tydens vertaaltyd (konstante `Size`)
- Formule: adr(X[k]) = adr(X) + k * Grootte van element

Kode om adres van element te bereken:

```
LDW   1,adr(X)   (laai adres van X)
LDW   2,0,index
MULI  2,2,Size   R2 := R2 * Size
ADD   1,1,2      R1 := R1 + 2R
```

CMP instruksie

- Spesiale instruksie om twee registers te vergelyk deur aftrekking
- Verskil van SUB-instruksie: oorvloei kan nie voorkom nie
- CMP stel twee bisse N ("non-zero") en Z ("zero")
- CMP a,b,c
  - R.a := R.b − R.c
  - Instruksies soos BEQ verwys na Z en N vlaggies wat gestel word deur CMP

IF-opdragte

```
IF x < y      LDW  1,0,-4   (-4: x)
              LDW  2,0,-8   (-8: y)
              CMP  1,1,2    R1 := R1 - R2
              BGE  1,0,3    spring 3 verder
THEN (* x < y *)
   x := 0     STW  0,0,-4
              BEQ  0,0,3    spring na end
ELSE (* x >= y *)
   x := 1     ADDI 1,0,1    R1 := 1
              STW  1,0,-4
END
```


Kode vir lusse

- Gebruik (CMP) vir toetse soos vir IF-opdrag
- Spring uit lus indien kondisie vals is
- Spring terug na begin aan einde van lus

Voorbeeld:

```
c := 5;           ADDI 1,0,5
                  STW  1,0,-8  (* -8: c *)
WHILE c > d DO    LDW  1,0,-8
                  LDW  2,0,-12 (* -12: d *)
                  CMP  1,1,2   (* c > d? *)
                  BLE  1,0,4
   c := c - 1     SUBI 1,1,1
                  STW  1,0,-8
                  BEQ  0,0,-6  (* na begin *)
END
```


Logiese bewerkings

- Logiese uitdrukkings word ontleed amper soos rekenkundige uitdrukkings, maar 'n lys van toetse en spronge is meer effektief
- "a OR b" beteken "if a then true else b"
- "a AND b" beteken "if a then b else false"
- Maak seker van die semantiek van die brontaal
  - indien "a" vals is, mag "b" uitgevoer word in "a AND b"?
  - wat moet gebeur as geen takke van 'n IF-opdrag uitvoerbaar is nie?

Kode vir logiese uitdrukkings

Voorbeeld: $(a < b)$ AND $(c < d)$ AND $(e < f)$

```
  CMP   1,a,b
  BGE   1,0, ---> F
  CMP   1,c,d
  BGE   1,0, ---> F
  CMP   1,e,f
  BGE   1,0, ---> F
- T -
```


Logiese toekennings

Voorstelling van waarheidswaardes: false = 0, true = 1

Kondisievlaggies ("Z" en "N") bevat resultaat van logiese toetse, maar dit kan nie direk toegeken word aan 'n register nie

Kodeformaat vir logiese toekenning

```
x := expr
```

```
   expr
   - T -
   ADDI  1,0,1
   BEQ   0,0,2
   ADDI  1,0,0 <--- F
   STW   1,x
```


Looptyd organisasie

- Prosedures moet so effektief moontlik implementeer word omdat dit so baie gebruik word
- Spesiale instruksies om implementering van prosedures effektief te maak:
  - Prosedureroep: `BSR`
  - Terugkeer: `RET`
- Gebruik stapel en aktiveringsrekords
- R31 (skakel: terugkeeradres)

Bewaring van terugkeeradres

R30 (wyser na top van stapel)

Prosedureroep:

```
BSR P   (R31 := PC; Jump to P)
```

Begin van prosedure (druk skakel op stapel):

```
PSH   31,30,4  (R30 := R30-4; M[R[30]] := R31)
```

Einde van prosedure (herstel skakel):

```
POP   31,30,4
RET   0,0,31
```

Aktiveringsrekords

- Arbitrêre keuse van registers om aktiveringsrekord af te baken:
  - R30: stapelwyser ("stack pointer")
  - R29: basiswyser ("frame pointer")

Skep en vernietiging van aktiveringsrekords

Begin van prosedure:

```
PSH   31,30,4   druk skakel op stapel
PSH   29,30,4   druk basiswyser op stapel
ADD   29,0,30   FP := SP (verskuif basiswyser)
SUBI  30,30,n   plek vir plaaslike data
```

Einde van prosedure:

```
ADD   30,0,29   SP := FP
POP   29,30,4   herstel FP
POP   31,30,4   herstel skakel
RET   0,0,31    keer terug
```