

Compilers

Nikos Papaspyrou Kostis Sagonas

nickie@softlab.ntua.gr kostis@cs.ntua.gr



National Technical University of Athens
School of Electrical and Computer Engineering
Software Engineering Laboratory
Polytechnioupoli, 15780 Zografou, Athens, Greece.

March 2017

Chapter 7:

Intermediate code

Intermediate code (i)

- ▶ **Why it is useful**
 - ▶ It facilitates translation
 - ▶ It facilitates optimization
 - ▶ It facilitates organization in front- and back-end

Intermediate code (ii)

- ▶ **Syntax-directed translation**
 - ▶ For each language construct, the corresponding intermediate code is defined
 - ▶ The parser is extended with semantic routines for generating the intermediate code

Intermediate code (ii)

- ▶ **Syntax-directed translation**
 - ▶ For each language construct, the corresponding intermediate code is defined
 - ▶ The parser is extended with semantic routines for generating the intermediate code
- ▶ **Templates** for generating intermediate code

Intermediate code (ii)

- ▶ **Syntax-directed translation**
 - ▶ For each language construct, the corresponding intermediate code is defined
 - ▶ The parser is extended with semantic routines for generating the intermediate code
- ▶ **Templates** for generating intermediate code
- ▶ **Attributes** for each symbol of the grammar

- ▶ **Quadruples**

$n: op, x, y, z$

Intermediate representation

(i)

▶ Quadruples

$n: op, x, y, z$

▶ Example:

$b*b-4*a*c$

1: *, b, b, \$1

2: *, 4, a, \$2

3: *, \$2, c, \$3

4: -, \$1, \$3, \$4

- ▶ Triples

$n: op, x, y$

Intermediate representation

(ii)

▶ Triples

$n: op, x, y$

▶ Example:

$b * b - 4 * a * c$

1: $*$, b , b

2: $*$, 4 , a

3: $*$, (2) , c

4: $-$, (1) , (3)

- ▶ *Abstract syntax trees*

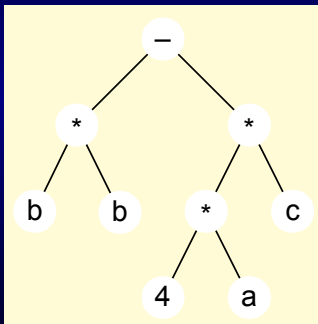
Intermediate representation

(iii)

- ▶ Abstract syntax trees

- ▶ Example:

$b * b - 4 * a * c$



Intermediate representation (iv)

- ▶ Prefix and postfix code

Intermediate representation

(iv)

- ▶ Prefix and postfix code

- ▶ Example:

`b*b-4*a*c`

`- * b b * * 4 a c`

prefix

`b b * 4 a * c * -`

postfix

Intermediate representation

(v)

- ▶ Directed acyclic graphs

Intermediate representation

(v)

- ▶ Directed acyclic graphs

- ▶ Example:

```
b+(a+1)*(a+1)+c-(a+1)*(a+1)+2/(a+1)
```

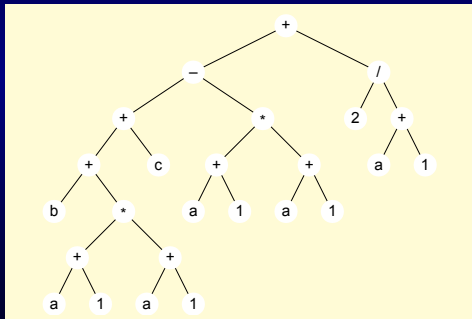

Intermediate representation

(v)

- ▶ Directed acyclic graphs

- ▶ Example:

$b+(a+1)^*(a+1)+c-(a+1)^*(a+1)+2/(a+1)$



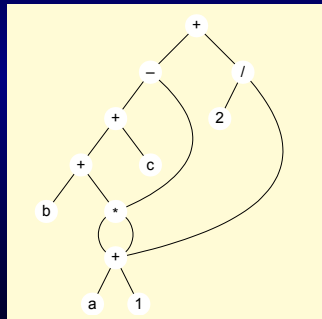
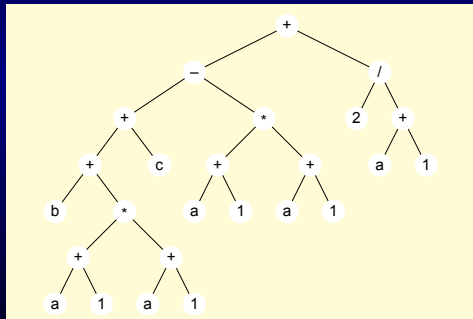
Intermediate representation

(v)

- ▶ Directed acyclic graphs

- ▶ Example:

$$b+(a+1)^*(a+1)+c-(a+1)^*(a+1)+2/(a+1)$$



The language of quadruples

- ▶ Form of a quadruple:

$n: op, x, y, z$

where:

- ▶ n : **label** (a natural number)
 - ▶ op : **operator**
 - ▶ x, y, z : **operands**
- ▶ For different types of operators, some operands may be omitted

Operands (i)

- ▶ **Constant**
 - ▶ integer, real, boolean
 - ▶ character, string, nil
- ▶ **Name**
 - ▶ variable, parameter, subprogram
- ▶ **Temporary variable:** $\$n$
- ▶ **Function result:** $\$\$$
- ▶ **Dereference:** $[x]$ x simple operand
- ▶ **Address:** $\{x\}$ x simple operand

Operands (ii)

- ▶ **Label**
 - ▶ of a statement in the source code
 - ▶ of a label
- ▶ **Parameter passing mode**
 - ▶ V : by value
 - ▶ R : by reference
 - ▶ RET : address of function result
- ▶ **Blank** : —
- ▶ **Temporarily blank** : * (for backpatching)

Operators (i)

- ▶ **unit**, I , $-$, $-$
- ▶ **endu**, I , $-$, $-$
beginning and end of a program unit
- ▶ **op**, x , y , z $op \in \{+, -, *, /, \%\}$
 $z := x \text{ op } y$
- ▶ **:=**, x , $-$, z
 $z := x$
- ▶ **array**, x , y , z
 $z :=$ the address of element $x[y]$

Operators (ii)

- ▶ *op*, *x*, *y*, *z* $op \in \{=, <>, >, <, >=, <=\}$
if $x \text{ op } y$ then jump to quadruple z
- ▶ *ifb*, *x*, $-$, *z*
if the boolean value x is true then jump to quadruple z
- ▶ *jump*, $-$, $-$, *z*
jump to quadruple z
- ▶ *label*, *I*, $-$, $-$
jumpl, $-$, $-$, *I*
define a label and jump to that

Operators (iii)

- ▶ `call`, $-$, $-$, I
call the program unit I
- ▶ `par`, x , m , $-$
pass the actual parameter x using passing mode m
- ▶ `ret`, $-$, $-$, $-$
return from the current program unit

Attributes used in intermediate code

- ▶ *PLACE*: the place where an l-value or r-value is stored
- ▶ *TYPE*: the type of an l-value or r-value
- ▶ *NEXT*: the list of quadruple labels that contain jumps to the next statement
- ▶ *TRUE, FALSE*: lists of labels containing jumps to the code that must be executed if a condition is true or false

Auxiliary routines (i)

- ▶ NEXTQUAD()
Returns the **number** of the next quadruple
- ▶ GENQUAD(*op, x, y, z*)
Generates the **next quadruple** *op, x, y, z*
- ▶ NEWTEMP(*t*)
Generates a **new temporary variable** of type *t*
- ▶ EMPTYLIST()
Generates an **empty list** of quadruple labels

Auxiliary routines (ii)

- ▶ **MAKELIST**(x)
Generates a list of quadruple labels containing **just the element** x
- ▶ **MERGE**(l_1, \dots, l_n)
Merges the list of quadruple labels $l_1 \dots l_n$
- ▶ **BACKPATCH**(l, z)
Replaces in all quadruples whose labels are contained in l the unknown quadruple label (last operand) with z (**backpatching**)

Arithmetic expressions

► Numerical constants

$\langle \text{r-value} \rangle ::= \langle \text{integer-const} \rangle \{ P_1 \}$

$P_1 : \{ \langle \text{r-value} \rangle . PLACE = \langle \text{integer-const} \rangle ; \}$

Arithmetic expressions

▶ Numerical constants

$\langle \text{r-value} \rangle ::= \langle \text{integer-const} \rangle \{ P_1 \}$

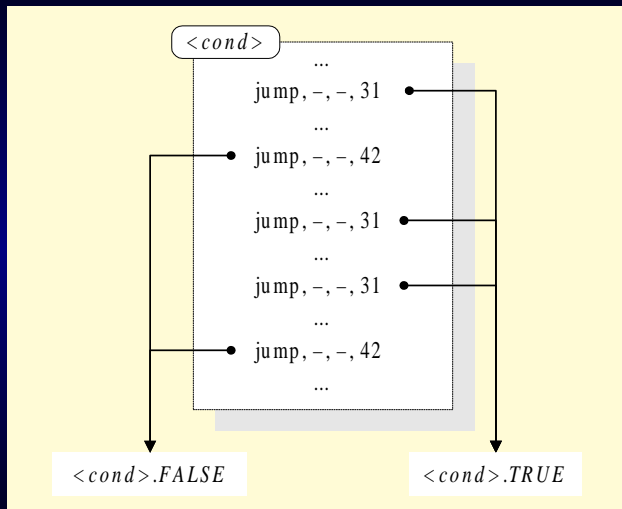
$P_1 : \{ \langle \text{r-value} \rangle . PLACE = \langle \text{integer-const} \rangle ; \}$

▶ Binary operators

$\langle \text{r-value} \rangle ::= \langle \text{expr} \rangle \langle \text{binop} \rangle \langle \text{expr} \rangle \{ P_{14} \}$

$P_{14} : \{ W = \text{NEWTEMP}(\langle \text{r-value} \rangle . TYPE);$
 $\text{GENQUAD}(\langle \text{binop} \rangle . NAME,$
 $\langle \text{expr} \rangle^1 . PLACE,$
 $\langle \text{expr} \rangle^2 . PLACE, W);$
 $\langle \text{r-value} \rangle . PLACE = W; \}$

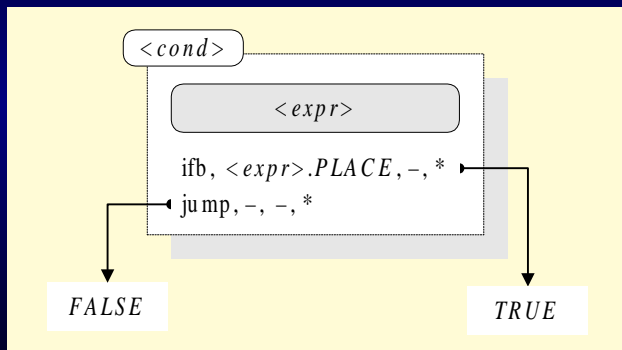
Conditions (i)



Conditions (ii)

- Conditions represented as 0/1

$\langle \text{cond} \rangle ::= \langle \text{expr} \rangle$



Conditions (iii)

► Conditions represented as 0/1

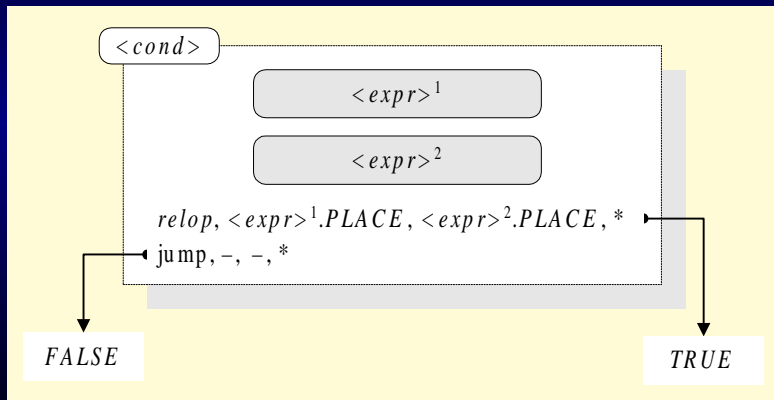
$\langle \text{cond} \rangle ::= \langle \text{expr} \rangle \{ P_{21} \}$

$P_{21} : \{ \langle \text{cond} \rangle . \text{TRUE} = \text{MAKELIST}(\text{NEXTQUAD}());$
 $\text{GENQUAD}(\text{ifb}, \langle \text{expr} \rangle . \text{PLACE}, -, *);$
 $\langle \text{cond} \rangle . \text{FALSE} = \text{MAKELIST}(\text{NEXTQUAD}());$
 $\text{GENQUAD}(\text{jump}, -, -, *); \}$

Conditions (iv)

► Arithmetic comparisons

$\langle \text{cond} \rangle ::= \langle \text{expr} \rangle^1 \langle \text{relop} \rangle \langle \text{expr} \rangle^2$



Conditions (v)

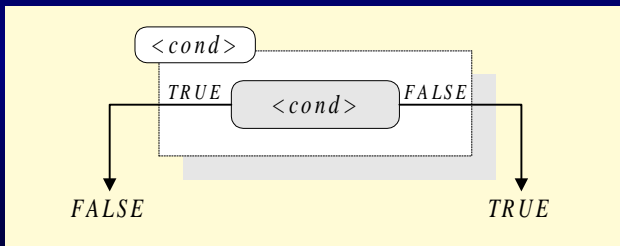
► Arithmetic comparisons

```
 $\langle \text{cond} \rangle ::= \langle \text{expr} \rangle^1 \langle \text{relop} \rangle \langle \text{expr} \rangle^2 \{ P_{23} \}$   
 $P_{23} : \{ \langle \text{cond} \rangle . \text{TRUE} = \text{MAKELIST}(\text{NEXTQUAD}());$   
           $\text{GENQUAD}(\langle \text{relop} \rangle . \text{NAME},$   
                   $\langle \text{expr} \rangle^1 . \text{PLACE},$   
                   $\langle \text{expr} \rangle^2 . \text{PLACE}, *);$   
           $\langle \text{cond} \rangle . \text{FALSE} = \text{MAKELIST}(\text{NEXTQUAD}());$   
           $\text{GENQUAD}(\text{jump}, -, -, *); \}$ 
```

Conditions (vi)

► Negation

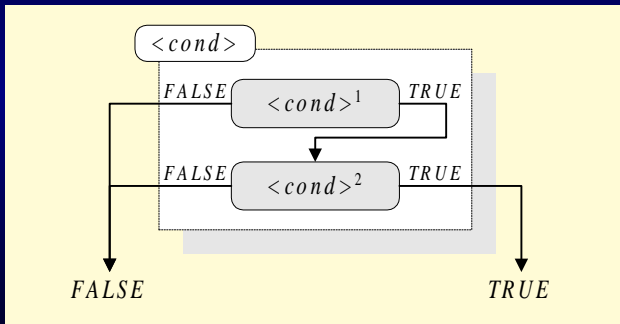
$\langle \text{cond} \rangle ::= \text{“not” } \langle \text{cond} \rangle$



Conditions (vii)

► Conjunction

$\langle \text{cond} \rangle ::= \langle \text{cond} \rangle^1 \text{ "and" } \langle \text{cond} \rangle^2$



Conditions (viii)

► Conjunction

$$\langle \text{cond} \rangle ::= \langle \text{cond} \rangle_1 \text{ “and” } \{ P_{25} \} \langle \text{cond} \rangle_2 \{ P_{26} \}$$
$$P_{25} : \{ \text{BACKPATCH}(\langle \text{cond} \rangle^1.\text{TRUE}, \text{NEXTQUAD}()); \}$$
$$P_{26} : \{ \langle \text{cond} \rangle.\text{FALSE} = \text{MERGE}(\langle \text{cond} \rangle^1.\text{FALSE}, \langle \text{cond} \rangle^2.\text{FALSE}); \}$$
$$\langle \text{cond} \rangle.\text{TRUE} = \langle \text{cond} \rangle^2.\text{TRUE}; \}$$

Simple statements

▶ The empty statement

$\langle \text{stmt} \rangle ::= \epsilon \{ P_{29} \}$

$P_{29} : \{ \langle \text{stmt} \rangle . \text{NEXT} = \text{EMPTYLIST}(); \}$

Simple statements

▶ The empty statement

$\langle \text{stmt} \rangle ::= \epsilon \{ P_{29} \}$

$P_{29} : \{ \langle \text{stmt} \rangle . \text{NEXT} = \text{EMPTYLIST}(); \}$

▶ Assignment

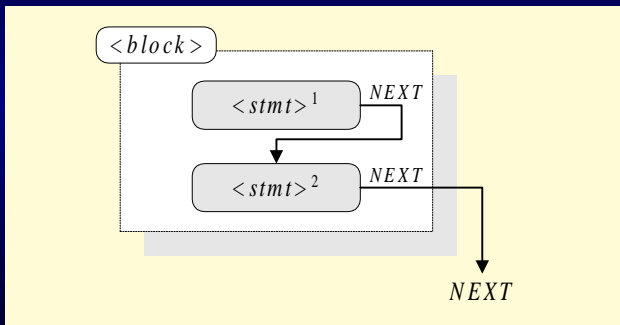
$\langle \text{stmt} \rangle ::= \langle \text{l-value} \rangle \text{ “:=” } \langle \text{expr} \rangle \{ P_{30} \}$

$P_{30} : \{ \text{GENQUAD}(\text{“:=”}, \langle \text{expr} \rangle . \text{PLACE}, -, \langle \text{l-value} \rangle . \text{PLACE}); \}$
 $\langle \text{stmt} \rangle . \text{NEXT} = \text{EMPTYLIST}(); \}$

Compound statement (i)

$\langle \text{stmt} \rangle ::= \langle \text{block} \rangle$

$\langle \text{block} \rangle ::= \text{“begin” } \langle \text{stmt} \rangle (\text{“;” } \langle \text{stmt} \rangle)^* \text{“end”}$

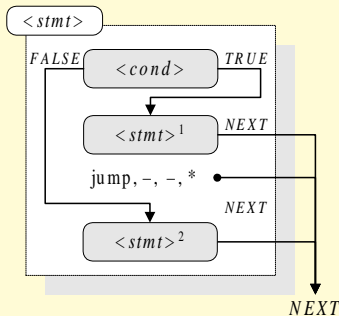
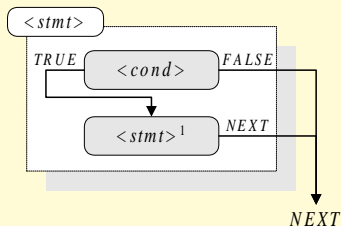


Compound statement (ii)

$$\begin{aligned}\langle \text{stmt} \rangle & ::= \langle \text{block} \rangle \{ P_{34} \} \\ P_{34} & : \{ \langle \text{stmt} \rangle . \text{NEXT} = \langle \text{block} \rangle . \text{NEXT}; \} \\ \langle \text{block} \rangle & ::= \text{“begin”} \langle \text{stmt} \rangle^1 \{ P_{35} \} \\ & \quad (\text{“;”} \{ P_{36} \} \langle \text{stmt} \rangle^2 \{ P_{37} \})^* \text{“end”} \{ P_{38} \} \\ P_{35} & : \{ L = \langle \text{stmt} \rangle^1 . \text{NEXT}; \} \\ P_{36} & : \{ \text{BACKPATCH}(L, \text{NEXTQUAD}()); \} \\ P_{37} & : \{ L = \langle \text{stmt} \rangle^2 . \text{NEXT}; \} \\ P_{38} & : \{ \langle \text{block} \rangle . \text{NEXT} = L; \}\end{aligned}$$

If statement (i)

$\langle \text{stmt} \rangle ::= \text{“if” } \langle \text{cond} \rangle \text{ “then” } \langle \text{stmt} \rangle [\text{“else” } \langle \text{stmt} \rangle]$



If statement (ii)

$\langle \text{stmt} \rangle ::= \text{“if” } \langle \text{cond} \rangle \{ P_{39} \} \text{“then” } \langle \text{stmt} \rangle^1$
 $\quad [\text{“else” } \{ P_{40} \} \langle \text{stmt} \rangle^2 \{ P_{41} \}] \{ P_{42} \}$

$P_{39} : \{ \text{BACKPATCH}(\langle \text{cond} \rangle.\text{TRUE}, \text{NEXTQUAD}());$
 $\quad L_1 = \langle \text{cond} \rangle.\text{FALSE};$
 $\quad L_2 = \text{EMPTYLIST}(); \}$

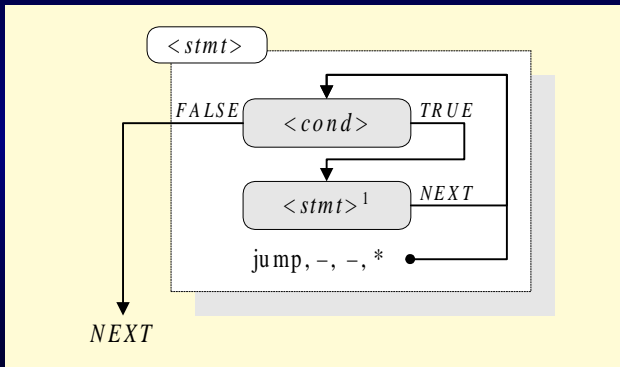
$P_{40} : \{ L_1 = \text{MAKELIST}(\text{NEXTQUAD}());$
 $\quad \text{GENQUAD}(\text{jump}, -, -, *);$
 $\quad \text{BACKPATCH}(\langle \text{cond} \rangle.\text{FALSE}, \text{NEXTQUAD}()); \}$

$P_{41} : \{ L_2 = \langle \text{stmt} \rangle^2.\text{NEXT}; \}$

$P_{42} : \{ \langle \text{stmt} \rangle.\text{NEXT} = \text{MERGE}(L_1, \langle \text{stmt} \rangle^1.\text{NEXT}, L_2); \}$

While statement (i)

$\langle \text{stmt} \rangle ::= \text{“while” } \langle \text{cond} \rangle \text{ “do” } \langle \text{stmt} \rangle$



While statement (ii)

$\langle \text{stmt} \rangle ::= \text{“while” } \{ P_{43} \} \langle \text{cond} \rangle \text{ “do” } \{ P_{44} \} \langle \text{stmt} \rangle^1 \{ P_{45} \}$

$P_{43} : \{ Q = \text{NEXTQUAD}(); \}$

$P_{44} : \{ \text{BACKPATCH}(\langle \text{cond} \rangle.\text{TRUE}, \text{NEXTQUAD}()); \}$

$P_{45} : \{ \text{BACKPATCH}(\langle \text{stmt} \rangle^1.\text{NEXT}, Q);$
 $\text{GENQUAD}(\text{jump}, -, -, Q);$
 $\langle \text{stmt} \rangle.\text{NEXT} = \langle \text{cond} \rangle.\text{FALSE}; \}$

Subprogram calls (i)

$\langle \text{call} \rangle ::= \langle \text{id} \rangle \text{ “(” } [\langle \text{expr} \rangle (\text{“,”} \langle \text{expr} \rangle)^*] \text{ “)”}$

$\langle \text{r-value} \rangle ::= \langle \text{call} \rangle$

$\langle \text{stmt} \rangle ::= \langle \text{call} \rangle$

- ▶ Parameter passing, using `par` quadruples
- ▶ Address of result, using an extra `par` quadruple, if it is a function
- ▶ Call, using a `call` quadruple

Subprogram calls (ii)

$$\langle \text{call} \rangle ::= \langle \text{id} \rangle \text{ “(” } \{ P_{46} \} [\langle \text{expr} \rangle^1 \{ P_{47} \} \\ \text{ (“,” } \langle \text{expr} \rangle^2 \{ P_{48} \})^*] \text{ “)” } \{ P_{49} \}$$
$$P_{46} : \{ N = 1; \}$$
$$P_{47} : \{ \text{GENQUAD}(\text{“par”}, \langle \text{expr} \rangle^1.P\text{PLACE}, \\ \text{PARAMMODE}(\langle \text{id} \rangle, N), -); \\ N = N + 1; \}$$
$$P_{48} : \{ \text{GENQUAD}(\text{“par”}, \langle \text{expr} \rangle^2.P\text{PLACE}, \\ \text{PARAMMODE}(\langle \text{id} \rangle, N), -); \\ N = N + 1; \}$$

Subprogram calls (iii)

$\langle \text{call} \rangle ::= \langle \text{id} \rangle \text{ “(” } \{ P_{46} \} [\langle \text{expr} \rangle^1 \{ P_{47} \} (\text{ “,” } \langle \text{expr} \rangle^2 \{ P_{48} \})^*] \text{ “)” } \{ P_{49} \}$

(continued)

```
 $P_{49}$  : { if (ISFUNCTION( $\langle \text{id} \rangle$ )) {  
     $W = \text{NEWTEMP}(\text{FUNCRESULT}(\langle \text{id} \rangle));$   
    GENQUAD(par, RET,  $W$ , -);  
     $\langle \text{call} \rangle . \text{PLACE} = W;$   
    }  
    GENQUAD(call, -, -,  $\langle \text{id} \rangle$ ); }
```


Subprogram calls (iv)

► Function call

$\langle \text{r-value} \rangle ::= \langle \text{call} \rangle \{ P_{50} \}$

$P_{50} : \{ \langle \text{r-value} \rangle . PLACE = \langle \text{call} \rangle . PLACE; \}$

Subprogram calls (iv)

▶ Function call

$\langle \text{r-value} \rangle ::= \langle \text{call} \rangle \{ P_{50} \}$

$P_{50} : \{ \langle \text{r-value} \rangle . PLACE = \langle \text{call} \rangle . PLACE; \}$

▶ Procedure call

$\langle \text{stmt} \rangle ::= \langle \text{call} \rangle \{ P_{51} \}$

$P_{51} : \{ \langle \text{stmt} \rangle . NEXT = \text{EMPTYLIST}(); \}$

Subprogram call (v)

► Return from a subprogram

$\langle \text{stmt} \rangle ::= \text{“return”} [\langle \text{expr} \rangle \{ P_{52} \}] \{ P_{53} \}$

$P_{52} : \{ \text{GENQUAD}(\text{retv}, \langle \text{expr} \rangle . \text{PLACE}, -, -); \}$

$P_{53} : \{ \text{GENQUAD}(\text{ret}, -, -, -); \}$

Subprogram call (v)

▶ Return from a subprogram

$\langle \text{stmt} \rangle ::= \text{“return”} [\langle \text{expr} \rangle \{ P_{52} \}] \{ P_{53} \}$

$P_{52} : \{ \text{GENQUAD}(\text{retv}, \langle \text{expr} \rangle . \text{PLACE}, -, -); \}$

$P_{53} : \{ \text{GENQUAD}(\text{ret}, -, -, -); \}$

▶ Subprogram definition

$\langle \text{body} \rangle ::= (\langle \text{local} \rangle)^* \{ P_{56} \} \langle \text{block} \rangle \text{“;”} \{ P_{57} \}$

$P_{56} : \{ \text{GENQUAD}(\text{unit}, I, -, -); \}$

$P_{57} : \{ \text{BACKPATCH}(\langle \text{block} \rangle . \text{NEXT}, \text{NEXTQUAD}()); \}$
 $\text{GENQUAD}(\text{endu}, I, -, -); \}$

Example (i)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;  
    while i <= j do begin  
        while a[i] < a[(m+n) div 2] do i := i+1;  
        while a[j] > a[(m+n) div 2] do j := j-1;  
        if i <= j then begin  
            temp := a[i]; a[i] := a[j]; a[j] := temp;  
            i := i+1; j := j-1  
        end  
    end;  
    quicksort(a, m, j); quicksort(a, i, n)  
end;
```

Example (ii)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;
```

Example (ii)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;
```

1: unit, quicksort, —, —

Example (ii)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;
```

1: unit, quicksort, —, —

2: <=, n, m, *

Example (ii)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;
```

- 1: unit, quicksort, —, —
- 2: <=, n, m, *
- 3: jump, —, —, *

Example (ii)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;
```

- 1: unit, quicksort, —, —
- 2: <=, n, m, *
- 3: jump, —, —, *
- 4: ret, —, —, —

Example (ii)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;
```

1: unit, quicksort, —, —

2: <=, n, m, 4

3: jump, —, —, *

4: ret, —, —, —

Example (ii)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;
```

1: unit, quicksort, —, —

2: <=, n, m, 4

3: jump, —, —, 5

4: ret, —, —, —

Example (ii)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;
```

- 1: unit, quicksort, —, —
- 2: <=, n, m, 4
- 3: jump, —, —, 5
- 4: ret, —, —, —
- 5: :=, m, —, i

Example (ii)

```
procedure quicksort (var a : array of integer;  
                    m, n : integer);  
    var i, j, temp : integer;  
begin  
    if n <= m then return;  
    i := m; j := n;
```

1: unit, quicksort, —, —

2: <=, n, m, 4

3: jump, —, —, 5

4: ret, —, —, —

5: :=, m, —, i

6: :=, n, —, j

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, *

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, *

8: jump, -, -, *

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9

8: jump, -, -, *

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9

8: jump, -, -, *

9: array, a, i, \$1

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9

8: jump, -, -, *

9: array, a, i, \$1

10: +, m, n, \$2

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9

8: jump, -, -, *

9: array, a, i, \$1

10: +, m, n, \$2

11: /, \$2, 2, \$3

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9

8: jump, -, -, *

9: array, a, i, \$1

10: +, m, n, \$2

11: /, \$2, 2, \$3

12: array, a, \$3, \$4

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
12: array, a, $3, $4
13: <, [$1], [$4], *
```

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
12: array, a, $3, $4
13: <, [$1], [$4], *
14: jump, -, -, *
```


Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
12: array, a, $3, $4
13: <, [$1], [$4], 15
14: jump, -, -, *
```

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
12: array, a, $3, $4
13: <, [$1], [$4], 15
14: jump, -, -, *
15: +, i, 1, $5
```

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

```
7: <=, i, j, 9
8: jump, -, -, *
9: array, a, i, $1
10: +, m, n, $2
11: /, $2, 2, $3
12: array, a, $3, $4
13: <, [$1], [$4], 15
14: jump, -, -, *
15: +, i, 1, $5
16: :=, $5, -, i
```

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9 17: jump, -, -, 9
8: jump, -, -, *
9: array, a, i, \$1
10: +, m, n, \$2
11: /, \$2, 2, \$3
12: array, a, \$3, \$4
13: <, [\$1], [\$4], 15
14: jump, -, -, *
15: +, i, 1, \$5
16: :=, \$5, -, i

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

7: <=, i, j, 9 17: jump, -, -, 9
8: jump, -, -, *
9: array, a, i, \$1
10: +, m, n, \$2
11: /, \$2, 2, \$3
12: array, a, \$3, \$4
13: <, [\$1], [\$4], 15
14: jump, -, -, 18
15: +, i, 1, \$5
16: :=, \$5, -, i

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|----------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | |
| 10: +, m, n, \$2 | |
| 11: /, \$2, 2, \$3 | |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|----------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | |
| 11: /, \$2, 2, \$3 | |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|----------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|------------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|------------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | 22: >, [\$6], [\$9], * |
| 13: <, [\$1], [\$4], 15 | |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|------------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | 22: >, [\$6], [\$9], * |
| 13: <, [\$1], [\$4], 15 | 23: jump, -, -, * |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|-------------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | 22: >, [\$6], [\$9], 24 |
| 13: <, [\$1], [\$4], 15 | 23: jump, -, -, * |
| 14: jump, -, -, 18 | |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|-------------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | 22: >, [\$6], [\$9], 24 |
| 13: <, [\$1], [\$4], 15 | 23: jump, -, -, * |
| 14: jump, -, -, 18 | 24: -, j, 1, \$10 |
| 15: +, i, 1, \$5 | |
| 16: :=, \$5, -, i | |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|-------------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | 22: >, [\$6], [\$9], 24 |
| 13: <, [\$1], [\$4], 15 | 23: jump, -, -, * |
| 14: jump, -, -, 18 | 24: -, j, 1, \$10 |
| 15: +, i, 1, \$5 | 25: :=, \$10, -, j |
| 16: :=, \$5, -, i | |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|-------------------------|
| 7: <=, i, j, 9 | 17: jump, —, —, 9 |
| 8: jump, —, —, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | 22: >, [\$6], [\$9], 24 |
| 13: <, [\$1], [\$4], 15 | 23: jump, —, —, * |
| 14: jump, —, —, 18 | 24: —, j, 1, \$10 |
| 15: +, i, 1, \$5 | 25: :=, \$10, —, j |
| 16: :=, \$5, —, i | 26: jump, —, —, 18 |

Example (iii)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
```

| | |
|-------------------------|-------------------------|
| 7: <=, i, j, 9 | 17: jump, -, -, 9 |
| 8: jump, -, -, * | 18: array, a, j, \$6 |
| 9: array, a, i, \$1 | 19: +, m, n, \$7 |
| 10: +, m, n, \$2 | 20: /, \$7, 2, \$8 |
| 11: /, \$2, 2, \$3 | 21: array, a, \$8, \$9 |
| 12: array, a, \$3, \$4 | 22: >, [\$6], [\$9], 24 |
| 13: <, [\$1], [\$4], 15 | 23: jump, -, -, 27 |
| 14: jump, -, -, 18 | 24: -, j, 1, \$10 |
| 15: +, i, 1, \$5 | 25: :=, \$10, -, j |
| 16: :=, \$5, -, i | 26: jump, -, -, 18 |

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

Example (iv)

```
if i <= j then begin
  temp := a[i]; a[i] := a[j]; a[j] := temp;
  i := i+1; j := j-1
end
```

27: <=, i, j, *

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, *

28: jump, -, -, *

Example (iv)

```
if i <= j then begin
  temp := a[i]; a[i] := a[j]; a[j] := temp;
  i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

```
27: <=, i, j, 29
28: jump, -, -, *
29: array, a, i, $11
30: :=, [$11], -, temp
31: array, a, i, $12
32: array, a, j, $13
```


Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

```
27: <=, i, j, 29
28: jump, -, -, *
29: array, a, i, $11
30: :=, [$11], -, temp
31: array, a, i, $12
32: array, a, j, $13
33: :=, [$13], -, [$12]
```

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

36: +, i, 1, \$15

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

36: +, i, 1, \$15

37: :=, \$15, -, i

Example (iv)

```
if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

36: +, i, 1, \$15

37: :=, \$15, -, i

38: -, j, 1, \$16

Example (iv)

```
if i <= j then begin
  temp := a[i]; a[i] := a[j]; a[j] := temp;
  i := i+1; j := j-1
end
```

27: <=, i, j, 29

28: jump, -, -, *

29: array, a, i, \$11

30: :=, [\$11], -, temp

31: array, a, i, \$12

32: array, a, j, \$13

33: :=, [\$13], -, [\$12]

34: array, a, j, \$14

35: :=, temp, -, [\$14]

36: +, i, 1, \$15

37: :=, \$15, -, i

38: -, j, 1, \$16

39: :=, \$16, -, j

Example (v)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
  if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
  end
end
end;
```

Example (v)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
  if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
  end
end
end;
```

7: <=, i, j, 9

8: jump, -, -, *

Example (v)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
  if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
  end
end
end;
```

7: <=, i, j, 9

8: jump, -, -, *

...

27: <=, i, j, 29

28: jump, -, -, *

Example (v)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
  if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
  end
end
end;
```

7: <=, i, j, 9

8: jump, —, —, *

...

27: <=, i, j, 29

28: jump, —, —, *

...

40: jump, —, —, 7

Example (v)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
  if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
  end
end
end;
```

7: <=, i, j, 9

8: jump, —, —, *

...

27: <=, i, j, 29

28: jump, —, —, 7

...

40: jump, —, —, 7

Example (v)

```
while i <= j do begin
  while a[i] < a[(m+n) div 2] do i := i+1;
  while a[j] > a[(m+n) div 2] do j := j-1;
  if i <= j then begin
    temp := a[i]; a[i] := a[j]; a[j] := temp;
    i := i+1; j := j-1
  end
end
end;
```

7: <=, i, j, 9

8: jump, —, —, 41

...

27: <=, i, j, 29

28: jump, —, —, 7

...

40: jump, —, —, 7

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, —

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, —

42: par, m, V, —

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, —

42: par, m, V, —

43: par, j, V, —

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, —

42: par, m, V, —

43: par, j, V, —

44: call, —, —, quicksort

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, —

42: par, m, V, —

43: par, j, V, —

44: call, —, —, quicksort

45: par, a, R, —

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

- 41: par, a, R, —
- 42: par, m, V, —
- 43: par, j, V, —
- 44: call, —, —, quicksort
- 45: par, a, R, —
- 46: par, i, V, —

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, —
42: par, m, V, —
43: par, j, V, —
44: call, —, —, quicksort
45: par, a, R, —
46: par, i, V, —
47: par, n, V, —

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, —
42: par, m, V, —
43: par, j, V, —
44: call, —, —, quicksort
45: par, a, R, —
46: par, i, V, —
47: par, n, V, —
48: call, —, —, quicksort

Example (vi)

```
    quicksort(a, m, j); quicksort(a, i, n)
end;
```

41: par, a, R, —
42: par, m, V, —
43: par, j, V, —
44: call, —, —, quicksort
45: par, a, R, —
46: par, i, V, —
47: par, n, V, —
48: call, —, —, quicksort
49: endu, quicksort, —, —