CS251: Introduction to Language Processing

Intermediate Code Generation

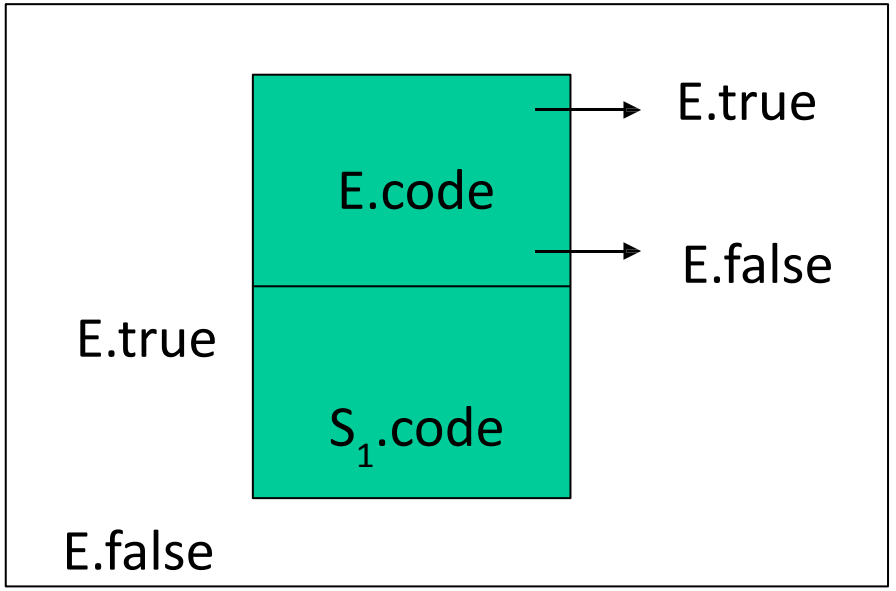
Vishwesh Jatala

Department of CSE

Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

2023-24-M



$S \rightarrow$ if E then S_1

E.true = newlabel

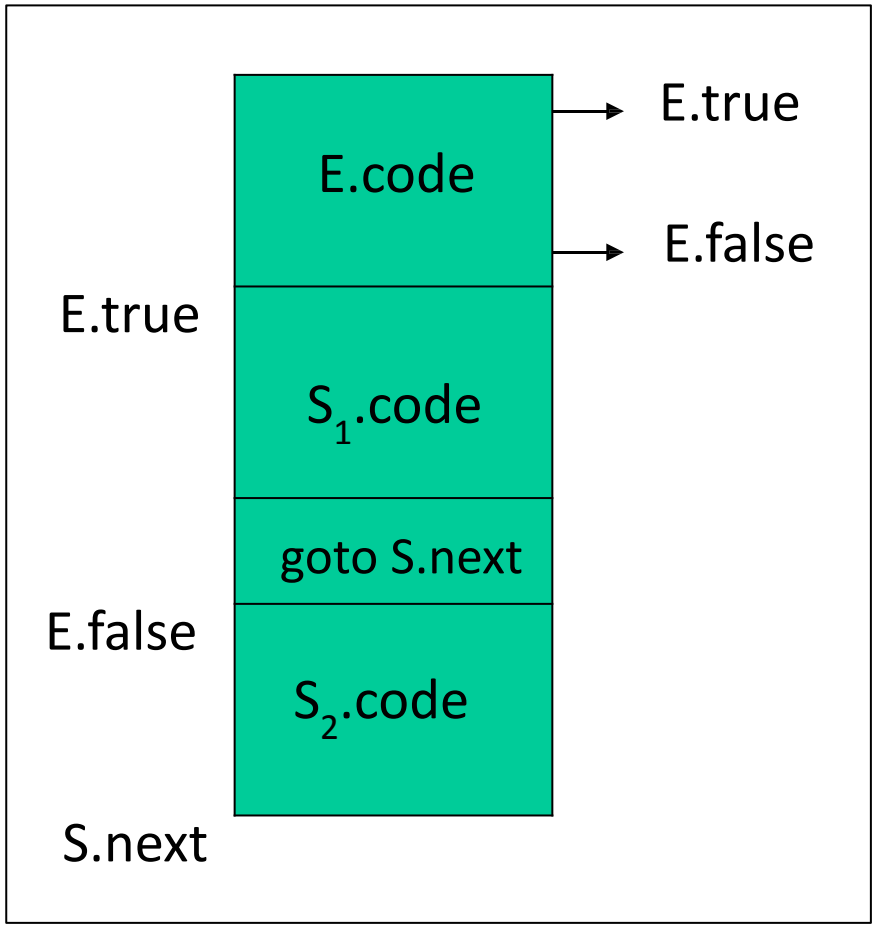
E.false = S.next

S_1.next = S.next

S.code = E.code ||

gen(E.true ':') ||

S_1.code



$S \rightarrow$ if E then S_1 else S_2

E.true = newlabel

E.false = newlabel

S_1.next = S.next

S_2.next = S.next

S.code = E.code ||

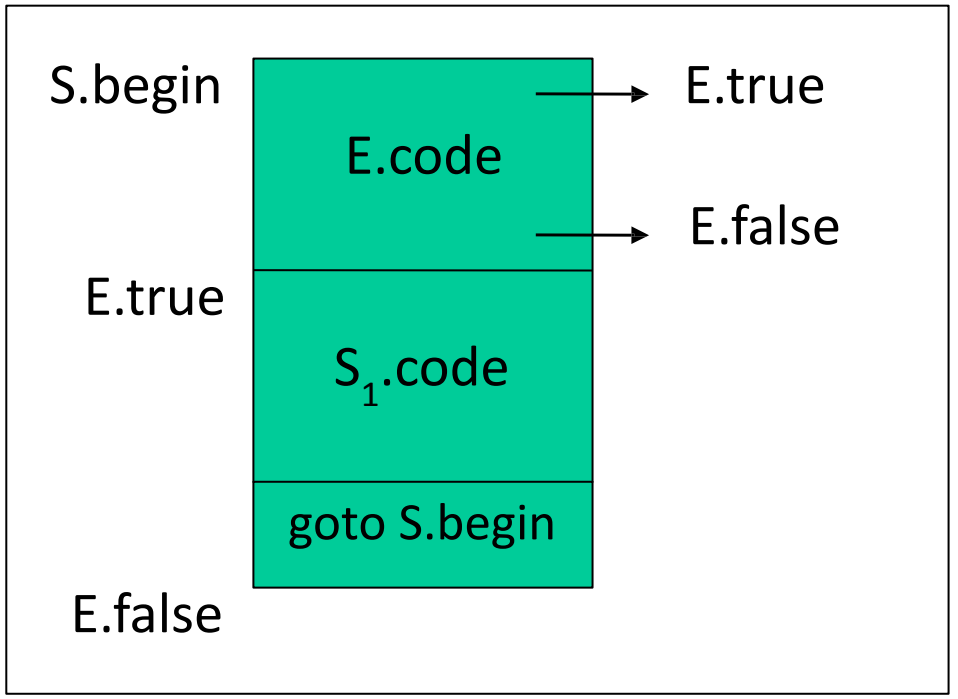
gen(E.true ':') ||

S_1.code ||

gen(goto S.next) ||

gen(E.false ':') ||

S_2.code



$S \rightarrow$ while E do S_1

S.begin = newlabel

E.true = newlabel

E.false = S.next

S_1.next = S.begin

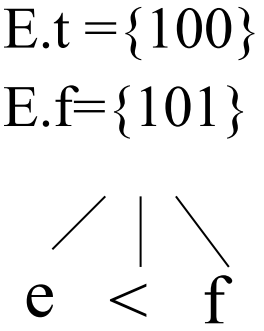
S.code = gen(S.begin ':') ||
E.code ||
gen(E.true ':') ||
S_1.code ||
gen(goto S.begin)

BackPatching

- Way to implement boolean expressions and flow of control statements in one pass
- We may not the know the target labels
- leave them unspecified
- Back Patching is putting the address instead of labels when the proper label is determined

Generate code for e < f

Initialize nextquad to 100



100: if e < f goto -
101 goto –

BackPatching

- **makelist(i):** create a newlist containing only i, return a pointer to the list.
- **merge(p1,p2):** merge lists pointed to by p1 and p2 and return a pointer to the concatenated list
- **backpatch(p,i):** insert i as the target label for the statements in the list pointed to by p

$E \rightarrow id_1$ relop id_2

E.truelist = makelist(nextquad)

E.falselist = makelist(nextquad+ 1)

emit(if id_1 relop id_2 goto ---)

emit(goto ---)

Boolean Expressions

$E \rightarrow E_1$ or M E_2

$M \rightarrow \epsilon$

- Insert a marker non terminal M into the grammar to pick up index of next quadruple

$E \rightarrow E_1$ or $M\ E_2$

backpatch(E_1.falselist, M.quad)

E.truelist = merge(E_1.truelist, E_2.truelist)

E.falselist = E_2.falselist

$M \rightarrow \epsilon$

M.quad = nextquad

$E \rightarrow E_1$ and M E_2

backpatch(E_1.truelist, M.quad)

E.truelist = E_2.truelist

E.falselist = merge(E_1.falselist, E_2.falselist)

$E \rightarrow$ not E_1

E.truelist = E_1 falselist

E.falselist = E_1.truelist

E → true

E.truelist = makelist(nextquad) emit(goto ---)

E → false

E.falselist = makelist(nextquad)

emit(goto ---)

M → ϵ M.quad = nextquad

Generate code for

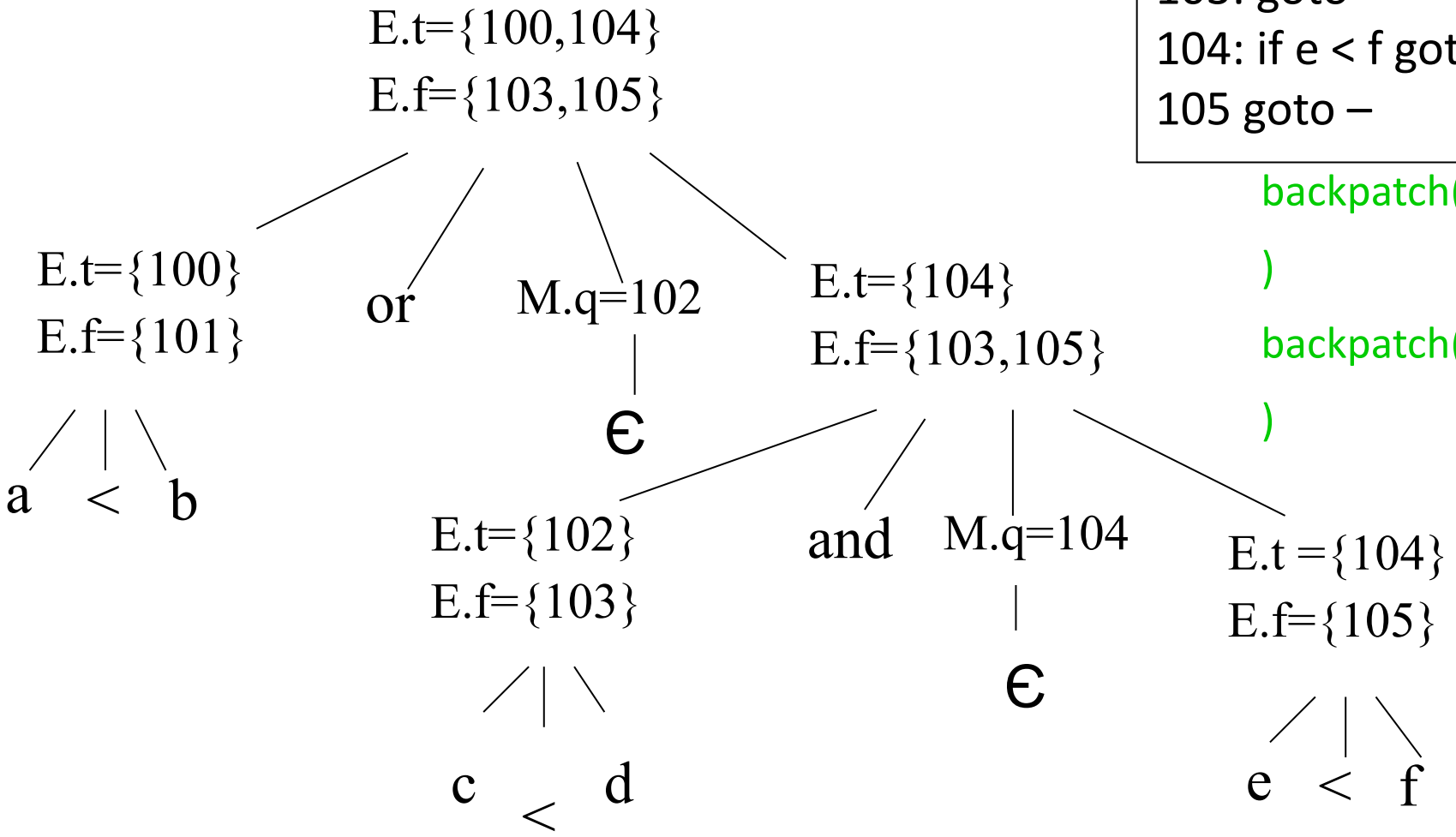
a < b or c < d and e < f

Initialize nextquad to 100

```
100: if a < b goto -
101: goto -    102
102: if c < d goto -    104
103: goto -
104: if e < f goto -
105 goto –
```

backpatch(102,104)

backpatch(101,102)



Flow of Control Statements

$S \rightarrow$ if E then S_1

| if E then S_1 else S_2

| while E do S_1

| begin L end

| A

$L \rightarrow L ; S$

| S

S : Statement

A : Assignment

L : Statement list

Scheme to implement translation

$S \rightarrow$ if E then M S_1

backpatch(E.truelist, M.quad)

S.nextlist = merge(E.falselist,S_1.nextlist)

$S \rightarrow$ if E them M_1 S_1 N else M_2 S_2

backpatch(E.truelist, M_1.quad)

backpatch(E.falselist, M_2.quad)

S.next = merge(S_1.nextlist,
N.nextlist, S_2.nextlist)

Scheme to implement translation

$S \rightarrow$ while M_1 E do M_2 S_1

backpatch(S_1.nextlist, M_1.quad)

backpatch(E.truelist, M_2.quad)

S.nextlist = E.falselist

emit(goto M_1.quad)

Reading Exercise

- Intermediate Code for switch statements
  - Section 6.8

Arrays

Array Expressions

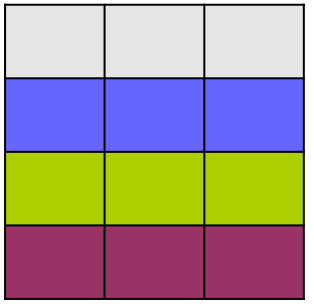
- What is the meaning of x = a[i]?
- a[i] is a+i*sizeof(type)

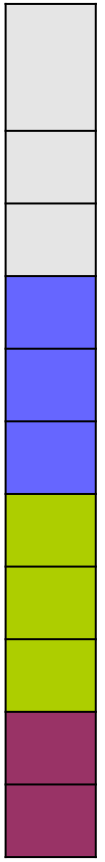
Array Expressions

- What is the meaning of a[i][j]?
  - Assume array declaration is int a[3][5];

Array Expressions

- a[i][j] for a[3][5]is
  a+i*5*sizeof(int)+j*sizeof(int)





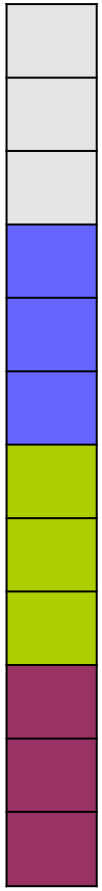
Array Expressions

- For instance, create IR for c+a[i][j].
- This requires us to know the types of a and c.
- Say, c is an integer (4 bytes) and a is int[2][3].
- Then, the IR is

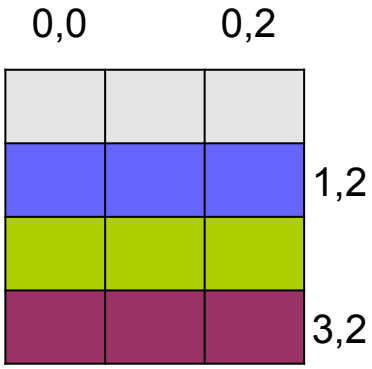
```
t1=i*12    ;3*4 bytes
t2=j*4     ;1*4 bytes
t3=t1+t2   ;offset from a
t4=a[t3]   ;assuming base[offset] is present in IR.
t5=c+t4
```

Array Representations

- In C, C++, Java, and so far, we have used *row-major* storage.

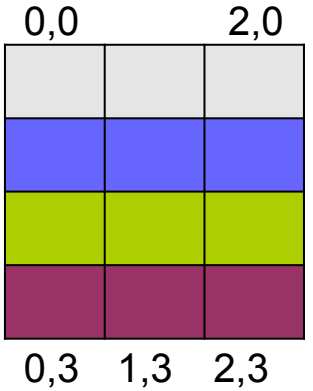


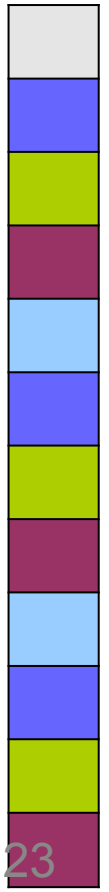
All elements of a row are stored together.



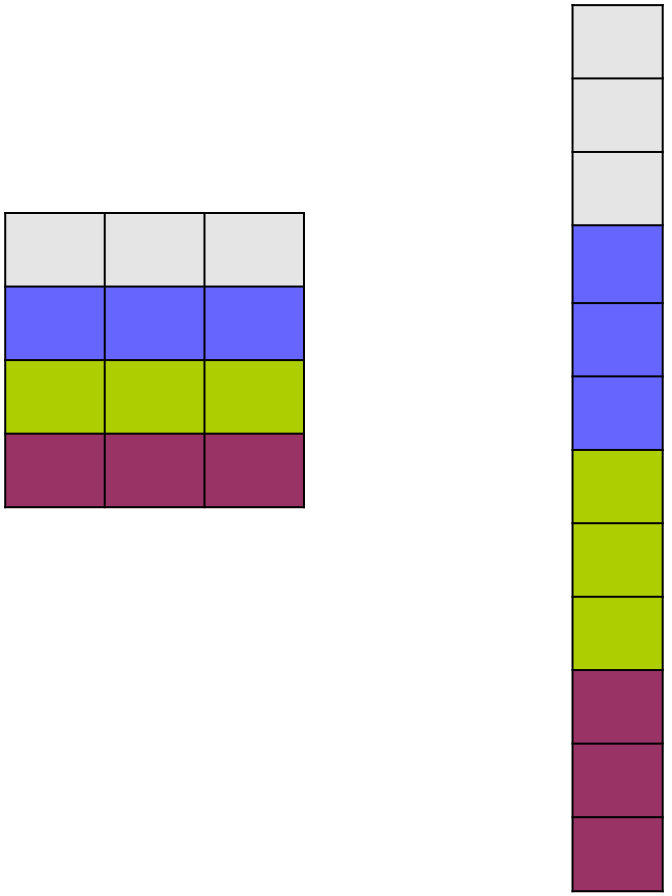
- In Fortran, we use *column-major* storage format.

each column is stored together.





Array Expressions



How to compute a[i][j][k][l]

with declaration as int a[w4][w3][w2][w1]?

Array Expressions

a[i][j][k][l]

Base = a

Offset = i*w3*w2*w1+j*w2*w1+k*w1+l

IR for Array Expressions

- $L \longrightarrow id\ [E] \mid L\ [E]$ // maintain three attributes: type, addr and base.

$L \rightarrow id\ [\ E\ ]$
$L \rightarrow L_1\ [\ E\ ]$
$E \rightarrow id$
$E \rightarrow L$
$E \rightarrow E_1 + E_2$
$S \rightarrow id = E$
$S \rightarrow L = E$

IR for Array Expressions

- L ⟶ id [E] | L [E] // maintain three attributes: type, addr and base.

L → id [E]	
L → L_1 [E]	
E → id	{ E.addr = id.addr; }
E → L	
E → E_1 + E_2	{ E.addr = new Temp(); gen(E.addr '=' E_1.addr + E_2.addr); }
S → id = E	{ gen(id.name '=' E.addr); }
S → L = E	

IR for Array Expressions

- L ⟶ id [E] | L [E] // maintain three attributes: type, addr and base.

L → id [E]	
L → L_1 [E]	
E → id	{ E.addr = id.addr; }
E → L	{ E.addr = new Temp(); gen(E.addr '=' L.base '[' L.addr ']'); }
E → E_1 + E_2	{ E.addr = new Temp(); gen(E.addr '=' E_1.addr + E_2.addr); }
S → id = E	{ gen(id.name '=' E.addr); }
S → L = E	{ gen(L.base '[' L.addr ']' '=' E.addr); }

Array Expressions

a[i][j][k][l]

Base = a

Offset/addr = i*w3*w2*w1+j*w2*w1+k*w1+l

IR for Array Expressions

- L ⟶ id [E] | L [E] // maintain three attributes: type, addr and base.

L → id [E]	{ L.type = id.type; L.addr = new Temp();
L → L_1 [E]	
E → id	{ E.addr = id.addr; }
E → L	{ E.addr = new Temp(); gen(E.addr '=' L.base '[' L.addr ']'); }
E → E_1 + E_2	{ E.addr = new Temp(); gen(E.addr '=' E_1.addr + E_2.addr); }
S → id = E	{ gen(id.name '=' E.addr); }
S → L = E	{ gen(L.base '[' L.addr ']' '=' E.addr); }

Array Expressions

a[i][j][k][l]

Base = a

Offset/addr = i*w3*w2*w1+j*w2*w1+k*w1+l

IR for Array Expressions

- $L \rightarrow id\ [E] \mid L\ [E]$ // maintain three attributes: type, addr and base.

$L \rightarrow id\ [\ E\ ]$	{ L.type = id.type; L.addr = new Temp();
$L \rightarrow L_1\ [\ E\ ]$	{ t = new Temp(); L.addr = new Temp(); gen(t '=' E.addr '*' L.type.width); gen(L.addr '=' L_1.addr '+' t); }
$E \rightarrow id$	{ E.addr = id.addr; }
$E \rightarrow L$	{ E.addr = new Temp(); gen(E.addr '=' L.base '[' L.addr ']'); }
$E \rightarrow E_1 + E_2$	{ E.addr = new Temp(); gen(E.addr '=' E_1.addr + E_2.addr); }
$S \rightarrow id = E$	{ gen(id.name '=' E.addr); }
$S \rightarrow L = E$	{ gen(L.base '[' L.addr ']' '=' E.addr); }

Reading Exercise

- How to compute these widths
  - Section 6.3.4
  - Section 6.3.5

Next Lecture

- Intermediate code generation
  - Declarations
  - Type checking
  - Functions
  - Runtime environment