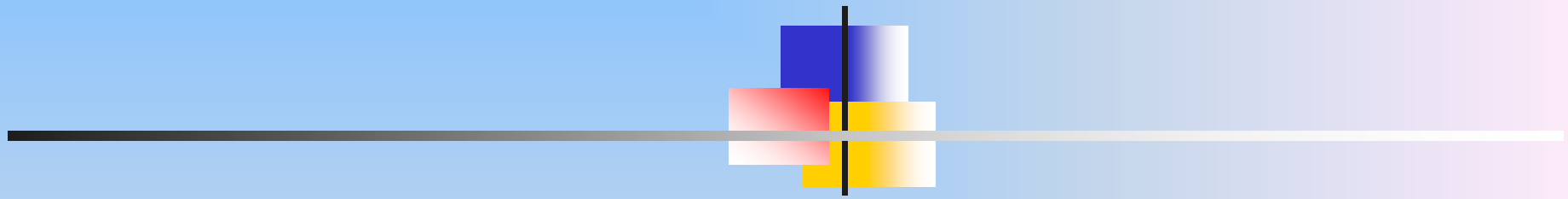
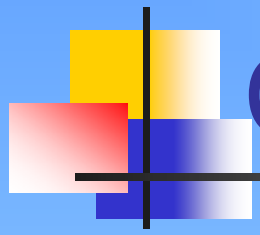
CSE302: Compiler Design



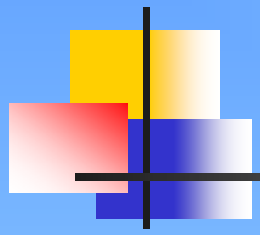
Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

April 10, 2007



Outline

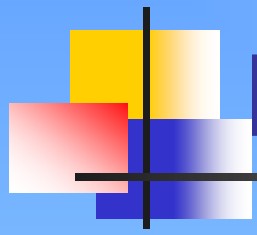
- Recap
- Intermediate code generation
- Summary and homework



Intermediate Representations

- Compiler front end
- Syntax tree
  - Directed acyclic graph (DAG) for expression
    - Before creating a new node, check whether an identical node already exists
- Three-address code
  - An address
    - A name: a pointer to its symbol-table entry
    - A constant
    - A compiler-generated temporary
  - Symbolic labels
    - L1, L2, ...

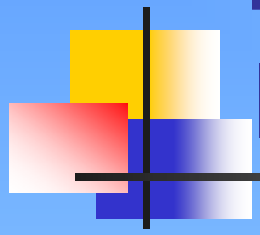
Three-Address Instruction Forms



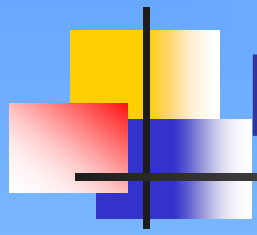
- x = y op z
- x = op y
- x = y
- goto L
- x = y[i]
- x[i] = y
- x = &y
- x = *y
- *x = y

- if x goto L and ifFlase x goto L
- if x relop y goto L
- Procedure calls and returns
  - param x_1
  - param x_2
  - ...
  - param x_n
  - call p, n or y = call p, n
  - return w

Data Structure Storing 3-Address Instructions

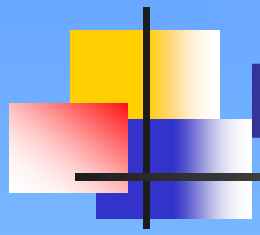


- Quadruples and Triples
  - op, arg1, arg2, result
  - op, arg1, arg2
  - Indirect triples
    - An instruction list of pointers to triples
    - An optimizing compiler can move an instruction by reordering the instruction list



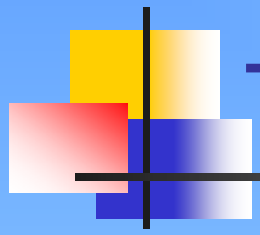
How to Translate Expressions?

- Assignment statements and expressions
  - S → id = E ;
    - S.code = E.code || gen(top.get(id.lexeme) ‘=’ E.addr)
  - E → E1 + E2
    - E.addr = new Temp()
    - E.code = E1.code || E2.code
      || gen(E.addr ‘=’ E1.addr ‘+’ E2.addr)
  - E → - E1 | (E1) | **id**
    - ?
  - a = b + - c
  - Incremental translation



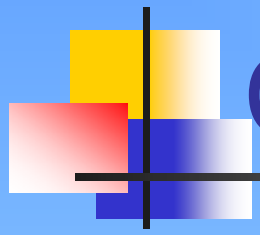
How to Address Array Elements?

- Row-major order with index starting at 0
  - A[i] locates at base+i×w
  - A[i1][i2] locates at base+i1×w1+i2×w2
    - int[2][3]: w1=12, w2=4
- L → **id** [E] | L [E]
- SDT for array references



Translation of Array References

- L→ **id** [E]
  - L.array = top.get(id.lexeme)
  - L.type = L.array.type.elem
  - L.addr = **new** Temp() //store offset value
  - gen(L.addr '=' E.addr '*' L.type.width)
- L→ L1 [E]
  - L.array = L1.array
  - L.type = L1.type.elem
  - t = **new** Temp()
  - L.addr = **new** Temp()
  - gen(t '=' E.addr '*' L.type.width)
  - gen(L.addr '=' L1.addr '+' t)
- S → id = E ; | L = E ;
- E → E1 + E2 | id | L



Outline

- Recap
- Intermediate code generation
- Summary