CODE GENERATION

UNNIKRISHNAN C

CODE GENERATION

- Instruction Selection
- Register Allocation and Assignment
- Instruction Reordering
- Transform basic blocks into more efficient basic blocks
- Transformation – Correctness an important factor
-

INPUTS TO THE CODE GENERATOR

- Intermediate Representation (IR)
- Symbol Table
  - To determine the dynamic addresses associated with data objects.
- IR
  - three-address code, virtual representation (byte-code)
  - stack-machine code, linear postfix notation, syntax tree, DAG

TARGET PROGRAM

- Reduced Instruction Set Architecture (RISC)
  - Many registers, three-address instructions, simple addressing modes
  - Simple Instruction Set Architecture (ISA)
- Complex Instruction Set Architecture (CISC)
  - Few registers, two address instructions, variety of addressing modes,
  - Several register classes, variable length instructions, instructions with side effects
- Stack-Based
  - Push Operands to stack then do operation on operand(s) on the stack.
  - Java Byte Codes, JIT

CODE GENERATOR

- Produces a relocatable code.
- This allows two subprograms to be compiled separately.
- Combines several relocatable codes and executed by a linking loader.

EXAMPLE ARCHITECTURE

- RISC
- Target ISA, IR and quality of the code important for the code generator
- Register Allocation
  - Allocates variables to reside in the register at each program point.
- Register Assignment
  - Pick a register for a variable
  - Register spilling may be required at some places.
- Optimization – For size, performance, power etc.

REGISTER ALLOCATION

- EXERCISE 1
- T = A+B
- T=T*C
- T=T/D

- EXERCISE2
- T=T+A
- T=T+C
- T=T/D

ARCHITECTURE OF INTEREST

- Three Address Code
- Operators
  - Load, Store, Compute, Jump, Conditional Jump
- Byte Addressable, N number of registers.
- OP TRGT, list_of_source_operands

ADDRESSING MODE

- A location can be indexed address of the form a(r) where
  - The symbol "a" corresponds to variable and r is a register
  - LD R1, a (R2) // R1= contents (a + contents (R2))
- A memory location can be integer indexed by a register.
  - LD R1, 100 (R2) // R1= contents (100+contents(R2))
- Indirect Addressing Mode (LD R1, *100 (R2))
  - R1= contents (contents (100+contents (R2)))
- Immediate Addressing Mode (LD R1, #100)
- Comments at the end of the instruction are preceded by //

OPERATIONS

- LD dst, addr (LD r1, x)
- ST dst, addr (ST x, r1)
- OP dst, src1, src2 (SUB (r1, r2,r3))
- Unary operator does not have src2.
- BR L (unconditional jump)
- Bcond r, L (BLTZ r1, L)

- Indexed Addressing mode
  - LD r1, A (r2)
  - LD r1, 100 (r2)
- Immediate Addressing Mode
  - LD r1, #100
  - ADD r1, r2, #100

INSTRUCTION COST

- Cost (instruction) = 1 + cost with addressing modes.
  - Corresponds to length in words of the instruction.
- Addressing mode with registers
  - have no additional cost.
- Addressing mode with memory location or constants
  - Additional cost of one (total cost two) for below operations.
  - LD R0, M
  - LD R1, * 100 (R2)

cost 3 because constant is not pa

(?) All arithmetic have cost 1 and load

FIND COST OF FOLLOWING CODE FRAGMENTS

- LD R0 , y
- LD R1, z
- ADD R0,R0,R1
- ST X,R0

2212

- LD R0, i
- MUL R0,R0,8
- LD R1, a (R0)
- ST b, R1

212 (becaues a is a variable ar

VALUE NUMBERING

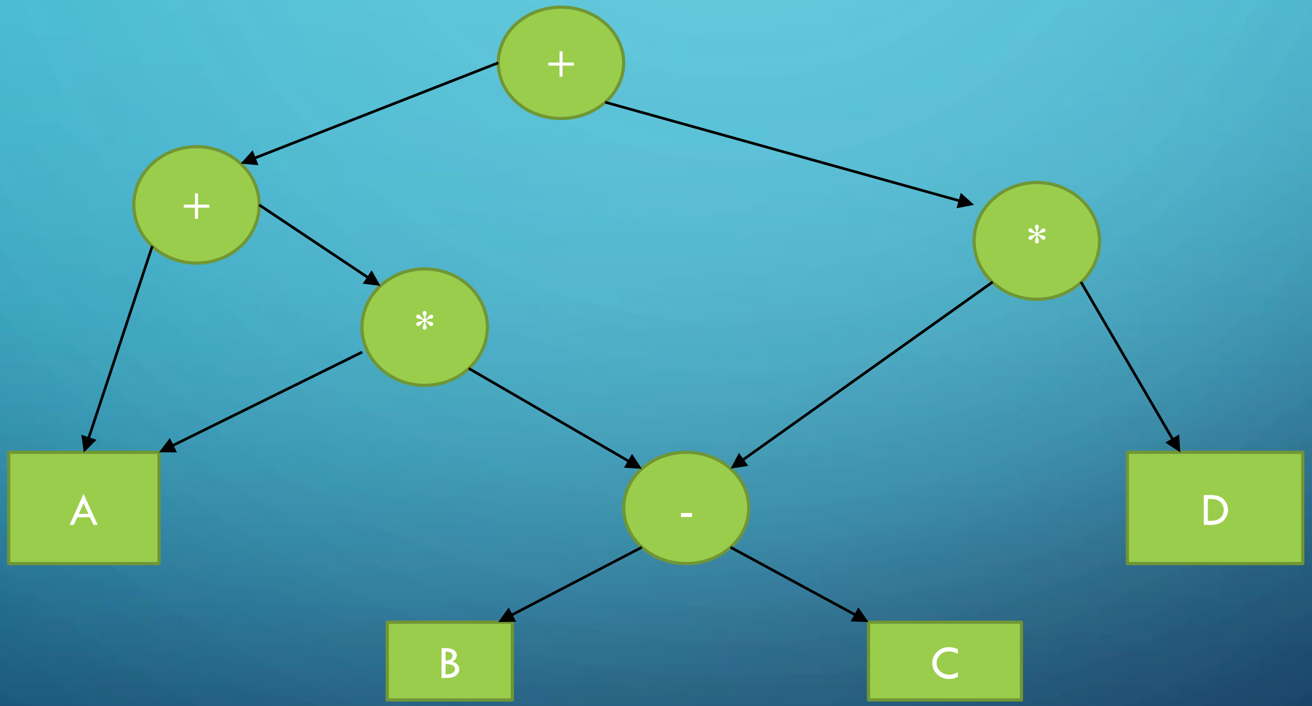
- i= i+10

Value No			
1	ID	Loc (i)	
2	NUM	10	
3	+	1	2
4	=	I	3

LOOPS

- Loop L contains a set of nodes, with a special node called e (entry node)
- e is not in ENTRY, the entry node of the entire CFG.
- No node inside L besides e has a predecessor outside L.
  - Every path from ENTRY to anynode in L goes through e.
- Every node in L has a non-empty path completely within L to e.
- Loops – {B2}, {B2,B3,B4}

IR- DAG (DIRECTED ACYCLIC GRAPH)



DATA FLOW ANALYSIS - SUMMARY

	Reaching Definition	**Live Variable**	**Available Expressions**
Domain	Sets of Definitions	Sets of variables	Sets of Expression
Direction	Forwards	Backwards	Forwards
Transfer Function	gen_B U (x- $kill_B$)	use_B U (x-def_B)	e_gen_B U(x- e_kill_B)
Boundary condition	OUT [ENTRY]= Ǿ	IN [EXIT] = Ǿ	OUT [ENTRY] = Ǿ
Meet ($\wedge$)	$\cup$	$\cup$	$\cap$

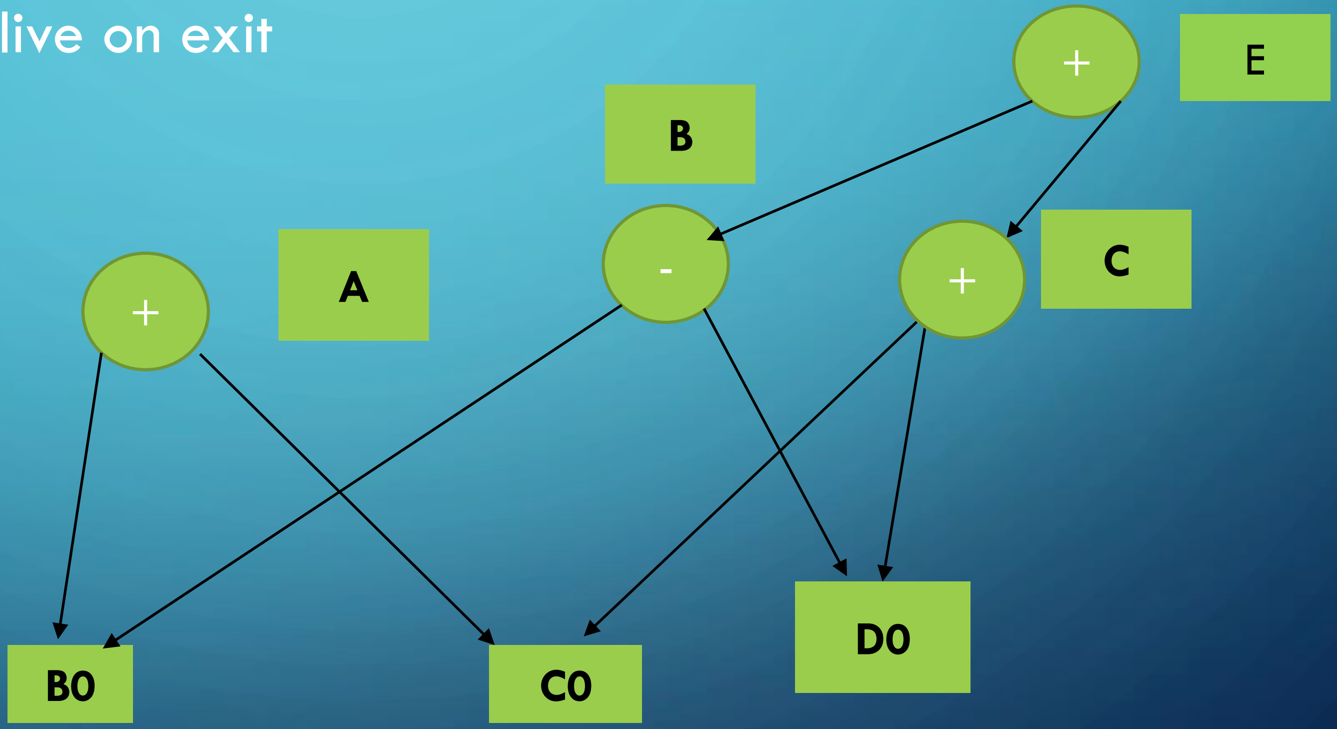
Equation	OUT[B]= FB (IN[B])
	IN[B]=$\wedge$ OUT[P] P ε pred(B)
		IN[B]= FB (OUT[B])
		OUT[B]=$\wedge$ OUT[S] S ε Succ B)
			OUT[B]= FB (IN[B])
			IN[B]=$\wedge$ OUT[P] P ε Pred(B)

Intialize	OUT [B]= Ǿ	IN[B]=Ǿ	OUT[B]= U

DEAD CODE ELIMINATION

Nodes E, C can be eliminated.

- i) a, b – live && c,e not live on exit
- Remove deadcode



SIMPLYFIY THREE ADDRESS CODE

- i) A is live on exit
- ii) A and B are live on exit
- Code
    - D=B*C
    - E=A+B
    - B=B+C
    - A=E-D

For i) we can eliminate B = B + C.For ii) Non can be elim

POINTER ASSIGNMENT - ISSUES

- X=*p
- *q=y

p might be dangling/null pointer.