A Simple Code Generator

- Generates code for a single basic block, which considers each three-address instruction and keeps track of what values are in what registers so it can avoid generating unnecessary loads and stores.
- One of the primary issues during code generation is deciding how to use registers to best advantage. There are four principal uses of registers:
  - In most machine architectures, some or all of the **operands of an operation must be in registers** in order to perform the operation.
  - **Registers make good temporaries, places to hold the result of a sub expression** while a larger expression is being evaluated, or more generally, a place to hold a variable that is used only within a single basic block.
  - **Registers are used to hold (global)values** that are computed in one basic block and used in other blocks, for example, a loop index that is incremented going around the loop and is used several times within the loop.
  - Registers are often used to help with **run-time storage management**, for example, to manage the run-time stack.

A Simple Code Generator

SINI ANNA ALEX

A Simple Code Generator

- Generates code for a single basic block, which considers each three-address instruction and keeps track of what values are in what registers so it can avoid generating unnecessary loads and stores.
- One of the primary issues during code generation is deciding how to use registers to best advantage. There are four principal uses of registers:
  - In most machine architectures, some or all of the **operands of an operation must be in registers** in order to perform the operation.
  - **Registers make good temporaries, places to hold the result of a sub expression** while a larger expression is being evaluated, or more generally, a place to hold a variable that is used only within a single basic block.
  - **Registers are used to hold (global)values** that are computed in one basic block and used in other blocks, for example, a loop index that is incremented going around the loop and is used several times within the loop.
  - Registers are often used to help with **run-time storage management**, for example, to manage the run-time stack.

A Simple Code Generator contd…

- Our code-generation algorithm considers each **three-address instruction** and **decides what loads are necessary** to get the needed operands into registers.
- After generating the loads, it **generates the operation itself**. Then, if there is a need to store the result into a memory location, it also generates that store.
- In order to make the needed decisions, we **require a data structure** that tells us what program variables currently have their value in a register, and which register or registers, if so.
- We also need to know whether the **memory location for a given variable currently has the proper value for that variable**, since a new value for the variable may have been computed in a register and not yet stored.

Data structure has the following descriptors: Register and Address Descriptors

- 1. For each available register, a **register descriptor** keeps track of the variable names whose current value is in that register. Since we shall use only those registers that are available for local use within a basic block.
  - we assume that initially, all register descriptors are empty. As the code generation progresses, each register will hold the value of zero or more names.
- 2. For each program variable, an **address descriptor** keeps track of the location or locations where the current value of that variable can be found.
  - The location might be a register, a memory address, a stack location, or some set of more than one of these. The information can be stored in the symbol-table entry for that variable name.

Function getReg(I)

- Function getReg(I), which selects registers for each memory location associated with the three-address instruction I.
- Function getReg has access to the register and address descriptors for all the variables of the basic block, and may also have access to certain useful data-flow information such as the variables that are live on exit from the block.

Machine Instructions for Operations

- **For a three-address instruction such as x = y + z, do the following:**

1. Use getReg (x = y + z) to select registers for x, y, and z. Call these Rx, Ry, and Rz.
2. If y is not in Ry (according to the register descriptor for Ry), then issue an instruction LD Ry , y' , where y' is one of the memory locations for y (according to the address descriptor for y).
3. Similarly, if z is not in Rz, issue an instruction LD Rz , z' , where z' is a location for z.
4. Issue the instruction ADD Rx , Ry , Rz.

Managing Register and Address Descriptors

- As the code-generation algorithm issues load, store, and other machine instructions, it needs to update the register and address descriptors. The rules are as follows:

1. For the instruction LD R,x

   (a)Change the **register descriptor** for register R so it holds only x.

   (b) Change the **address descriptor** for x by adding register R as an additional location.

2. For the Instruction ST x,R, change the **address descriptor** for x to include its own memory location.

3. For an operation such as ADD Rx,Ry,Rz implementing a three-address instruction x = y + z

   (a) Change the **register descriptor** for Rx so that it holds only x.

   (b) Change the **address descriptor** for x so that its only location is Rx.

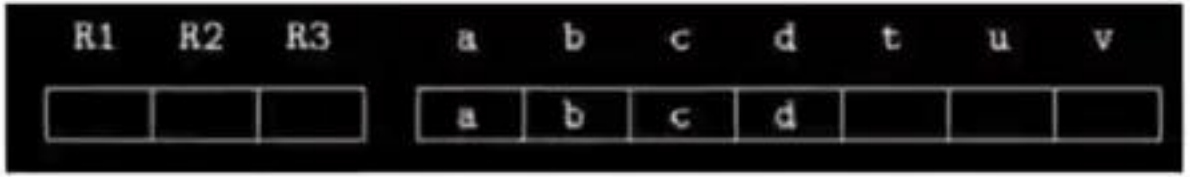
   (c) Remove Rx from the **address descriptor** of any variable other than x.

The register and address descriptors before and after the translation of each three-address instruction

```
t = a - b
u = a - c
```

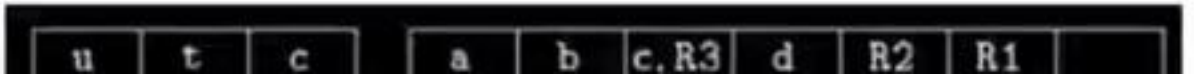
Register descriptor

Address descriptor




```
t = a - b
    LD R1, a
    LD R2, b
    SUB R2, R1, R2
```


```
u = a - c
    LD R3, c
    SUB R1, R1, R3
```



Managing Register and Address Descriptors

- As the code-generation algorithm issues load, store, and other machine instructions, it needs to update the register and address descriptors. The rules are as follows:

1. For the instruction LD R,x

   (a)Change the **register descriptor** for register R so it holds only x.

   (b) Change the **address descriptor** for x by adding register R as an additional location.

2. For the instruction ST x,R, change the **address descriptor** for x to include its own memory location.

3. For an operation such as ADD Rx,Ry,Rz implementing a three-address instruction x = y + z

   (a) Change the **register descriptor** for Rx so that it holds only x.

   (b) Change the **address descriptor** for x so that its only location is Rx.

   (c) Remove Rx from the **address descriptor** of any variable other than x.

The register and address descriptors before and after the translation of each three-address instruction

```
t = a - b
u = a - c
```

Register descriptor

R1	R2	R3

Address descriptor

a	b	c	d	t	u	v
a	b	c	d			

```
t = a - b
      LD R1, a
      LD R2, b
      SUB R2, R1, R2
```

a	t	
a, R1	b	c	d	R2		
---	---	---	---	---	---	---

```
u = a - c
      LD R3, c
      SUB R1, R1, R3
```

u	t	c
a	b	c, R3	d	R2	R1	
---	---	---	---	---	---	---

Managing Register and Address Descriptors

- As the code-generation algorithm issues load, store, and other machine instructions, it needs to update the register and address descriptors. The rules are as follows:

1. For the instruction LD R,x

   (a)Change the **register descriptor** for register R so it holds only x.

   (b) Change the **address descriptor** for x by adding register R as an additional location.

2. For the instruction ST x,R, change the **address descriptor** for x to include its own memory location.

3. For an operation such as ADD Rx,Ry,Rz implementing a three-address instruction x = y + z

   (a) Change the **register descriptor** for Rx so that it holds only x.

   (b) Change the **address descriptor** for x so that its only location is Rx.

   (c) Remove Rx from the **address descriptor** of any variable other than x.

The register and address descriptors before and after the translation of each three-address instruction

```
t = a - b
u = a - c
```

Register descriptor

Address descriptor

R1	R2	R3

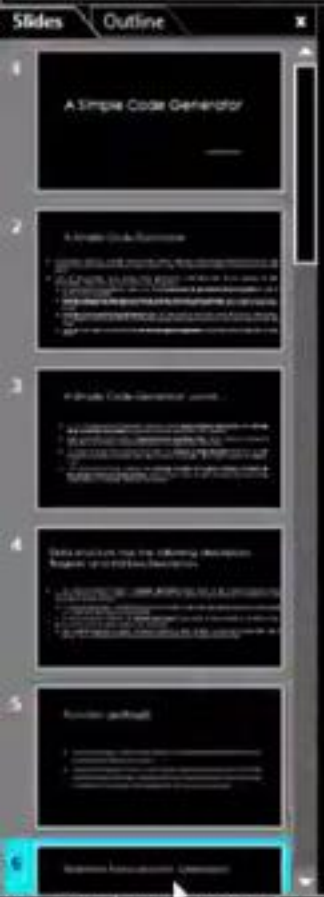
a	b	c	d	t	u	v
a	b	c	d			

```
t = a - b
    LD R1, a
    LD R2, b
    SUB R2, R1, R2
```

a	t	
a, R1	b	c	d	R2		
---	---	---	---	---	---	---

```
u = a - c
    LD R3, c
    SUB R1, R1, R3
```

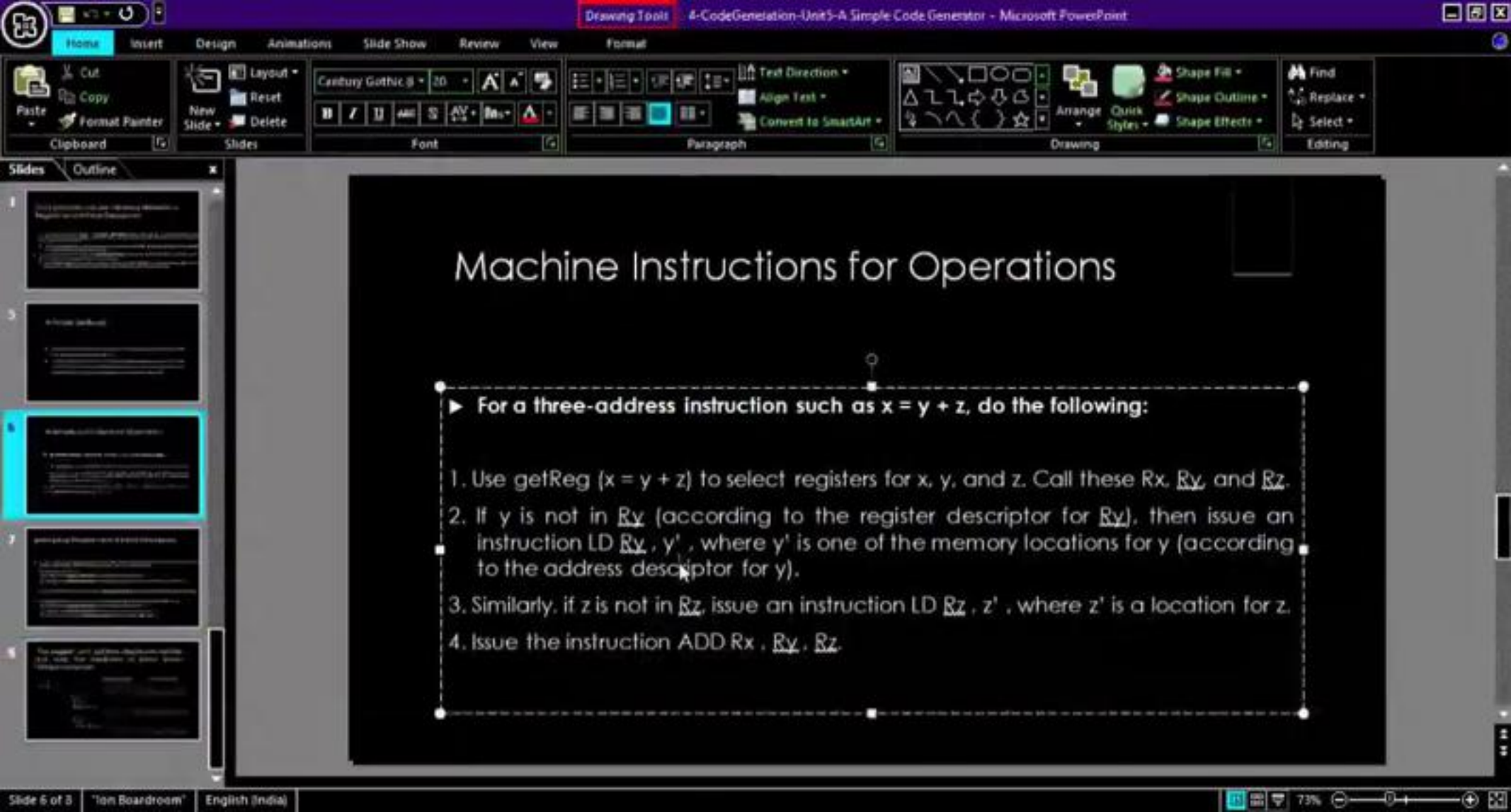
u	t	c
a	b	c, R3	d	R2	R1	
---	---	---	---	---	---	---



Machine Instructions for Operations

- **For a three-address instruction such as x = y + z, do the following:**

1. Use getReg (x = y + z) to select registers for x, y, and z. Call these Rx, Ry, and Rz.
2. If y is not in Ry (according to the register descriptor for Ry), then issue an instruction LD Ry , y' , where y' is one of the memory locations for y (according to the address descriptor for y).
3. Similarly, if z is not in Rz, issue an instruction LD Rz , z' , where z' is a location for z.
4. Issue the instruction ADD Rx , Ry , Rz.


Drawing Tools
4-CodeGeneration-Unit5-A Simple Code Generator - Microsoft PowerPoint
Home
Insert
Design
Animations
Slide Show
Review
View
Format
Cut
Copy
Format Painter
Paste
Clipboard
New Slide
Layout
Reset
Delete
Slides
Century Gothic
20
Font
Text Direction
Align Text
Convert to SmartArt
Paragraph
Arrange
Quick Styles
Shape Fill
Shape Outline
Shape Effects
Drawing
Find
Replace
Select
Editing
Slides
Outline
Machine Instructions for Operations
For a three-address instruction such as x = y + z, do the following:
1. Use getReg (x = y + z) to select registers for x, y, and z. Call these Rx, Ry, and Rz.
2. If y is not in Ry (according to the register descriptor for Ry), then issue an instruction LD Ry , y' , where y' is one of the memory locations for y (according to the address descriptor for y).
3. Similarly, if z is not in Rz, issue an instruction LD Rz , z' , where z' is a location for z.
4. Issue the instruction ADD Rx , Ry , Rz.
Slide 6 of 8
"Ion Boardroom"
English (India)
73%

Machine Instructions for Operations

- **For a three-address instruction such as x = y + z, do the following:**

1. Use getReg (x = y + z) to select registers for x, y, and z. Call these Rx, Ry, and Rz.
2. If y is not in Ry (according to the register descriptor for Ry), then issue an instruction LD Ry , y' , where y' is one of the memory locations for y (according to the address descriptor for y).
3. Similarly, if z is not in Rz, issue an instruction LD Rz , z' , where z' is a location for z.
4. Issue the instruction ADD Rx , Ry , Rz.