CISC Design

Hardware Flowchart

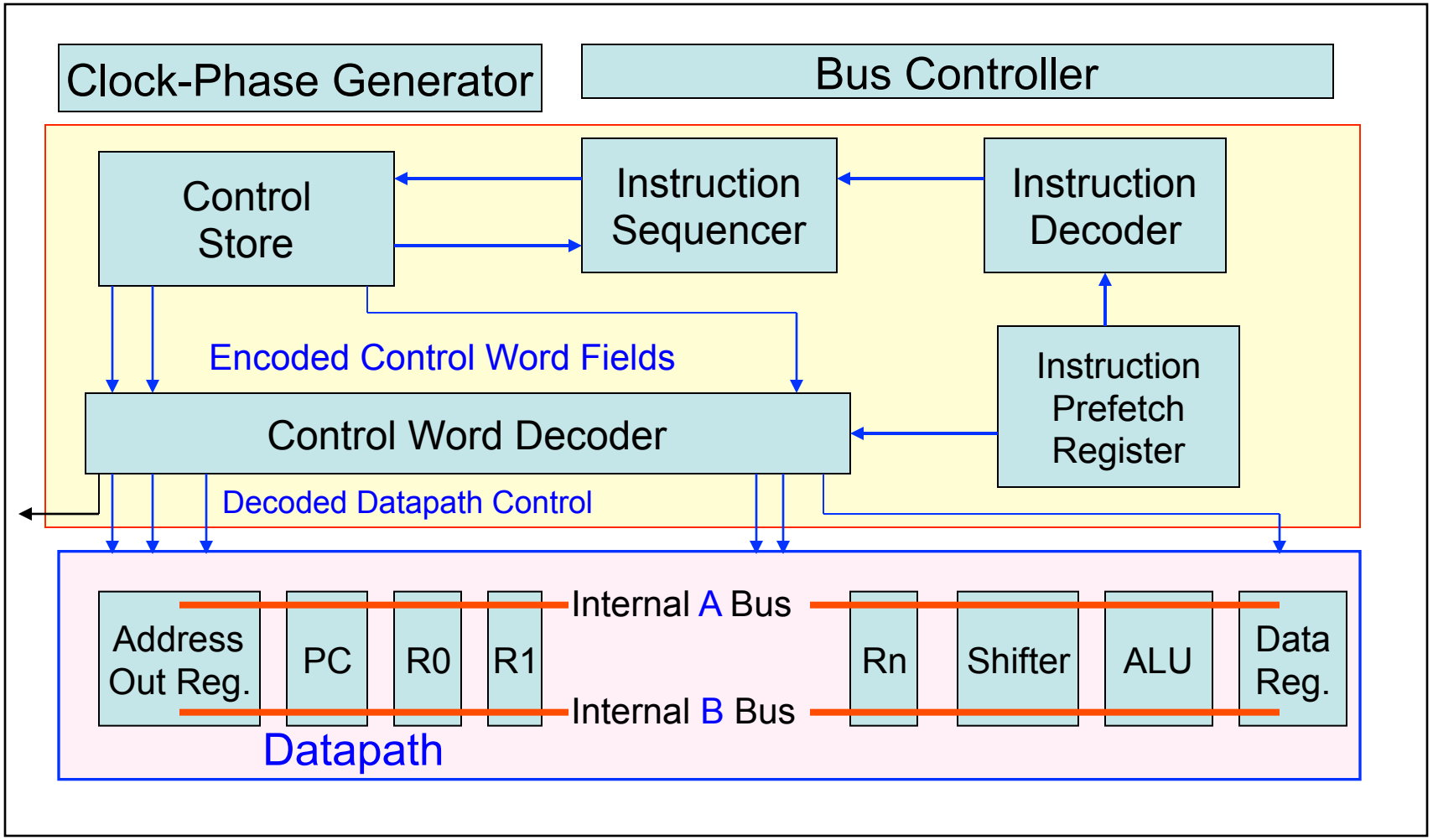
Virendra Singh
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

EE-309: Microprocessors

Lecture 22 (14 Sep 2015)

CADSL

Micro-coded Implementation



Hardware Flowchart

Hardware Flowchart

Hardware Flowchart

- Tells us how to get from the architecture to the implementation
- Links programmer's (external) model and the hardware (internal) implementation
- Specify exactly how commands from the instruction set are carried out using Datapath

Flowchart Objective

- Limit controller size to some fraction of chip area
- Make CPU as fast as possible
- Complete the project as early as possible
- Make the flowcharts easy to translate into hardware

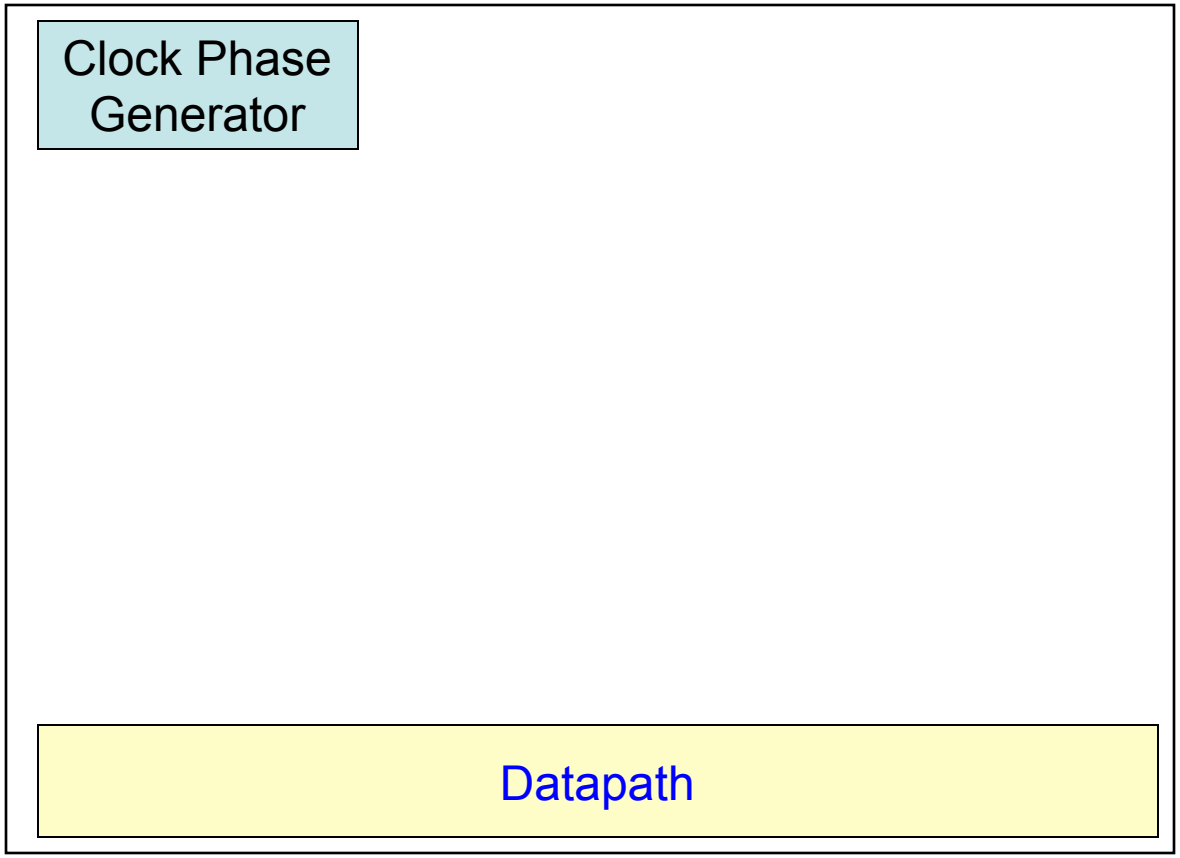
Hardware Flowchart

Pre-requisites:

- Instruction set summary
  - Instruction Format
  - Operations
  - Addressing modes
  - Registers
- Datapath
  - Programmer's register set
  - Additional registers
  - ALU and any special functional units
  - Internal data paths
  - Rules of operation

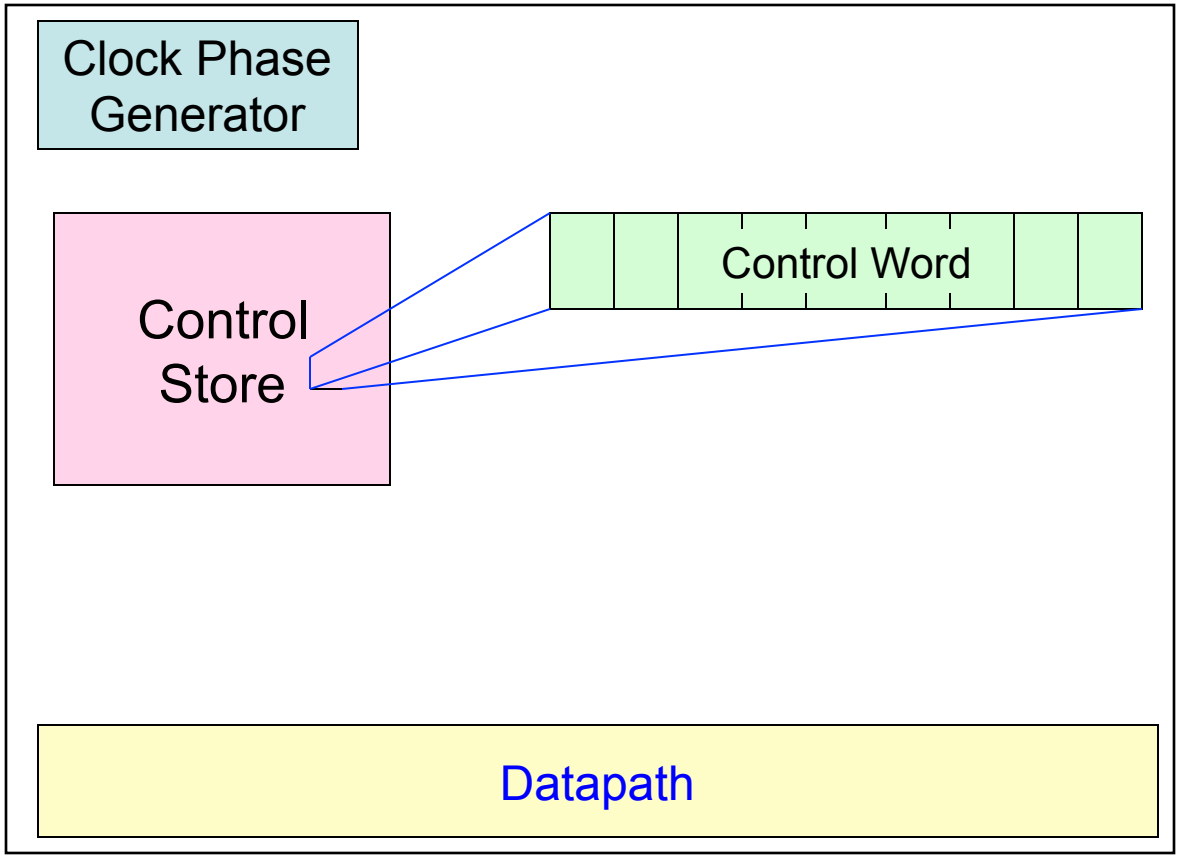
- Datapath will evolve

Development of Implementation



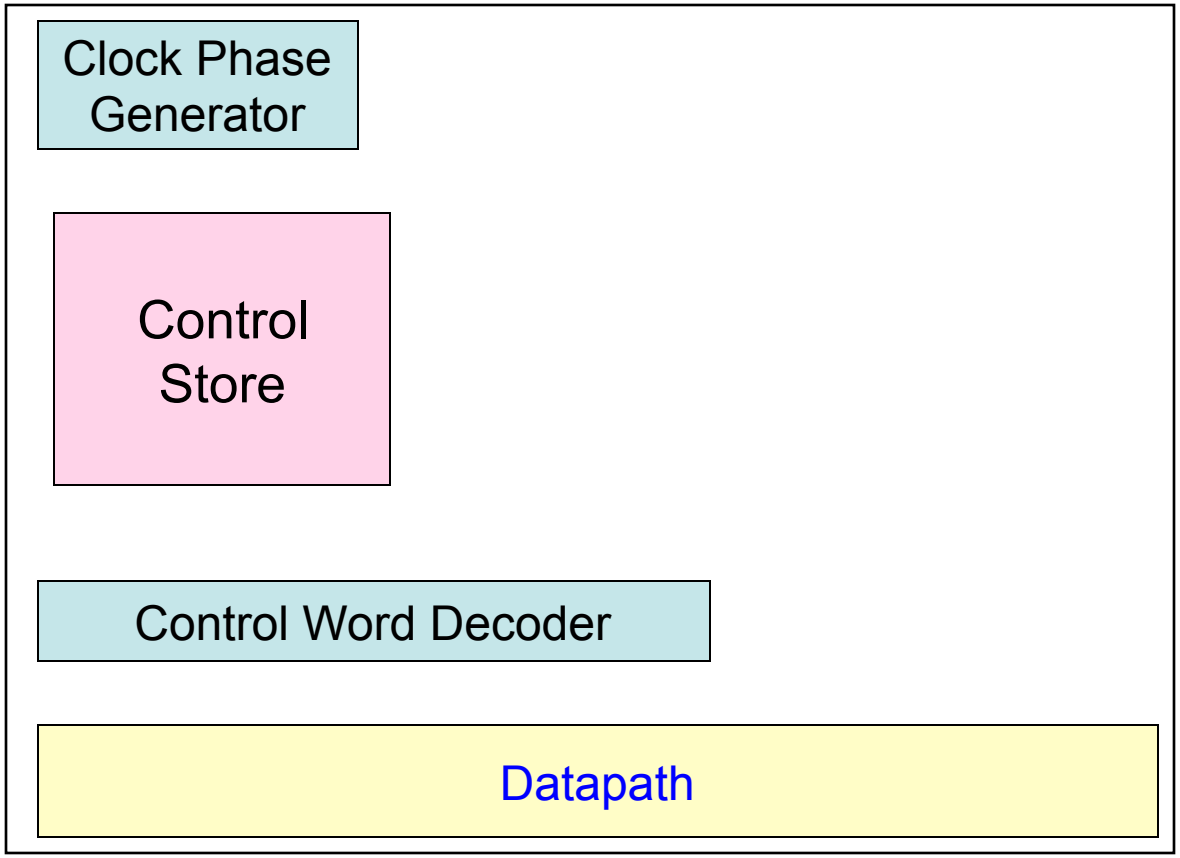
- The architecture specification is the only input
- Begin with a guess of Datapath
- Do flowchart for the instructions
- This modifies and refines the Datapath and develops the control store and control strategy
- The final Datapath is derived output

Development of Implementation



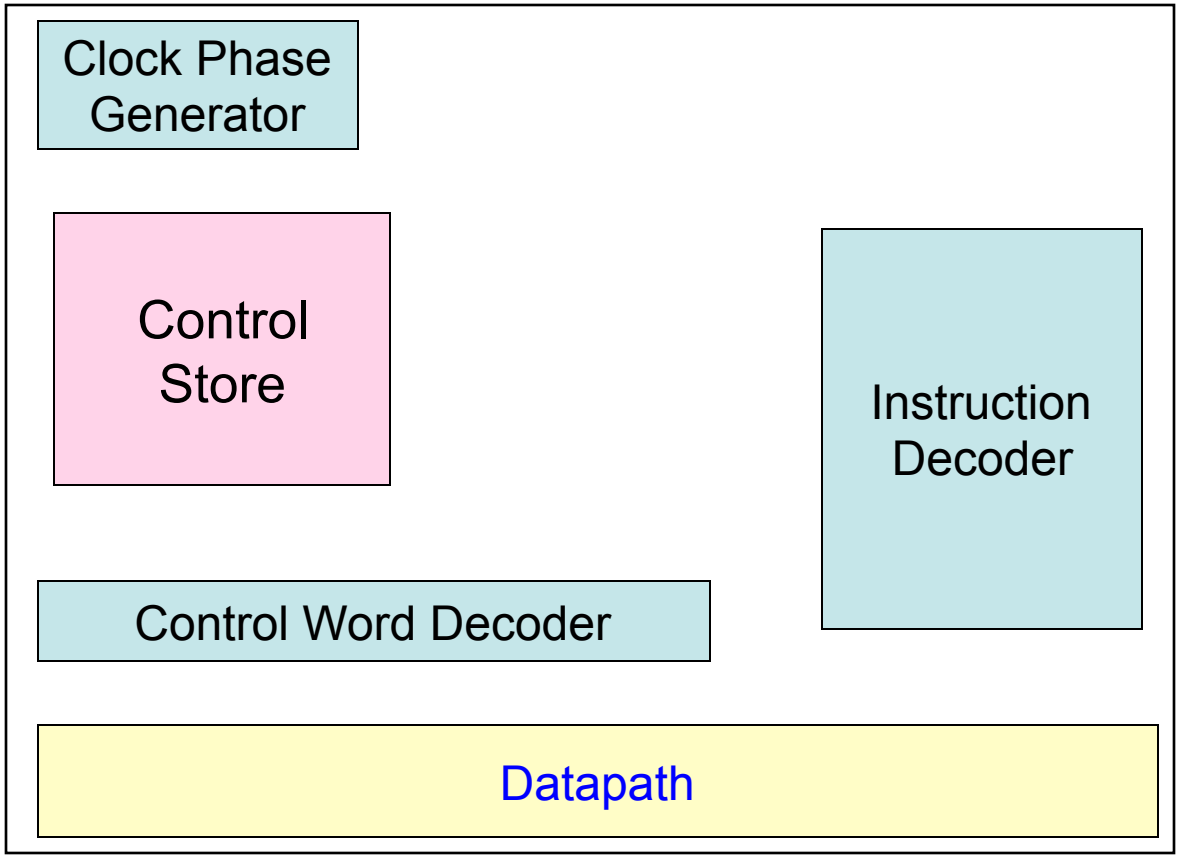
- Once flowcharts are fairly complete, derive the control word format using the flowchart states
- When the flowcharts are complete, so is the Datapath
- Control word format is derived output

Development of Implementation



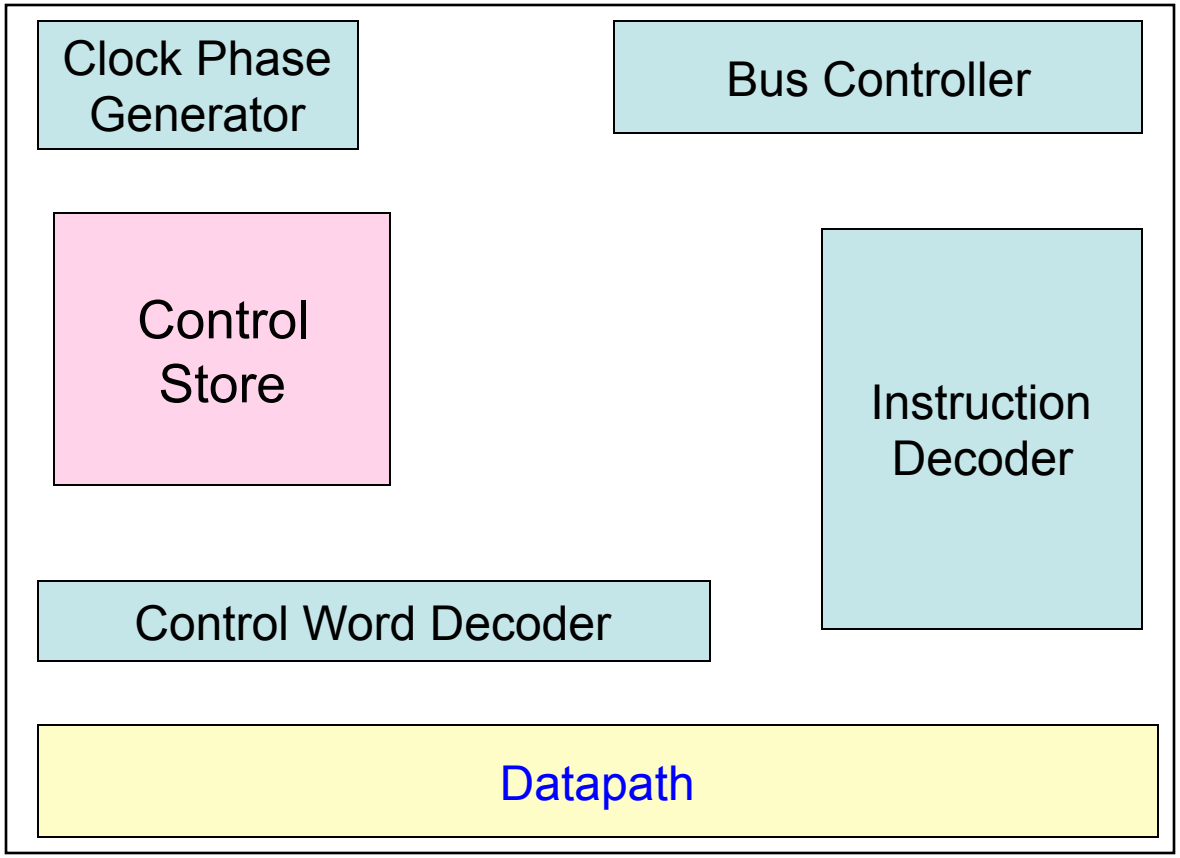
- After defining control word format, you assign bit patterns to the control fields in a way that minimizes control word decoders between the control store and the Datapath

Development of Implementation



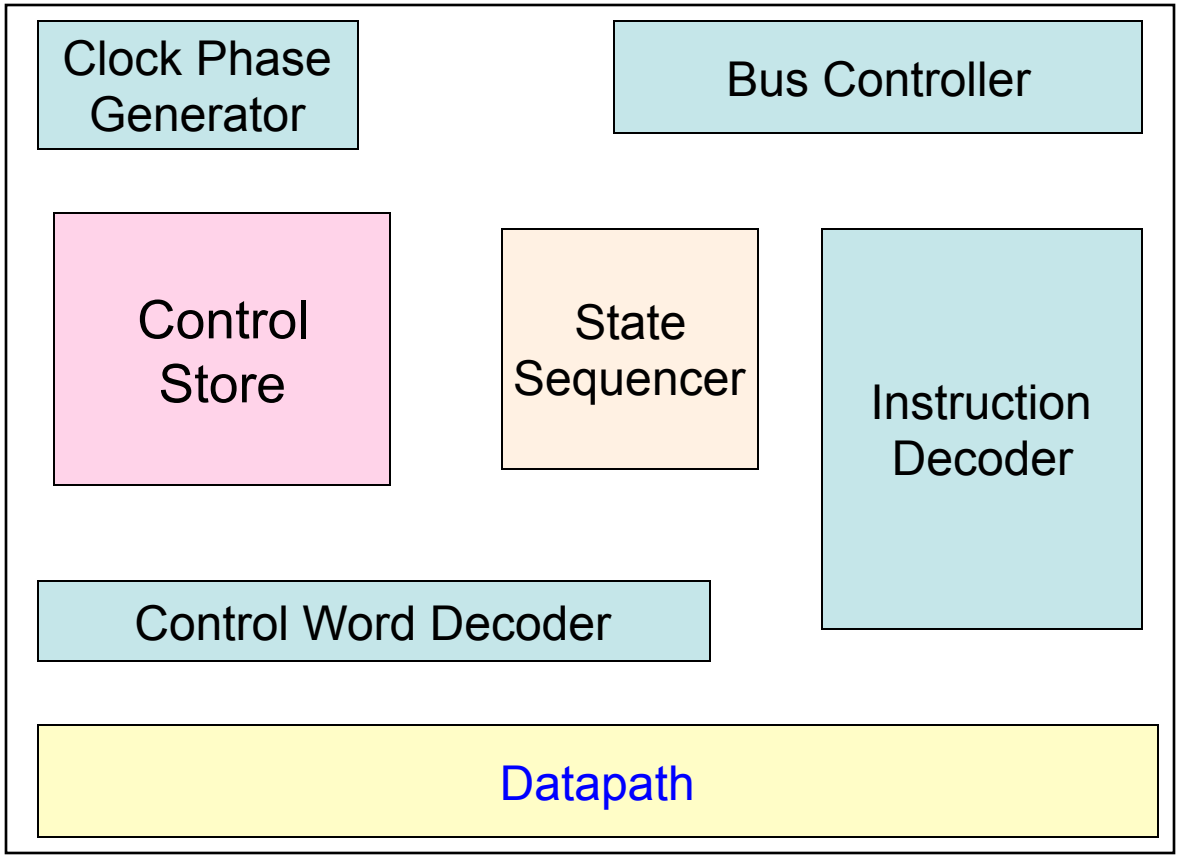
- Instruction decoders are defined the flowcharts and the architecture specification

Development of Implementation



- Completed flowcharts, control word format, and the initial bus specification defines the bus controller

Development of Implementation



- Last is the logic of the state sequencer, the part of the chip that says what to do next (where is the next control word?)
- Once every thing around it is defined, you build exactly what you need! (The state sequencer is derived output)

MIN Instruction Set

- ✓ ADD
- ✓ AND
- ✓ BZ – Branch if zero bit is set. (Register Indirect only)
- ✓ LOAD – Second operand is source and Rx is destination
- ✓ POP – Postincrement with register indirect only
- ✓ PUSH – Predecrement with register indirect only
- ✓ STORE
- ✓ SUB
- ✓ TEST

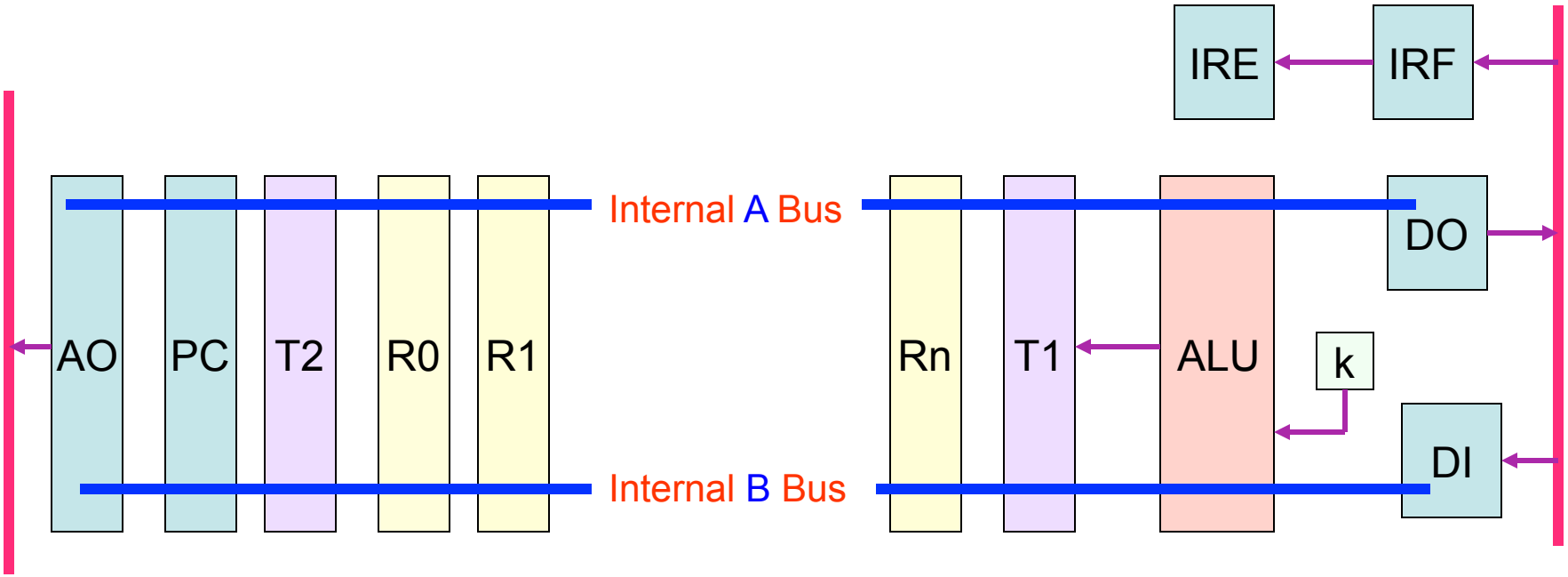
MIN Instruction Set

Second Operand Address

Address Modes

- AB - Base (Ry) plus displacement (second instruction word) is an operand address
- AI – Register indirect. Ry holds an operand address
- AR – Register direct: The result is stored in Ry. For two operand instructions, Ry also is an operand source

MIN Datapath



MIN Datapath

Rules of Operation

1. A transfer from source to bus to destination takes one state time
2. A source can drive up to three destination loads
3. Inputs to the ALU are from A (internal) bus and either k (values 0, +1, -1) or the B (internal) bus
4. When ALU is destination. T1 is automatically loaded from the ALU output
5. A transfer to AO activates the on-chip external bus controller. This bus controller postpones the next state until the external transfer is complete.

Flowcharts

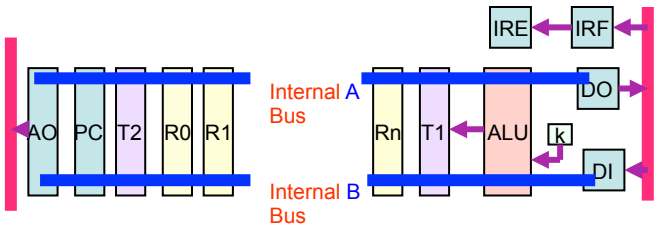
ADD RX AR RY

Register-to-Register

R → R ADD

rx → a → alu
ry → b → alu

t1 → b → ry



ADD RX AI (RY)

Register-to-Memory

R → M ADD

edb → di
ry → b → ao

State

di → b → alu
rx → a → alu

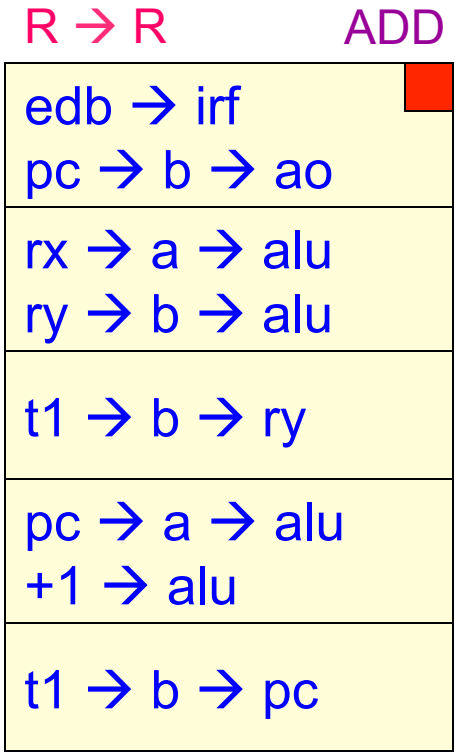
ry → b → ao
t1 → a → do

Sequence

Flowcharts

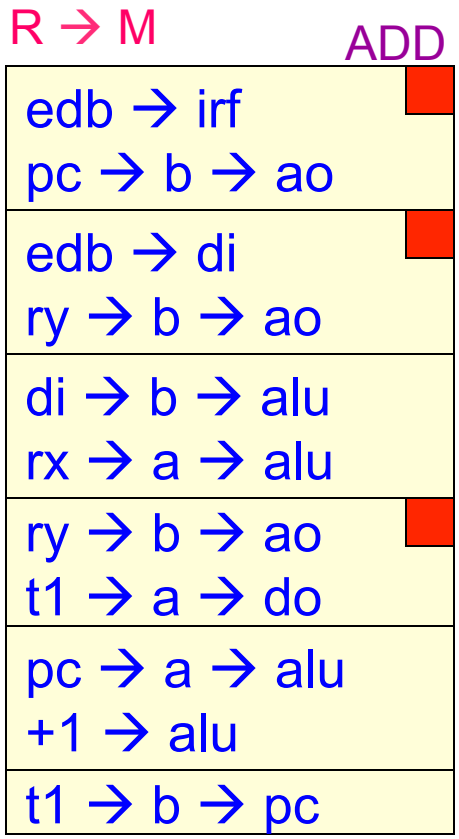
ADD RX AR RY

Register-to-Register



ADD RX AI (RY)

Register-to-Memory



- Execution Speed

Thank You