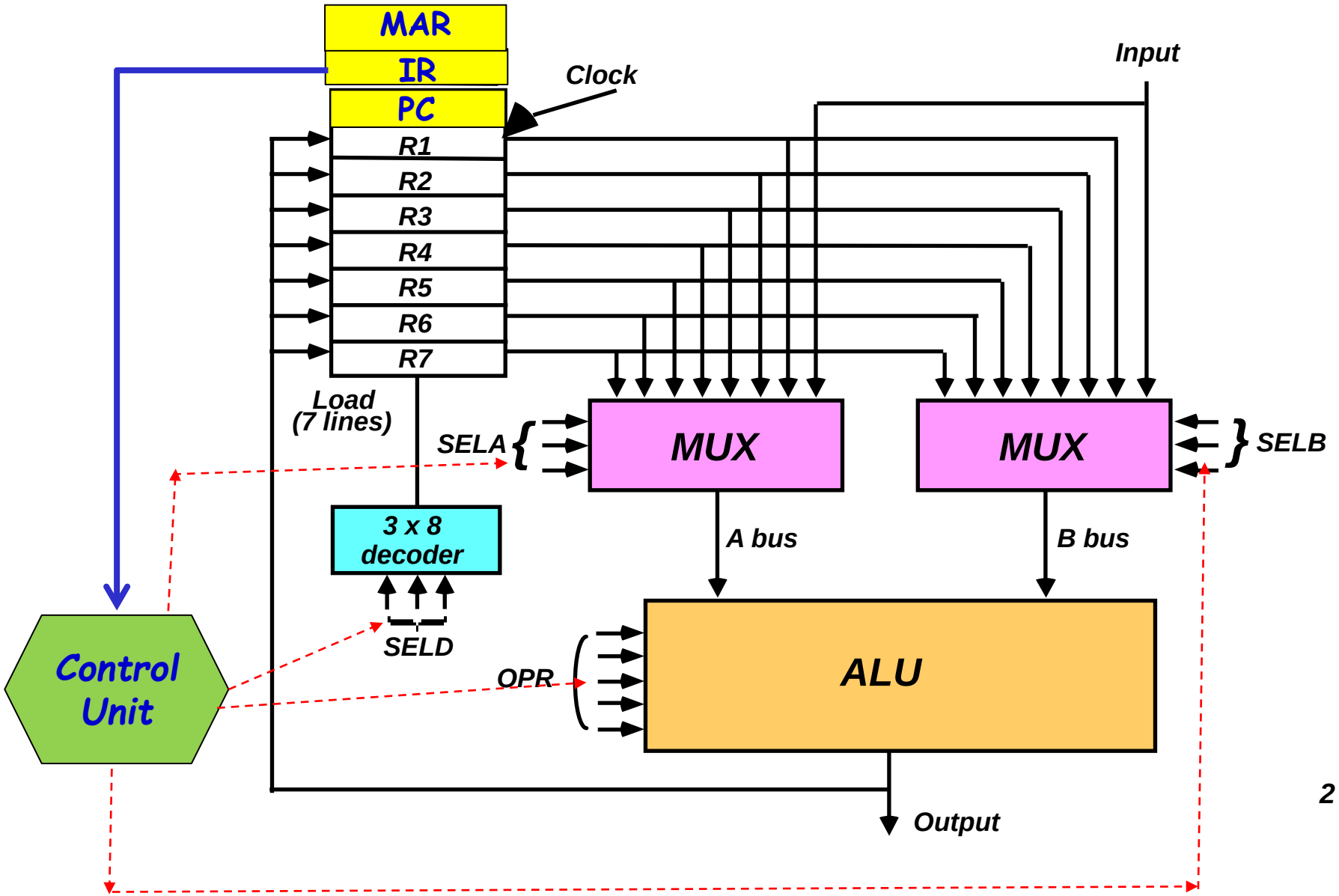
Lecture 6

21-01-2021

General Processor Organization



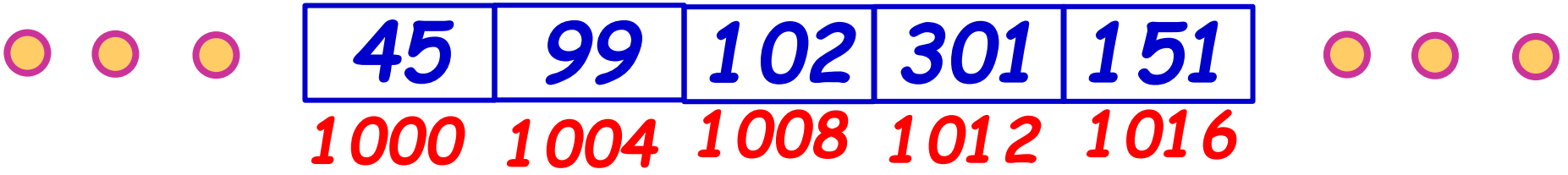
2

Review of Addressing Modes

- **Register addressing-** Operand is in register
  - Example: **Add R1,R2,R3**
  - **If R2 contains 100, i.e. (R2)=100**
  - **(R3)=200**
  - **Then after Add, (R1)=300**

Review of Addressing Modes

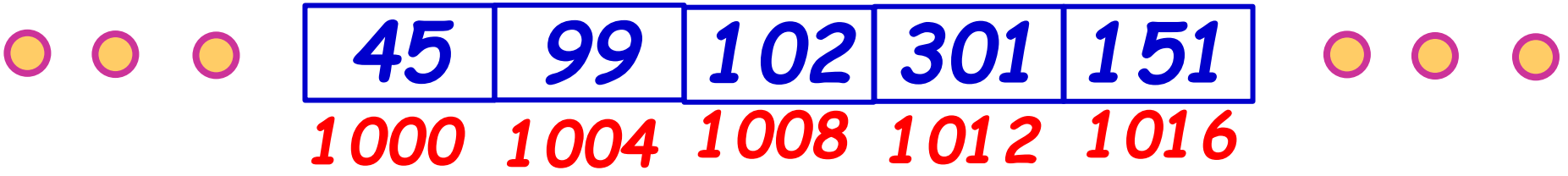
- **Memory Direct Address (aka absolute addressing)**– Operand in memory at given address
  - Example: **Load R1, 1004**



  - **After instruction execution, (R1)=99**

Review of Addressing Modes

- **Register Indirect Addressing** – Register contains memory address of operand
  - Example: **Load R1, (R2)**



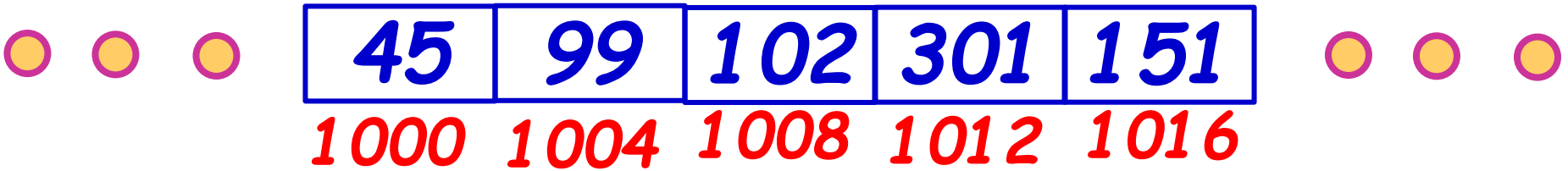
  - **If (R2)=1012, then (R1)=301 after instruction execution**

Common Addressing Modes

- **Indexed Address (Displacement)**- Add base and register value to get address of operand in memory

  Offset **Base Register**

  - Example **Load R1, 1000(R2)**



  - **If (R2)=4, then (R1)=99 after execution**

Common Addressing Modes

- **PC relative address** - Address from instruction is added to the current value in the Program Counter
  - **Example** **J -16** - Jumps to address PC-16
  - If current value of PC is 1016, then PC value will be set to 1000

Some Questions

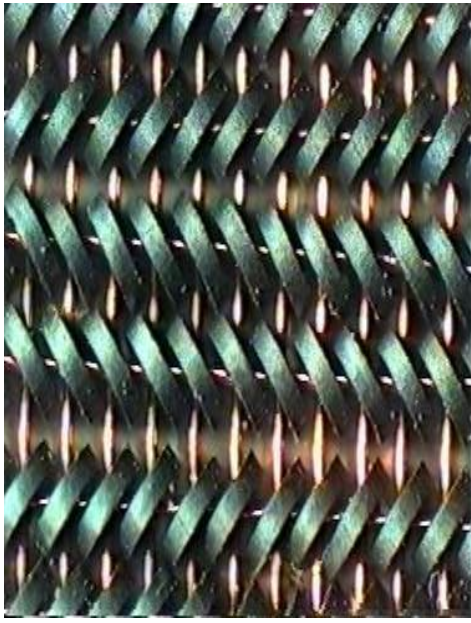
- How come people suddenly realized in 1984 that RISC is good?

Was CISC a historical blunder?

- Were the people who designed processors before 1984 morons?

CISC Era: Flashback

- Critical parts of programs were written in assembly language:
  - Compilers were hard to write --- especially for machines with GP registers
  - Magnetic core memory was used as main memory --- slow and expensive:
  - **Imperative to minimize code size…**
  - Instructions with memory operands (memory-to-memory) were preferred



1980s… Technology was advancing…

- **Compilers were improving:**
  - Compiler writing tools became available
  - Optimizing compilers could effectively use registers
- **Caches came into being:**
  - Allowed main memory to be accessed at speeds comparable to control memory
- **Semiconductor memory was replacing magnetic core memory:**
  - Reduced performance gap between control and main memory

RISC Philosophy

- The rarely used instructions can be eliminated to save chip space:

What is the best way to use this saved chip area?

  - On-chip cache and large number of registers can be provided.

  - **What would be the advantages of this?**

Summary of a RISC Processor

- Small number of instructions
- Small number of addressing modes, **Load/Store**
- **Large number of registers** (>=32)
- Instructions execute in one or two clock cycles
- Uniform length instructions and fixed instruction format.
- **Register-Register Architecture:**
  - Separate memory instructions (load/store)
- Separate instruction/data cache
- **Hardwired control**
- **Pipelining** (Why CISC are not pipelined?)

Load/Store Architecture

- Memory access:
  - Through 2 dedicated instructions --- **load and store** --- unlike CISC
- All operations are performed only using operands present in registers:
  - **Memory accessed using load/store instructions only.**

Implementation Issues

- **A CISC processor devotes about half of the chip area to the control unit.**
  - A RISC processor typically uses only about 10% of the area for the control unit, devoting the rest to registers and caches instead.
- Reduced processor design time.
  - The simple control unit and circuitry of RISC result in faster design cycles.

Why Does RISC Lead to Improved Performance?

- Larger number of registers (visible and invisible)
  - **Also much larger cache**
  - **Leads to decreased data traffic to memory.**
  - **Remember memory is the bottleneck.**
- Register-Register architecture leads to more uniform instructions:
  - **Efficient pipelining becomes possible.**
- However, increased instruct memory traffic:
  - Because of larger number of instructions .