ADDRESSING MODES AND PIPELINING

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

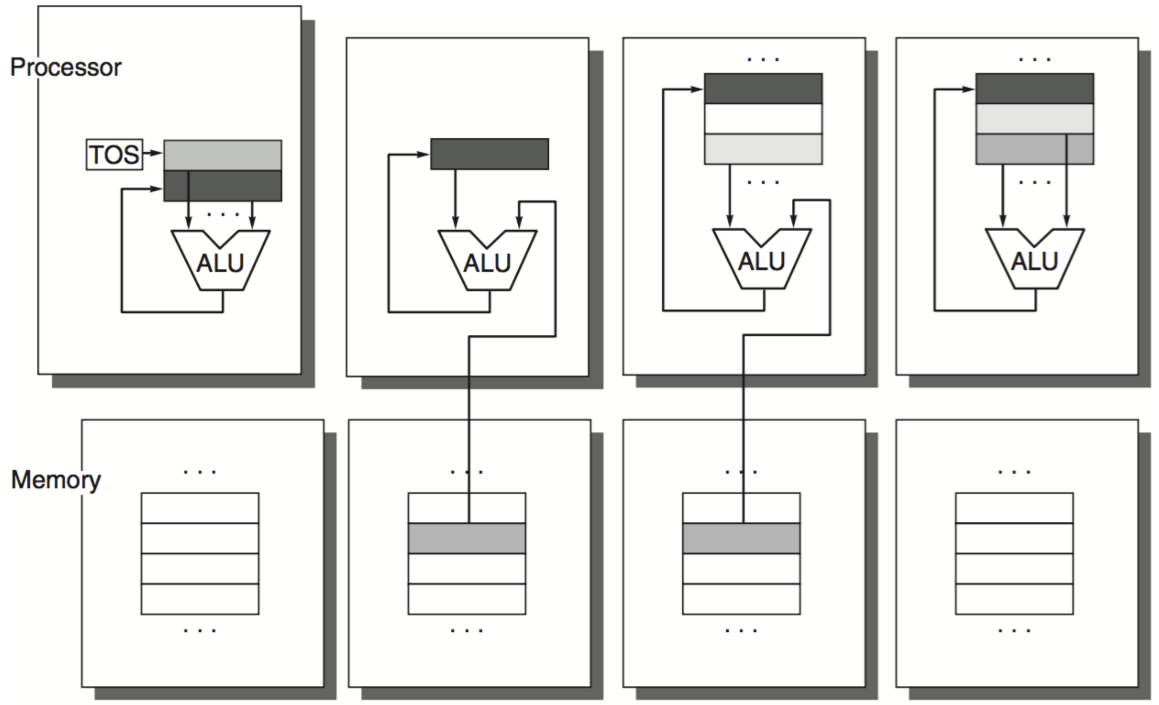
Overview

- Announcement
  - Tonight: Homework 1 release (due on Sept. 4th)
    - Verify your uploaded files before deadline

- This lecture
  - RISC vs. CISC
  - Addressing modes
  - Pipelining

ISA Types

- Operand locations



(a) Stack (b) Accumulator (c) Register-memory (d) Register-register/ load-store

Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

Which Set of Instructions?

- ISA influences the execution time
  - CPU time = IC x CPI x CT
- Complex Instruction Set Computing (CISC)
- Reduced Instruction Set Computing (RISC)

Which Set of Instructions?

- ISA influences the execution time
  - CPU time = IC x CPI x CT
- Complex Instruction Set Computing (CISC)
  - May reduce IC, increase CPI, and increase CT
  - CPU time may be increased
- Reduced Instruction Set Computing (RISC)
  - May increases IC, reduce CPI, and reduce CT
  - CPU time may be decreased

RISC vs. SISC

RISC ISA

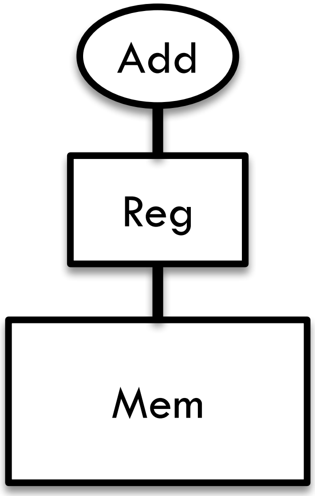
- Simple operations
  - Simple and fast FU
- Fixed length
  - Simple decoder
- Limited inst. formats
  - Easy code generation

CISC ISA

- Complex operations
  - Costly memory access
- Variable length
  - Complex decoder
- Limited registers
  - Hard code generation

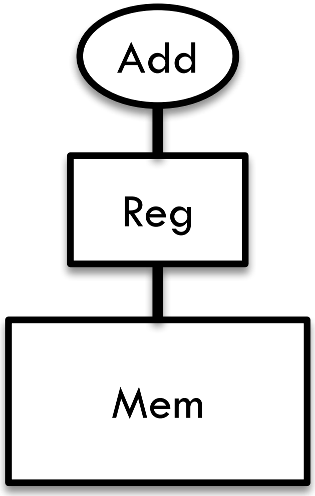
Memory Addressing

- Register
  - Add r4, r3
- Immediate
  - Add r4, #3
- Displacement
  - Add r4,100(r1)
- Register indirect
  - Add r4, (r1)



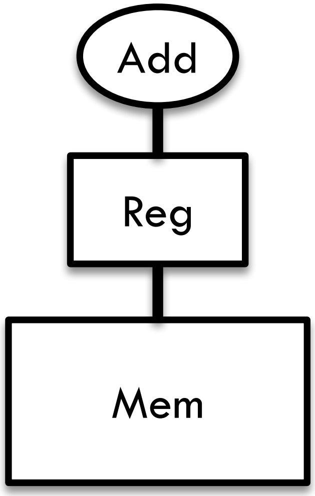
Memory Addressing

- Register
  - Add r4, r3 Reg[4]=Reg[4]+Reg[3]
- Immediate
  - Add r4, #3 Reg[4]=Reg[4]+3
- Displacement
  - Add r4,100(r1) …+Mem[100+Reg[1]]
- Register indirect
  - Add r4, (r1) …+Mem[Reg[1]]



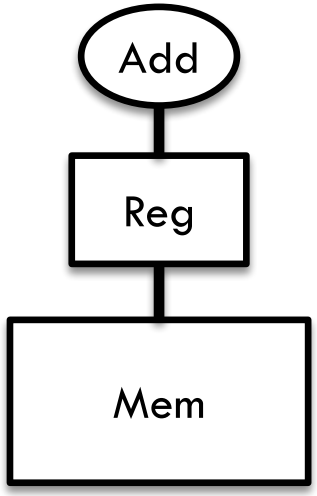
Memory Addressing

- Indexed
  - Add r3, (r1+r2)
- Direct
  - Add r1, (1001)
- Memory indirect
  - Add r1,@(r3)
- Auto-increment
  - Add r1, (r2)+



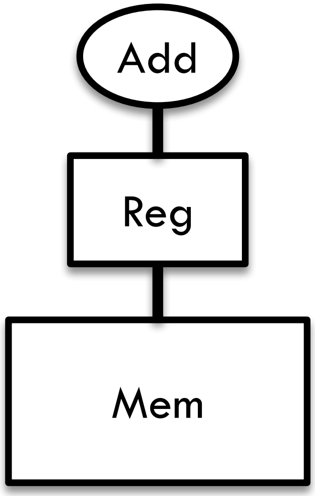
Memory Addressing

- Indexed
  - Add r3, (r1+r2)…+Mem[Reg[1]+Reg[2]]
- Direct
  - Add r1, (1001) …+Mem[1001]
- Memory indirect
  - Add r1,@(r3) …+Mem[Mem[Reg[3]]]
- Auto-increment
  - Add r1, (r2)+ …+Mem[Reg[2]]
  - Reg[2]=Reg[2]+d



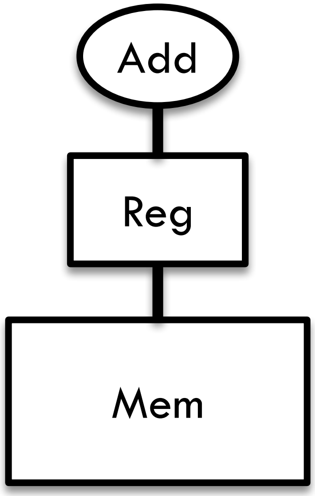
Memory Addressing

- Auto-decrement
  - Add r1, -(r2)
- Scaled
  - Add r1, 100(r2)[r3]



Memory Addressing

- Auto-decrement
  - Add r1, -(r2) Reg[2]=Reg[2]-d
  - …+Mem[Reg[2]]
- Scaled
  - Add r1, 100(r2)[r3]
  - …+Mem[100+Reg[2]+Reg[3] x d]



Example Problem

- Find the effective memory address
  - Add r2, 200(r1)
  - Add r2, (r1)
  - Add r2, @(r1)

Registers

r1	100
r2	200

Memory

…	…
100	400
200	500
300	600
400	700
500	800

Example Problem

- Find the effective memory address
  - Add r2, 200(r1)
    - r2 = r2 + Mem[300]
  - Add r2, (r1)
    - r2 = r2 + Mem[100]
  - Add r2, @(r1)
    - r2 = r2 + Mem[400]

Registers

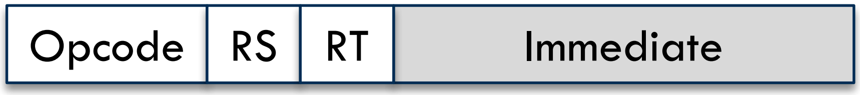
r1	100
r2	200

Memory

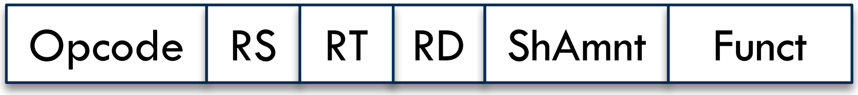
…	…
100	400
200	500
300	600
400	700
500	800

Instruction Format

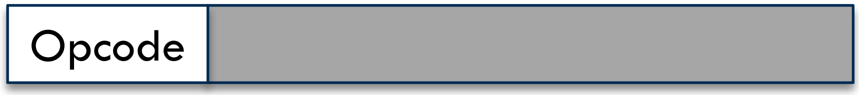
- A guideline for generating/interpreting instructions
- Example: MIPS
  - Fixed size 32-bit instructions
  - Three opcode types
    - I-type: load, store, conditional branch



    - R-type: ALU operations



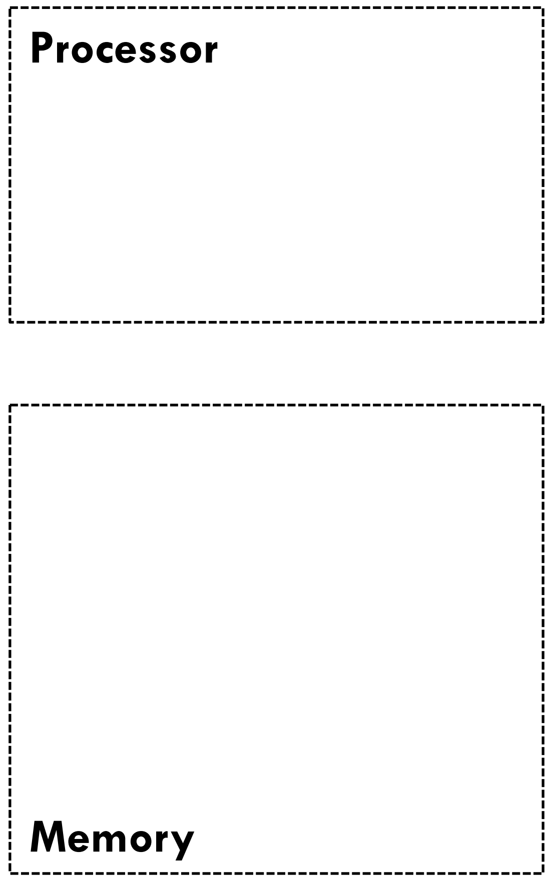
    - J-type: jump



Pipelining

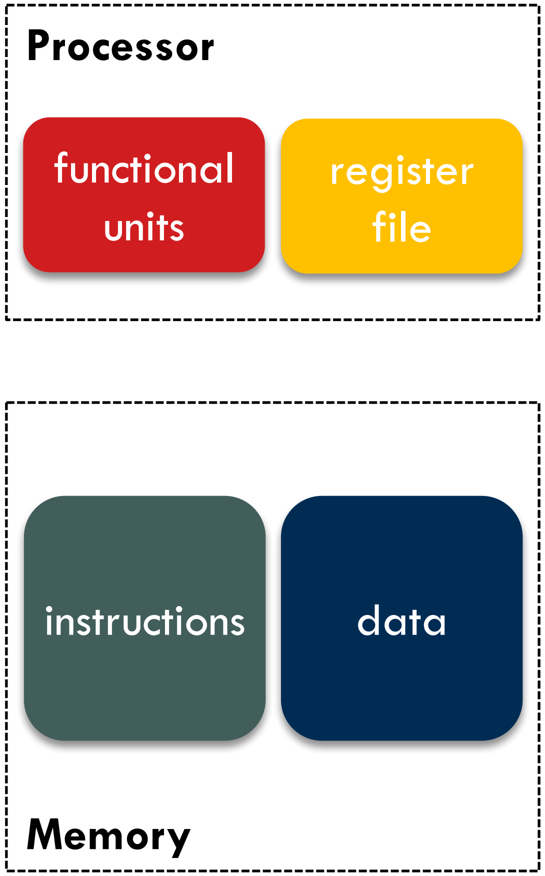
Processing Instructions

- Every RISC instruction may require multiple processing steps



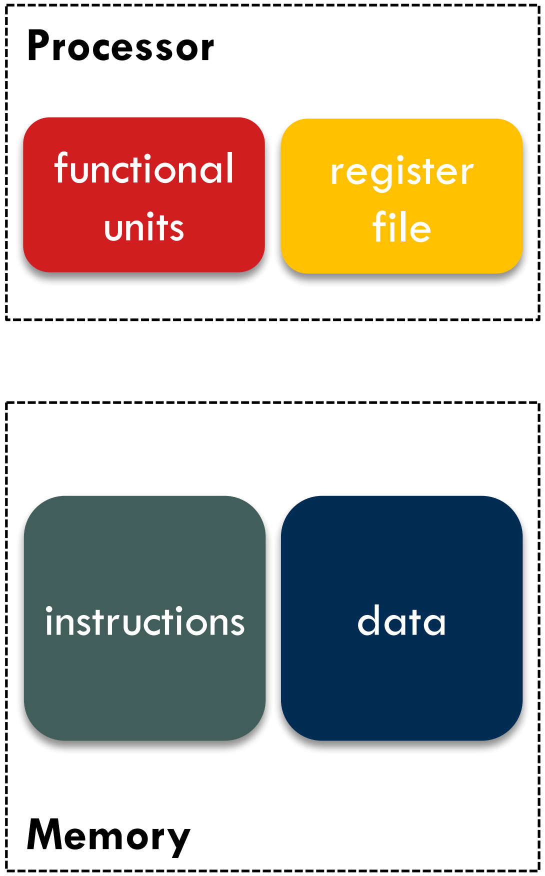
Processing Instructions

- Every RISC instruction may require multiple processing steps



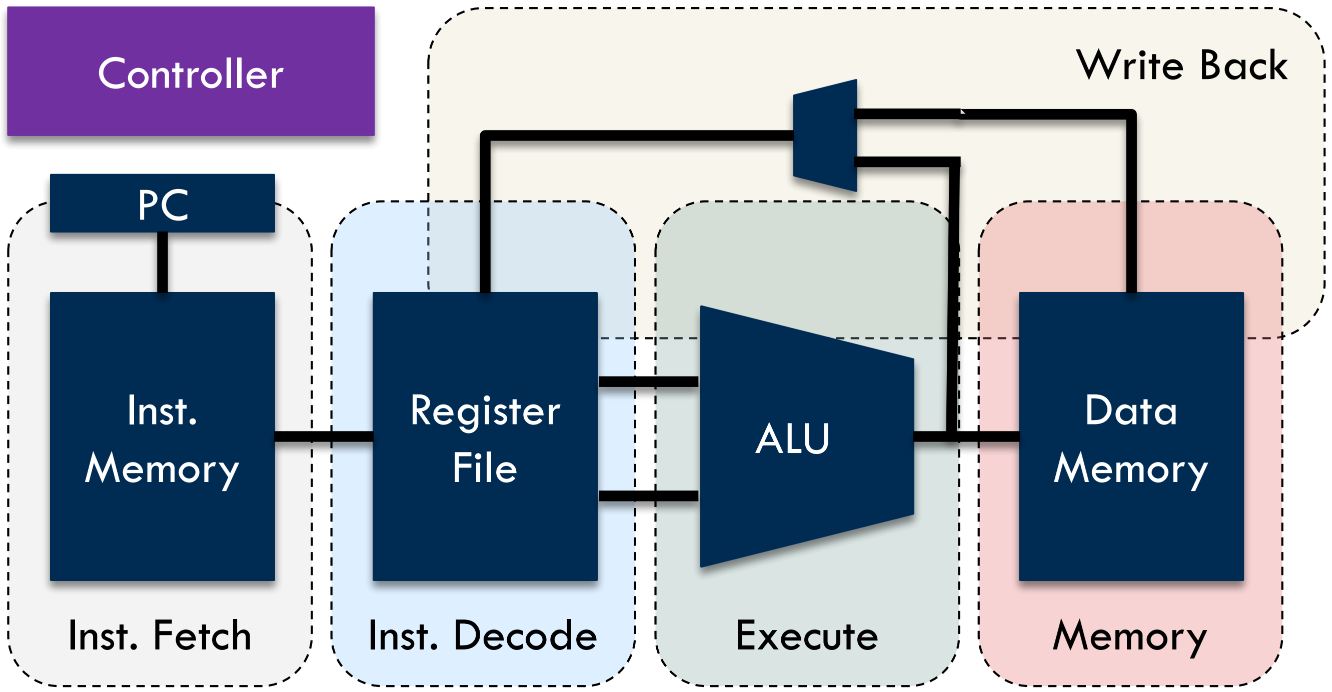
Processing Instructions

- Every RISC instruction may require multiple processing steps
  - Instruction Fetch (IF)
  - Instruction Decode (ID)
  - Register Read (RR)
    - All instructions?
  - Execute Instructions (EXE)
  - Memory Access (MEM)
    - All instructions?
  - Register Write Back (WB)



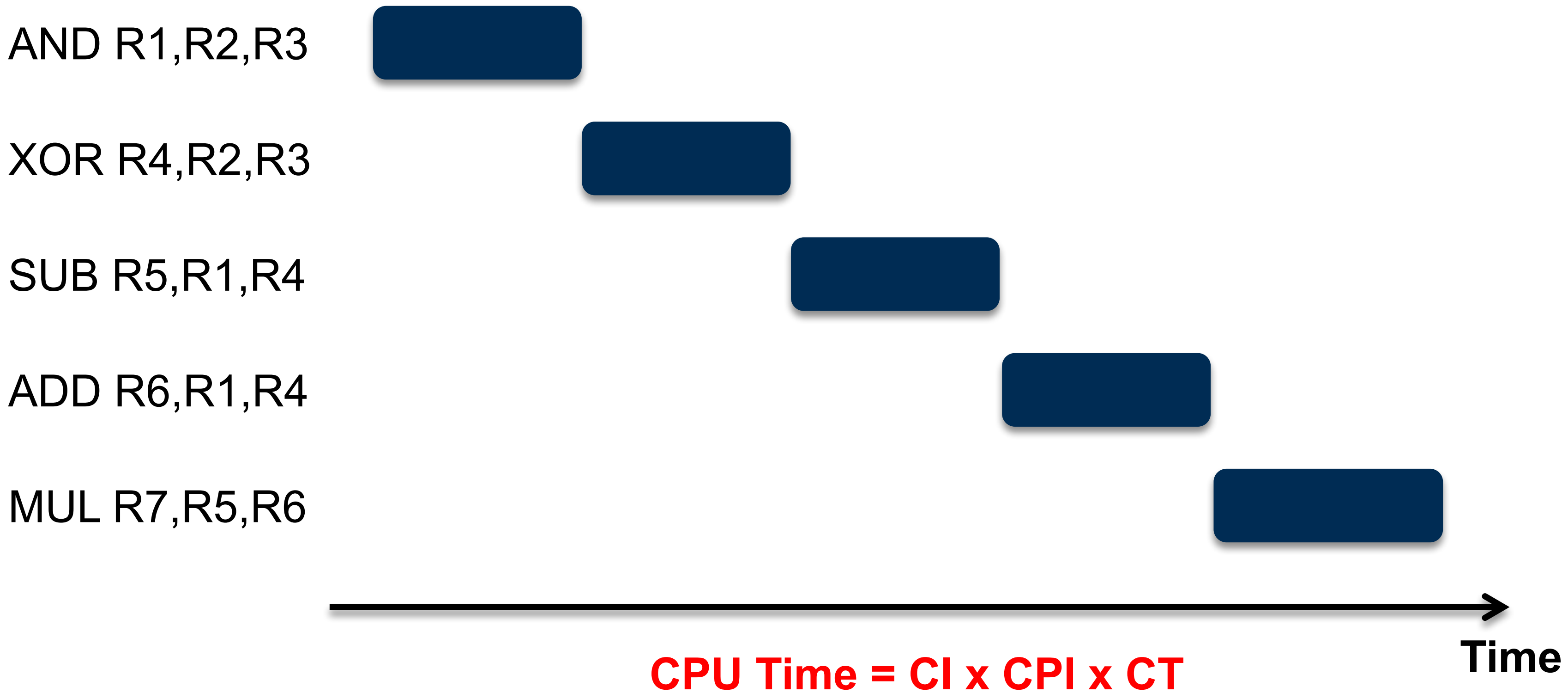
Single-cycle RISC Architecture

- Example: simple MIPS architecture
  - Critical path includes all of the processing steps



Single-cycle RISC Architecture

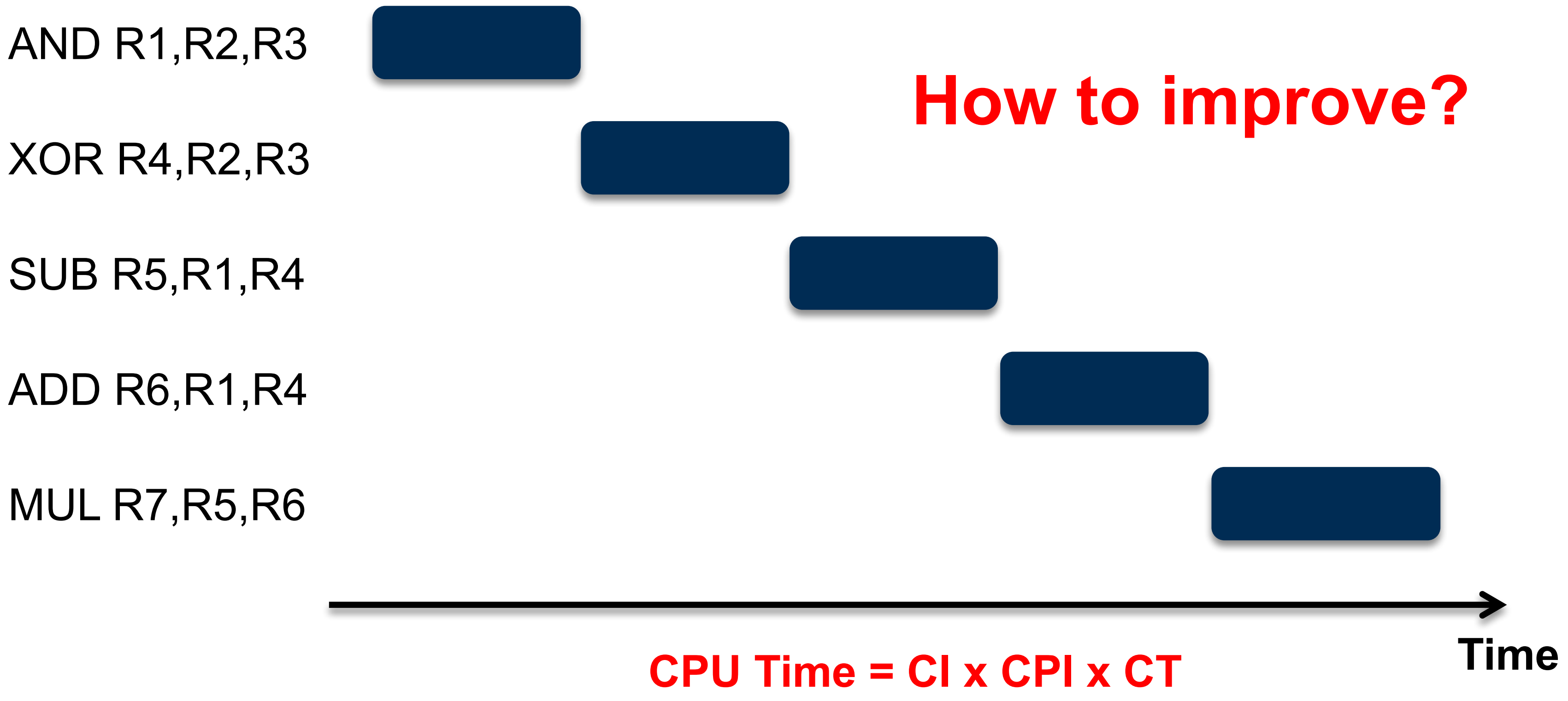
- Example program
  - CT=6ns; CPU Time = ?



CPU Time = CI x CPI x CT

Single-cycle RISC Architecture

- Example program
  - CT=6ns; CPU Time = 5 x 1 x 6ns = 30ns


AND R1,R2,R3
XOR R4,R2,R3
SUB R5,R1,R4
ADD R6,R1,R4
MUL R7,R5,R6
How to improve?
Time
CPU Time = CI x CPI x CT