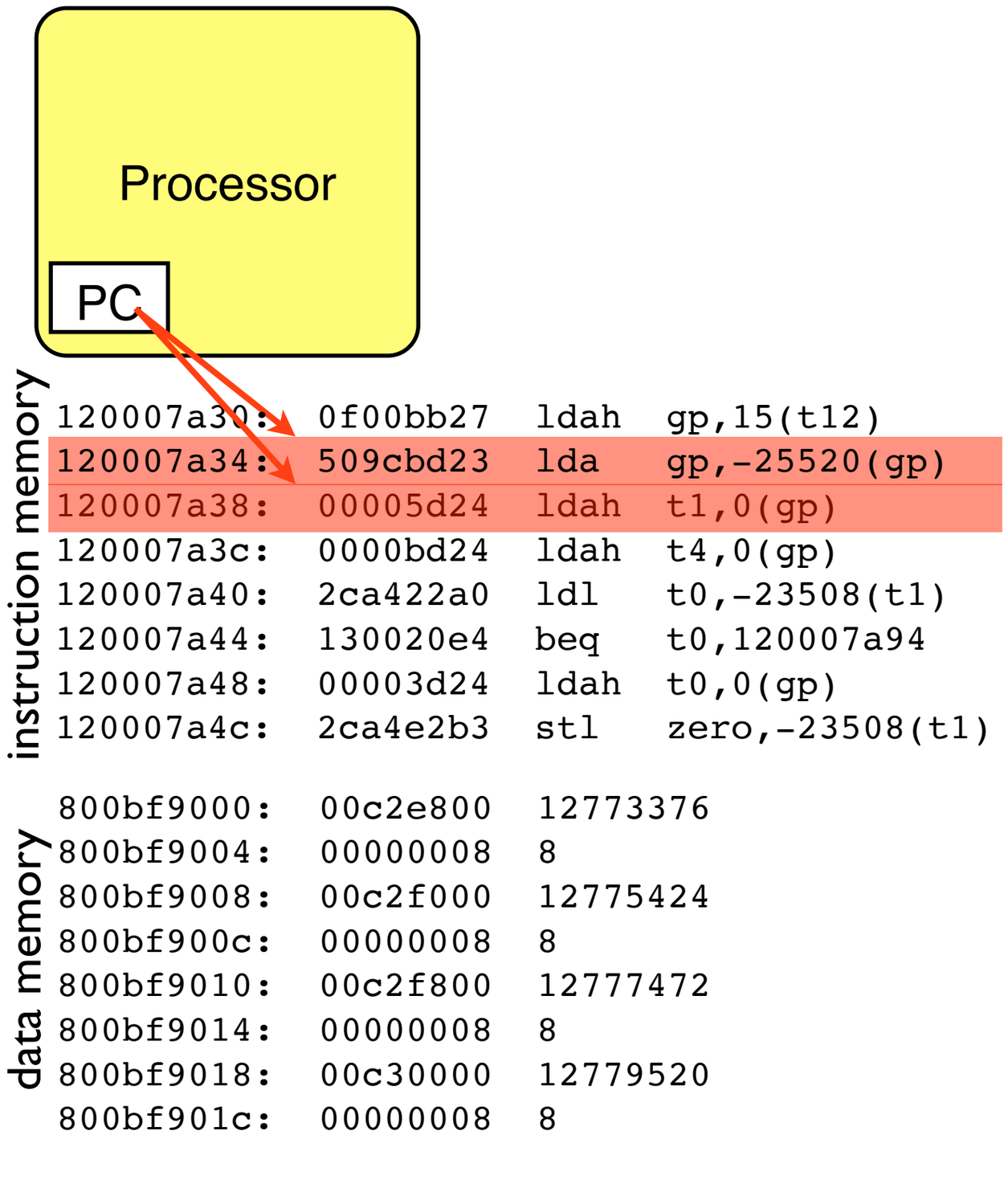
Processor Design – Single Cycle Processor

Hung-Wei Tseng

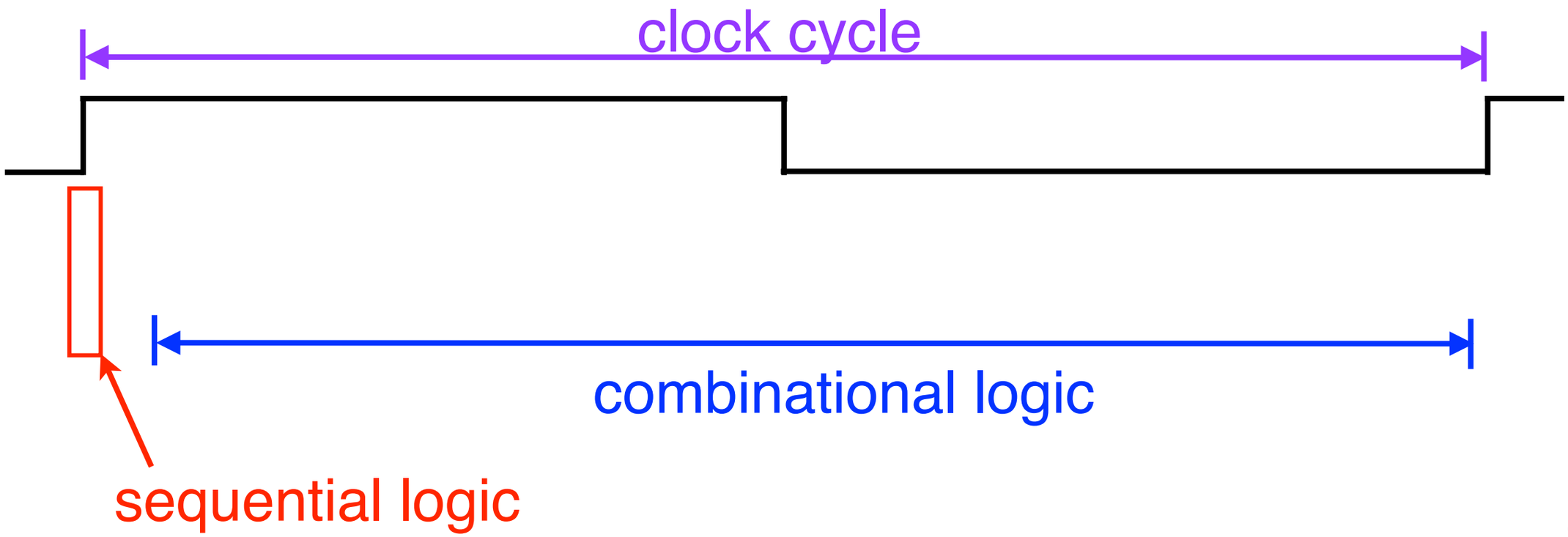
Recap: the stored-program computer

- Store instructions in memory
- The program counter (PC) controls the execution



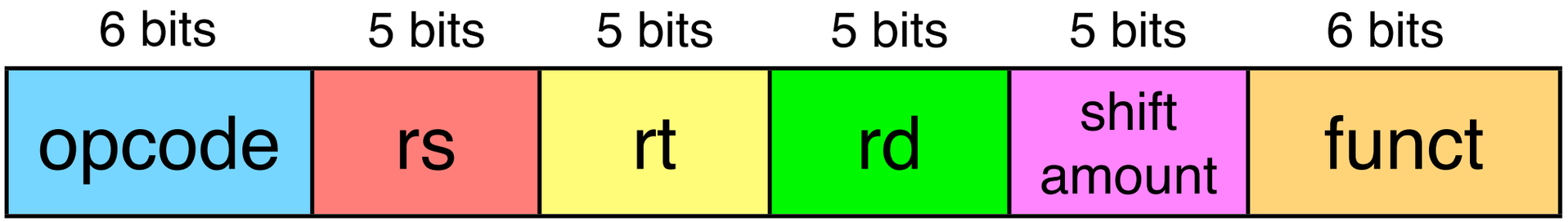
Recap: Clock

- A hardware signal defines when data is valid and stable
  - Think about the clock in real life!
- We use edge-triggered clocking
  - Values stored in the sequential logic is updated only on a clock edge

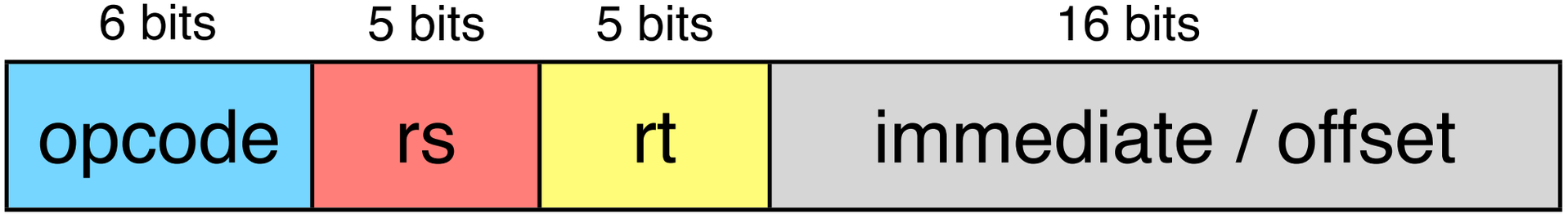


Recap: MIPS ISA

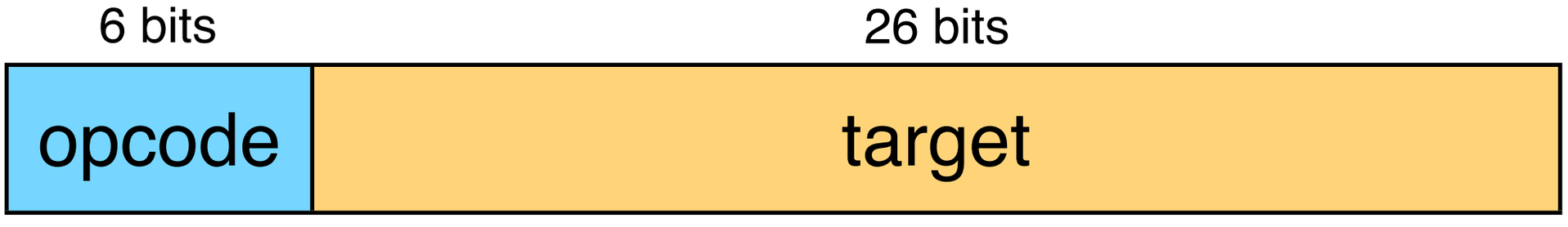
- R-type: add, sub, and etc...



- I-type: addi, lw, sw, beq, and etc...



- J-type: j, jal, and etc...



Outline

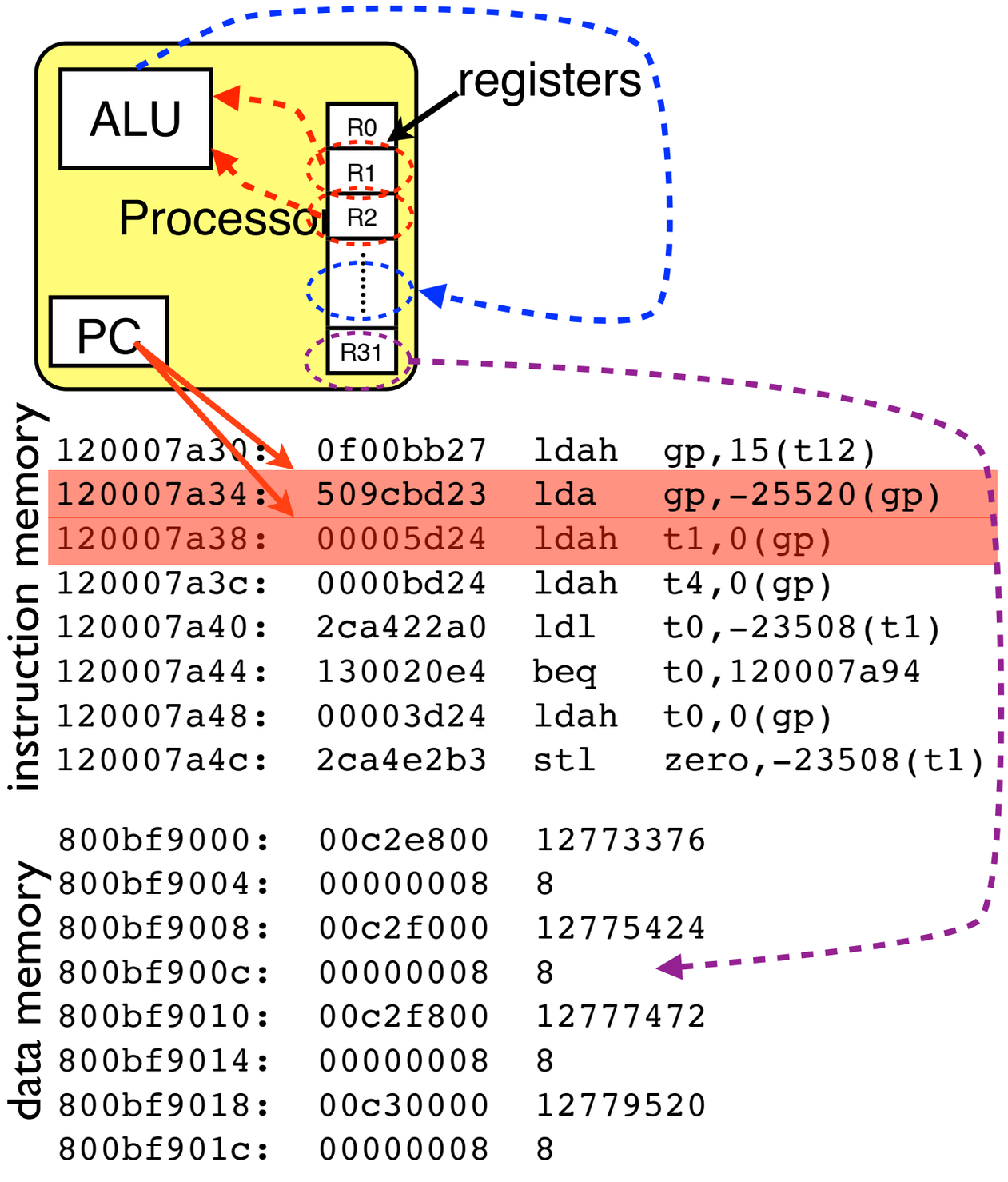
- Implementing a Single-cycle MIPS processor

Designing a simple MIPS processor

- Support MIPS ISA in hardware
  - Design the datapath: add and connect all the required elements in the right order
  - Design the control path: control each datapath element to function correctly.
- Starts from designing a single cycle processor
  - Each instruction takes exactly one cycle to execute

Basic steps of execution

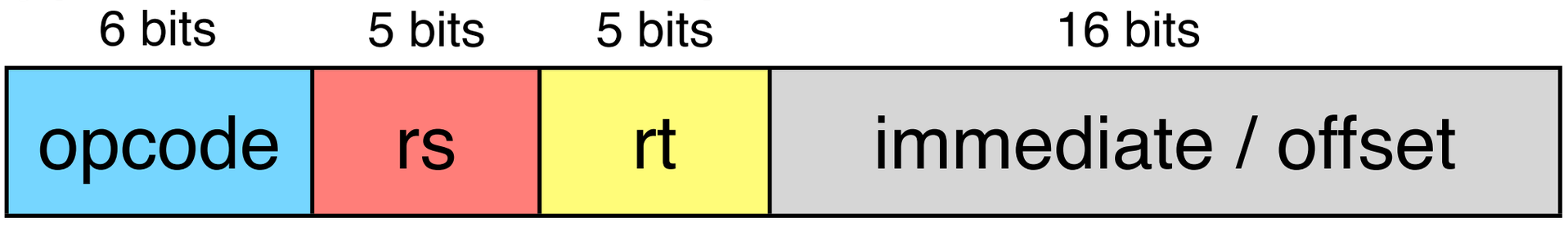
- Instruction fetch: where?
  instruction memory
- Decode:
  - What's the instruction?
  - Where are the operands?
    registers
- Execute **ALUs**
- Memory access
  - Where is my data?
    data memory
- Write back
  - Where to put the result
    registers
- Determine the next PC



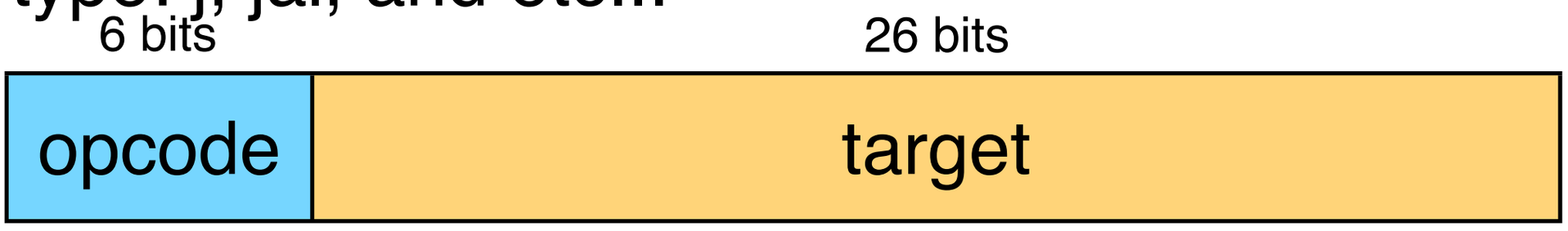
Recap: MIPS ISA

- R-type: add, sub, and etc...

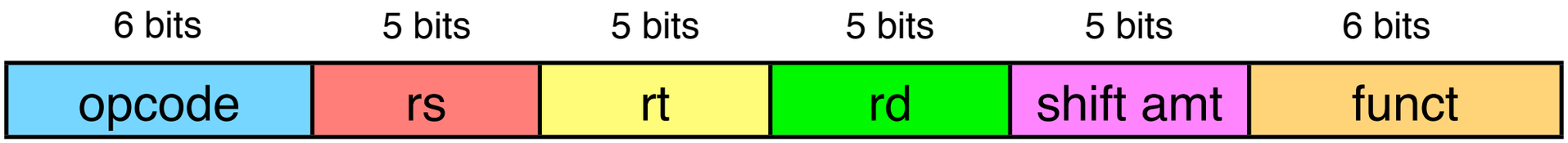
- I-type: addi, lw, sw, beq, and etc...



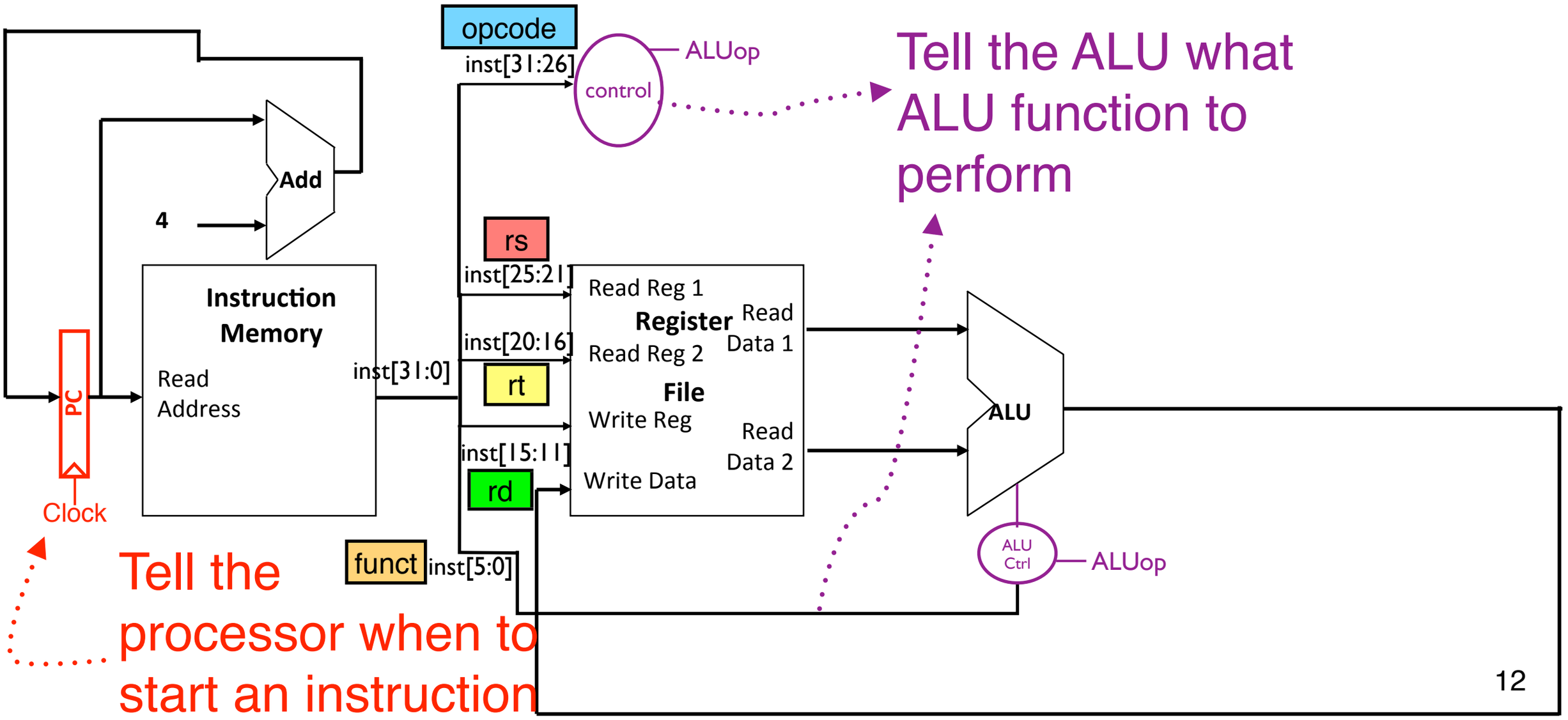
- J-type: j, jal, and etc...



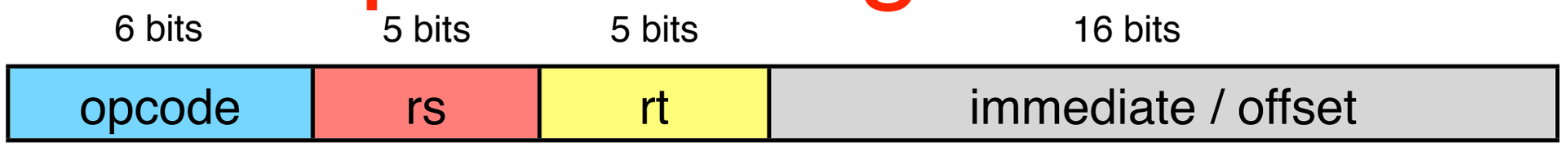
Implementing an R-type instruction



instruction = MEM[PC]
REG[rd] = REG[rs] op REG[rt]
PC = PC + 4



Implementing a load instruction



instruction = MEM[PC]

REG[rt] = MEM[signext(immediate) + REG[rs]]

PC = PC + 4

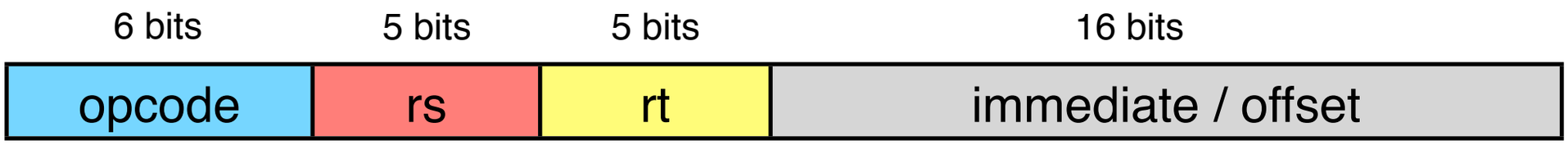
Set different control signals for different types of instructions

Set to 1 if it's a load

Set to 1 if it's a load

Set to 0 if it's a load

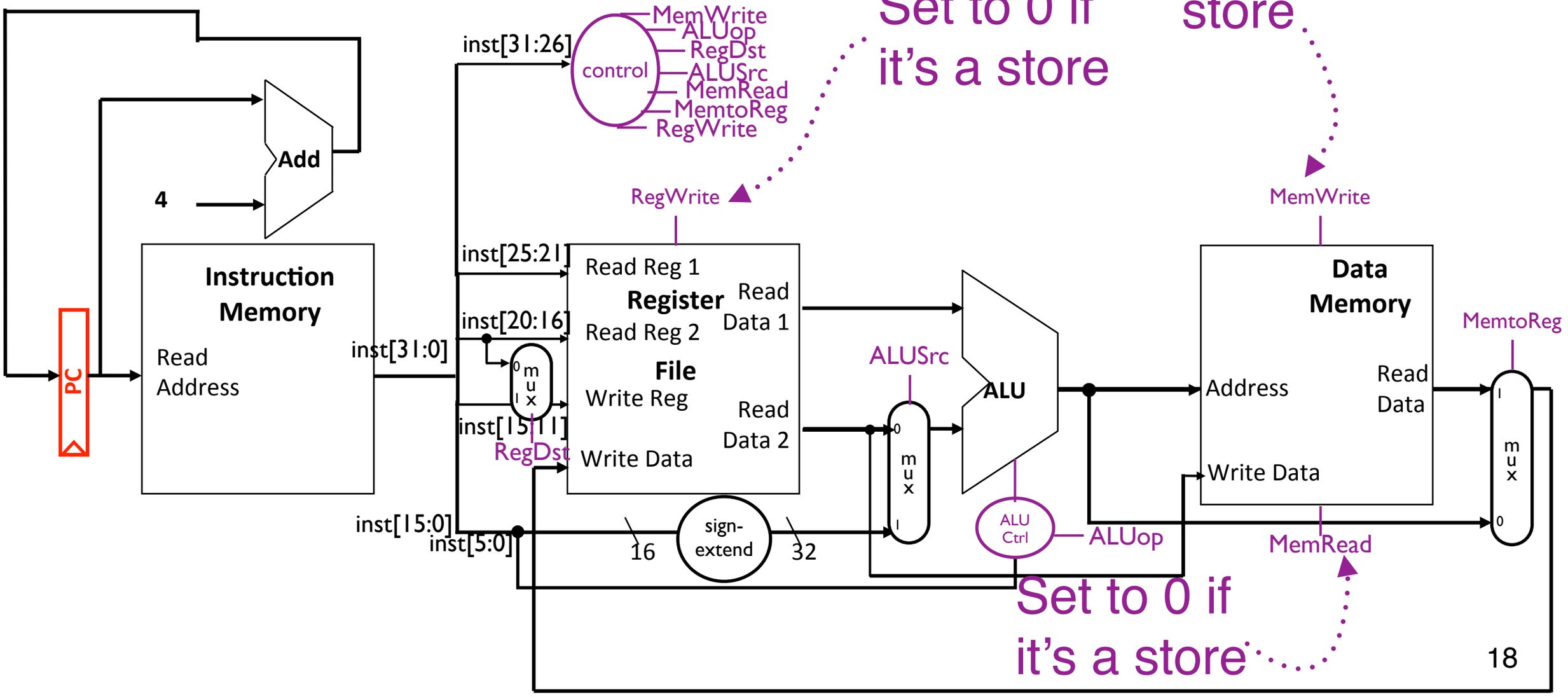
Implementing a store instruction



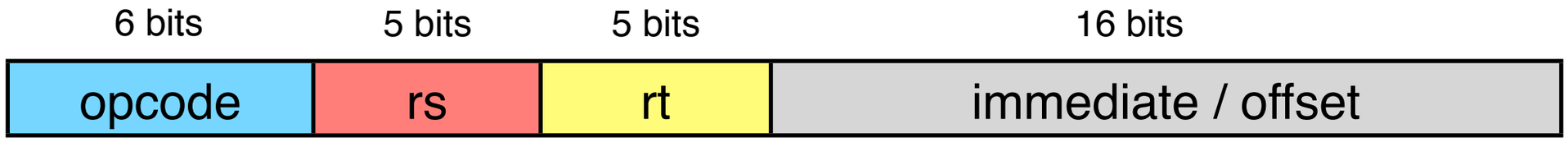
instruction = MEM[PC]

MEM[signext(immediate) + REG[rs]] = REG[rt]

PC = PC + 4



Implementing a branch instruction



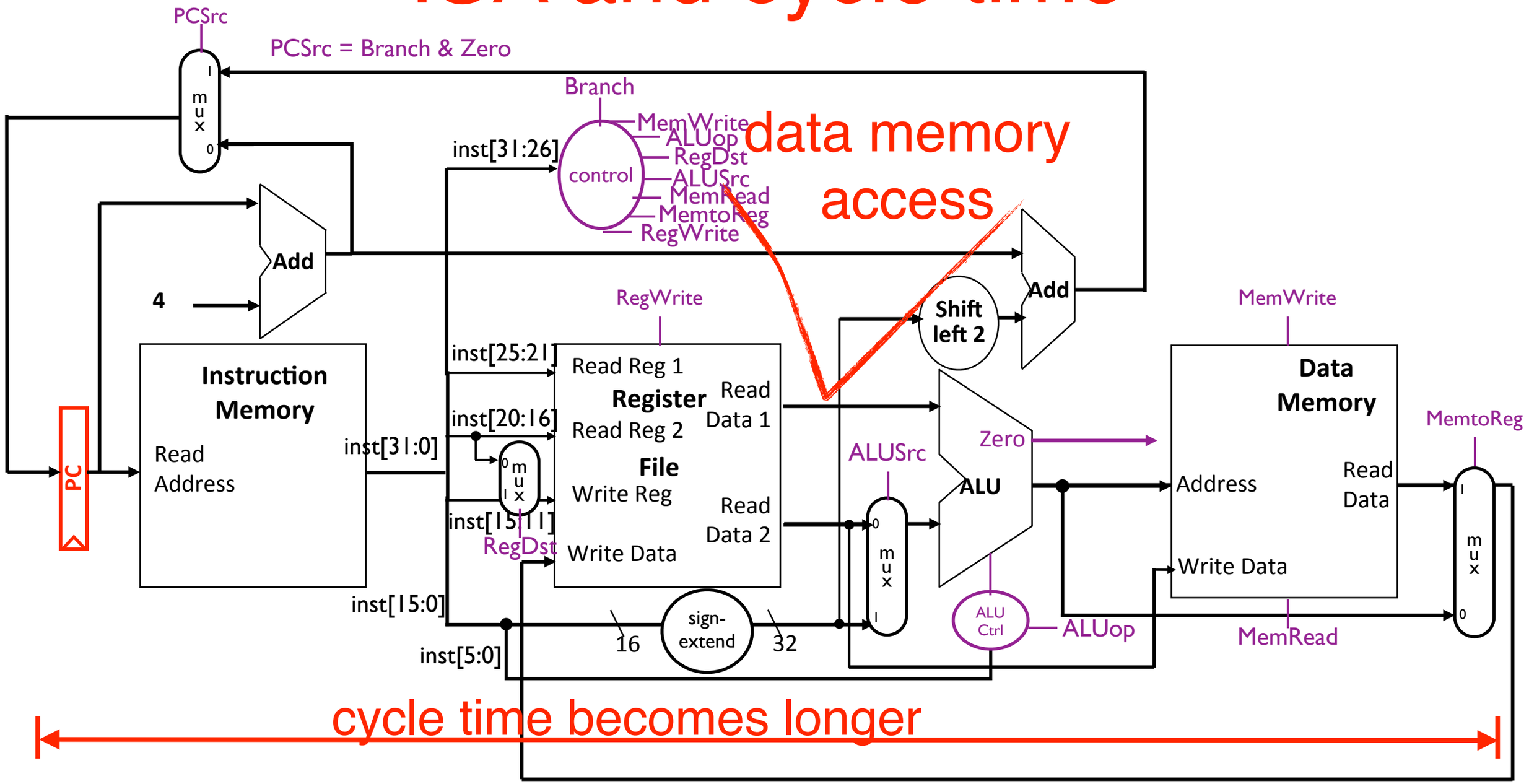
instruction = MEM[PC]

PC = (REG[rs] == REG[rt]) ? PC + 4 + SignExtImmediate *4 : PC + 4

PCSrc = Branch & Zero

Calculate the target address

ISA and cycle time



x86 supports add instruction with memory content as a source operand.
e.g. add %eax, (%r12) means %eax = %eax + memory[%r12]
If MIPS wants to support this instruction, what modification is necessary? What's the performance impact?

Q & A