CprE 381: Computer Organization and Assembly Level Programming

Processor Design

Henry Duwe
Electrical and Computer Engineering
Iowa State University

Administrative

- Exam 1 will be returned on Canvas by tomorrow morning
- HW4
  - Redo your worst exam problem
  - Make it perfect (limited partial credit)
    - Sometimes you (or your classmates) get points for parts not 100% correct
    - Ask questions in OH or on Canvas (I'm fine discussing specific solutions)

Administrative – Course Workload

- Qualitative Feedback:
  - Too much work
  - Liked "putting it together"
  - Liked and disliked seeing different ways to implement
  - Disliked amount of testing
- Going forward:
  - Reducing the number of instructions needed to support → reduce work on lower-value content
  - Reduce # of test programs individual teams are responsible for → common test pool created by class
  - Testing framework and processor interface/memory instantiation provided

Administrative – Course Workload

- Lab 2
  - Median reporting student: 8.2hrs outside
  - Median grade (all students): 100%
- Lab 3
  - Median reporting student: 9.8hrs outside
  - Most students had 0hrs in lab two weeks ago
- 4 week average from reporting students
  - ~5hrs on labs outside of class
  - Will keep updated throughout semester

Processor Design Approach

- Break operation down to steps even logic gates can understand
- Generally decompose into two kinds of operations:
  - Things that deal with the real data (datapath)
  - Things that control the stuff operating on the real data (control)
- Find a decomposition that is simple and efficient
- We will start simple
  - Later (in lecture and lab), we'll add features to improve performance

The Processor: Datapath & Control

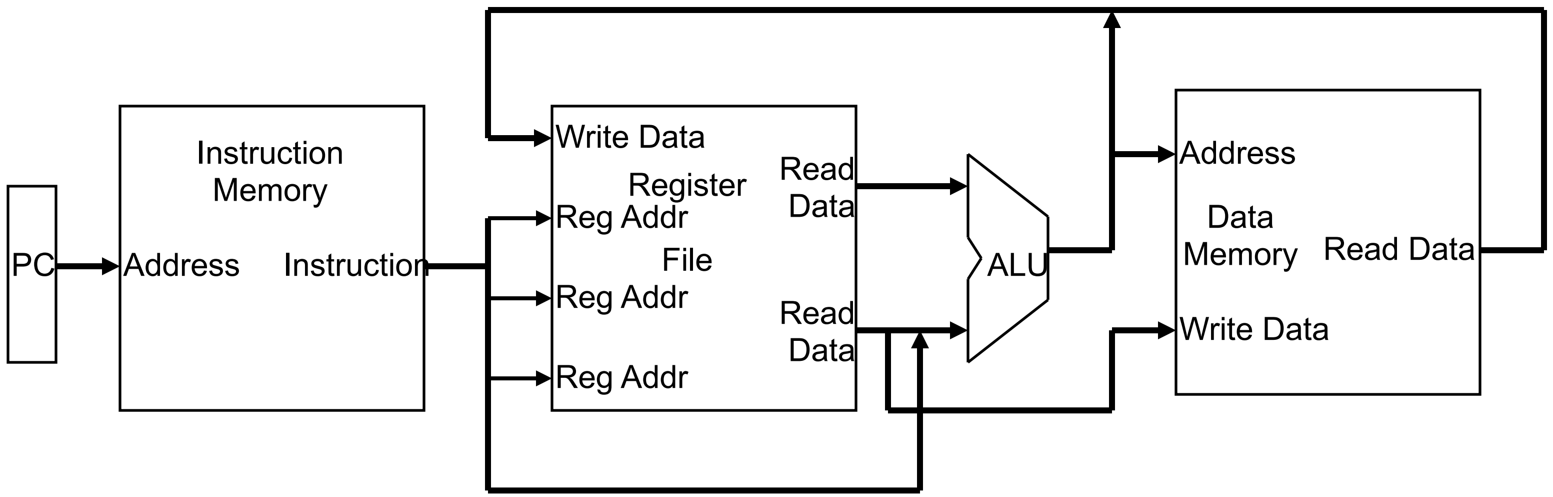
- Textbook MIPS implementation: simplified to contain only:
  - Memory-reference instructions: `lw, sw`
  - Arithmetic-logical instructions: `add, addu, sub, subu, and, or, xor, nor, slt, sltu`
  - Arithmetic-logical immediate instructions: `addi, addiu, andi, ori, xori, slti, sltiu`
  - Control flow instructions: `beq, j`

- You lab version will require more instructions and thus modifications to this base design!
  - More info in Lec07,1

- Generic implementation:
  - Use the program counter (**PC**) to supply the instruction address and **fetch** the instruction from memory (and update the PC)
  - **Decode** the instruction (and read registers)
  - **Execute** the instruction (already built in lab)

Abstract Implementation View

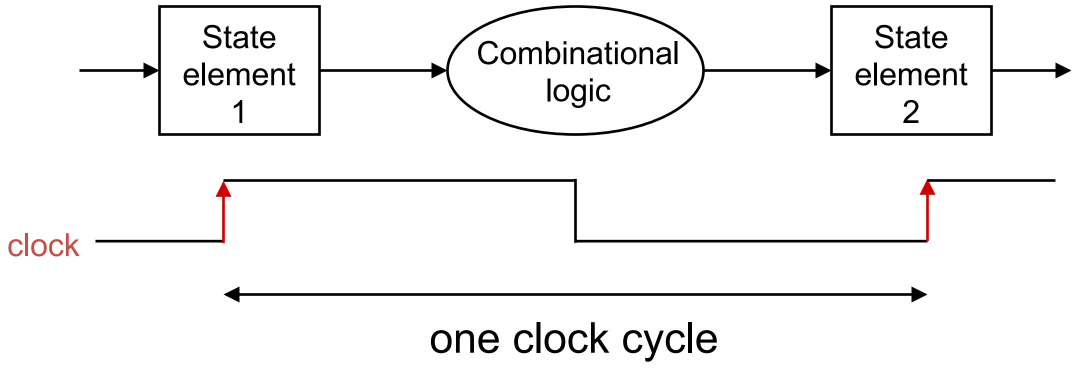
- Two types of blocks:
  - Elements that operate on data values (combinational)
  - Elements that contain state (sequential)



- **Single cycle** operation
- Split memory (Harvard) model - one memory for instructions and one for data

Edge-triggered Implementation

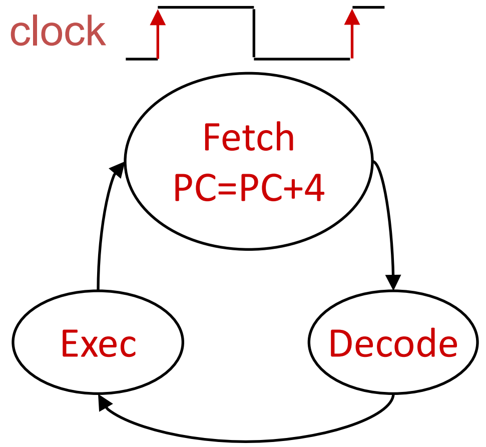
- An edge-triggered methodology, typical execution
  - Read contents of some state elements (combinational activity, so no clock control signal needed)
  - Send values through some combinational logic
  - Write results to one or more state elements on clock edge

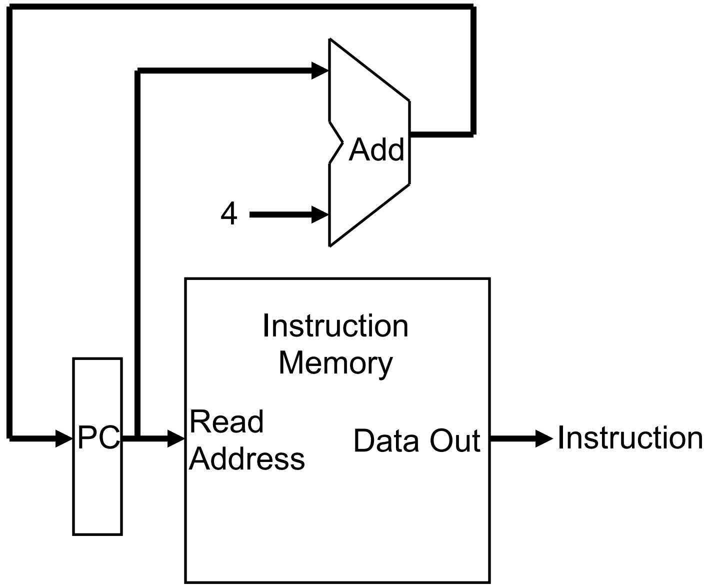


- Assumes state elements are written on every clock cycle; if not, need explicit write control signal
  - Write occurs only when both the write control is asserted and the clock edge occurs

Fetching Instructions

- Fetching instructions involves
  - reading the instruction from the Instruction Memory
  - updating the PC value to be the address of the next (sequential) instruction

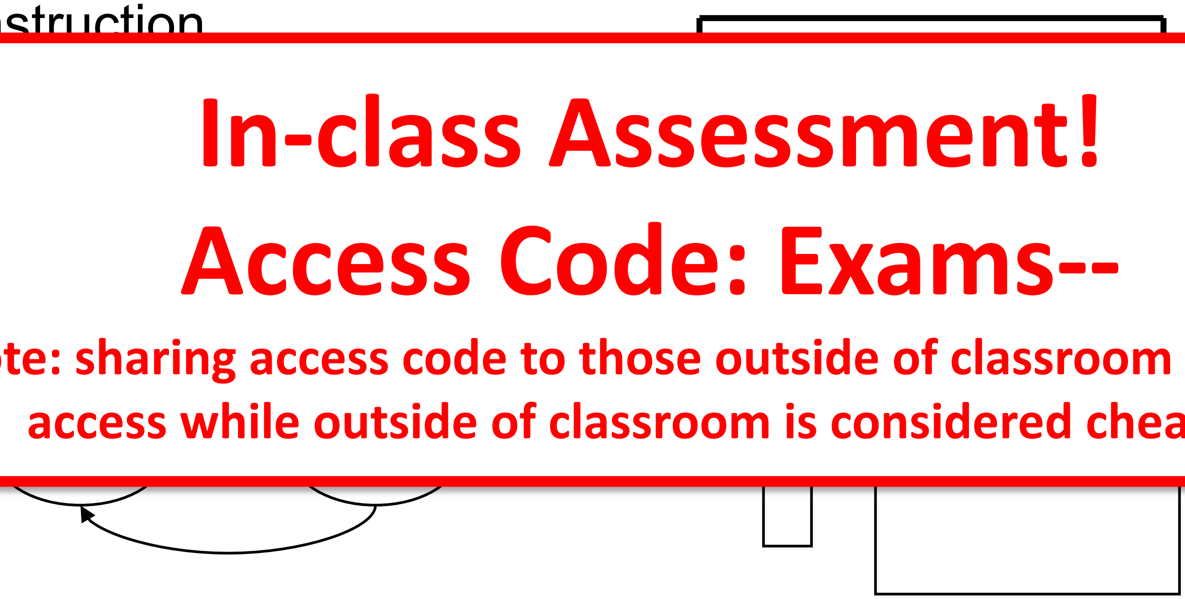




  - PC is updated every clock cycle, so it does not need an explicit write control signal – just a clock signal
  - Reading from the Instruction Memory is a combinational activity

Fetching Instructions

- Fetching instructions involves
  - reading the instruction from the Instruction Memory
  - updating the PC value to be the address of the next (sequential) instruction



In-class Assessment!

Access Code: Exams--

Note: sharing access code to those outside of classroom or using access while outside of classroom is considered cheating

  - PC is updated every clock cycle, so it does not need an explicit write control signal – just a clock signal
  - Reading from the Instruction Memory is a combinational activity

PC Assessment

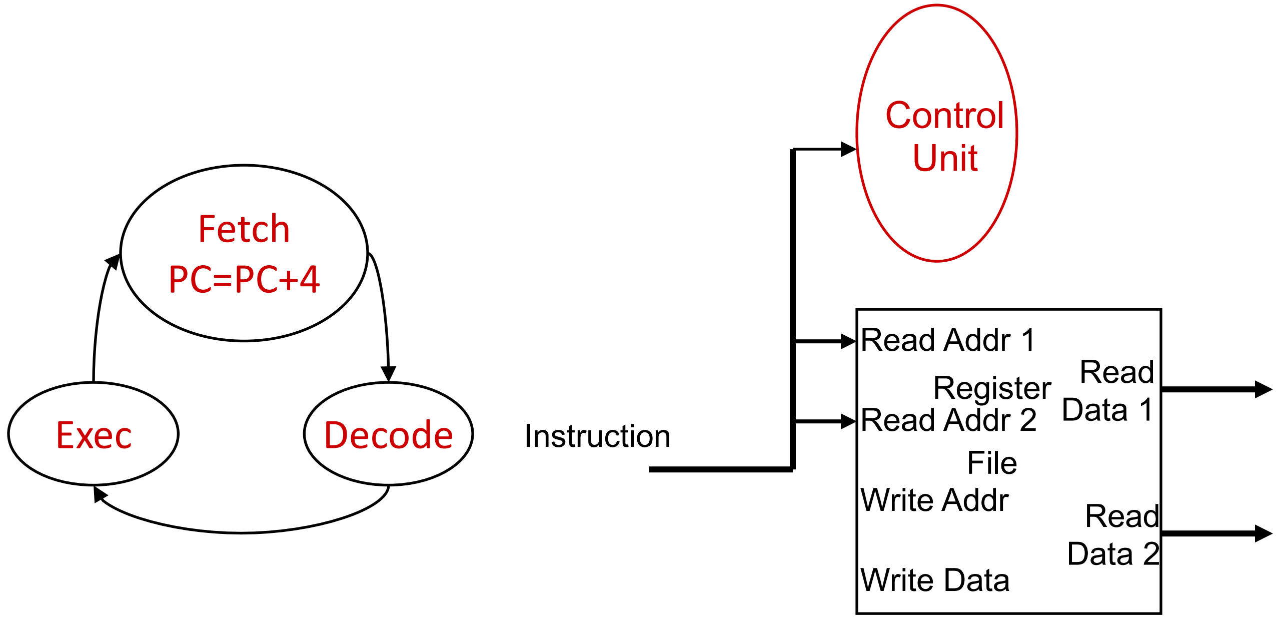
- Consider the following instruction sequence located at (0x00400000):

```
begin:
   addi $t5, $zero, 1195
   j cond
loop:
   sra $t5, $t5, 1
   andi $t6, $t5, 0x003C
   add $t9, $a0, $t6
   sw $t5, 4($t9)
cond:
   bne $t5, $zero, loop
   jr $s7
```

- What PC value fetches the sra $t5, $t5, 1 instruction (use 0xXXXXXXXX format)?

Decoding Instructions

- Decoding instructions involves
  - Sending the fetched instruction's opcode and function field bits to the control unit



  - Reading two values from the Register File
    - Register File addresses are contained in the instruction

Reading Registers "Just in Case"

- Note that both RegFile read ports are active for all instructions during the Decode cycle using the `rs` and `rt` instruction field addresses
  - Since we haven't decoded the instruction yet, we don't know what the instruction is!
  - *Just in case* the instruction uses values from the RegFile do "work ahead" by reading the two source operands

Which instructions *do* make use of the RegFile values?

Reading Registers "Just in Case"

- Note that both RegFile read ports are active for all instructions during the Decode cycle using the `rs` and `rt` instruction field addresses
  - Since we haven't decoded the instruction yet, we don't know what the instruction is!
  - *Just in case* the instruction uses values from the RegFile do "work ahead" by reading the two source operands

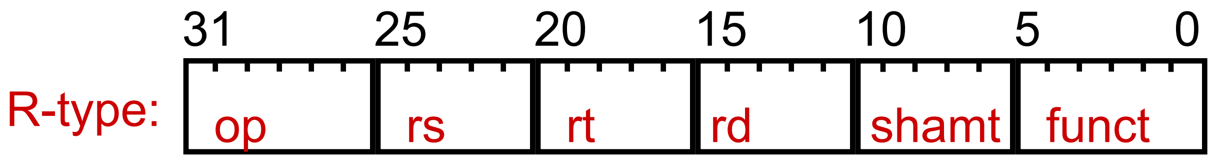
Which instructions *do* make use of the RegFile values?

- Also, all instructions (except `j`) use the ALU after reading the registers

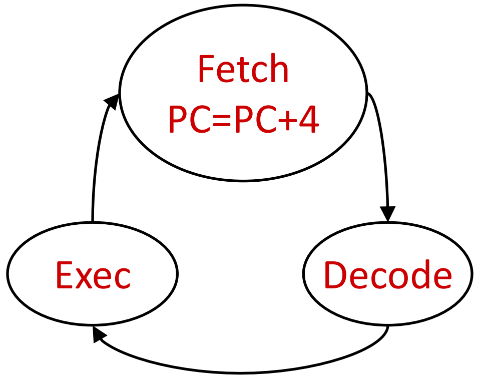
For what purpose? arithmetic? memory? control flow?

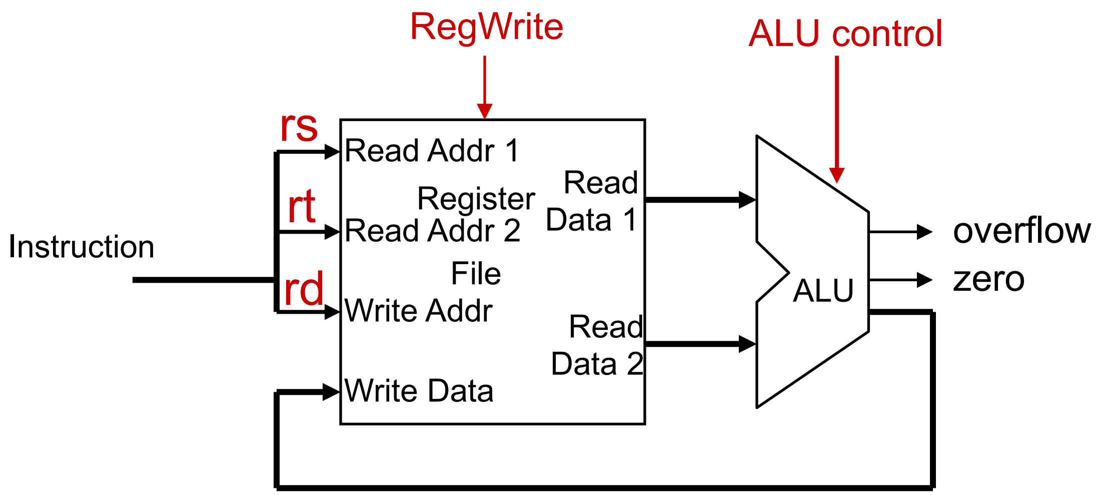
Review: Executing R Format Operations

- R format operations (`add, sub, slt, and, or`)



  - Perform operation (op and funct) on values in rs and rt
  - Store the result back into the Register File (into location rd)





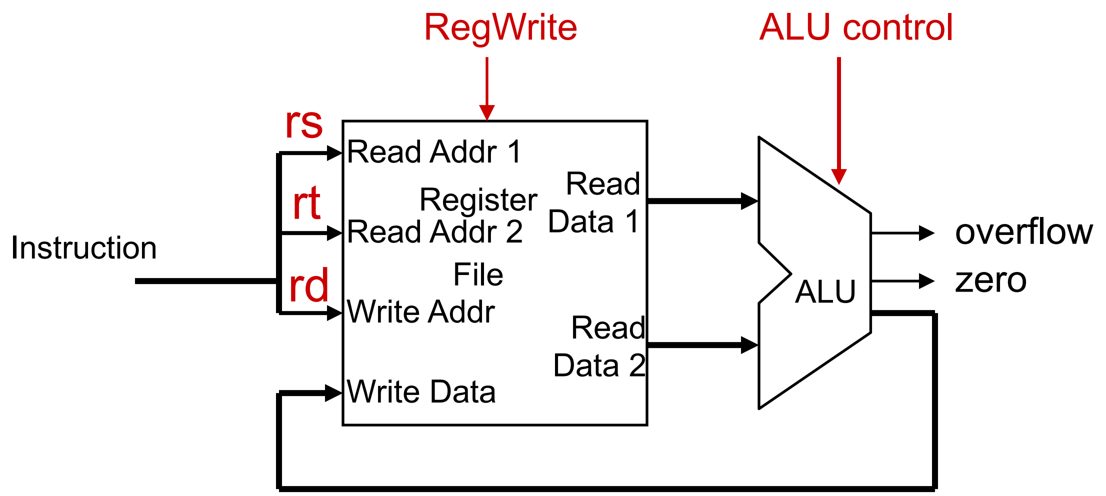
  - Note that Register File is not written every cycle (e.g. `sw` or `jr`), so we need an explicit write control signal for the Register File

Consider the `slt` Instruction

- Remember the R format instruction **`slt`**

```
slt $t0, $s0, $s1   # if $s0 < $s1
                    #    then  $t0 = 1
                    # else  $t0 = 0
```

  - Where does the 1 (or 0) come from to store into $t0 in the Register File at the end of the execute cycle?



Acknowledgments

- These slides contain material developed and copyright by:
  - Joe Zambreno (Iowa State)
  - David Patterson (UC Berkeley)
  - Mary Jane Irwin (Penn State)
  - Christos Kozyrakis (Stanford)
  - Onur Mutlu (Carnegie Mellon)
  - Krste Asanović (UC Berkeley)