Processors & Pipelines

A general summary of CS:APP 4.1

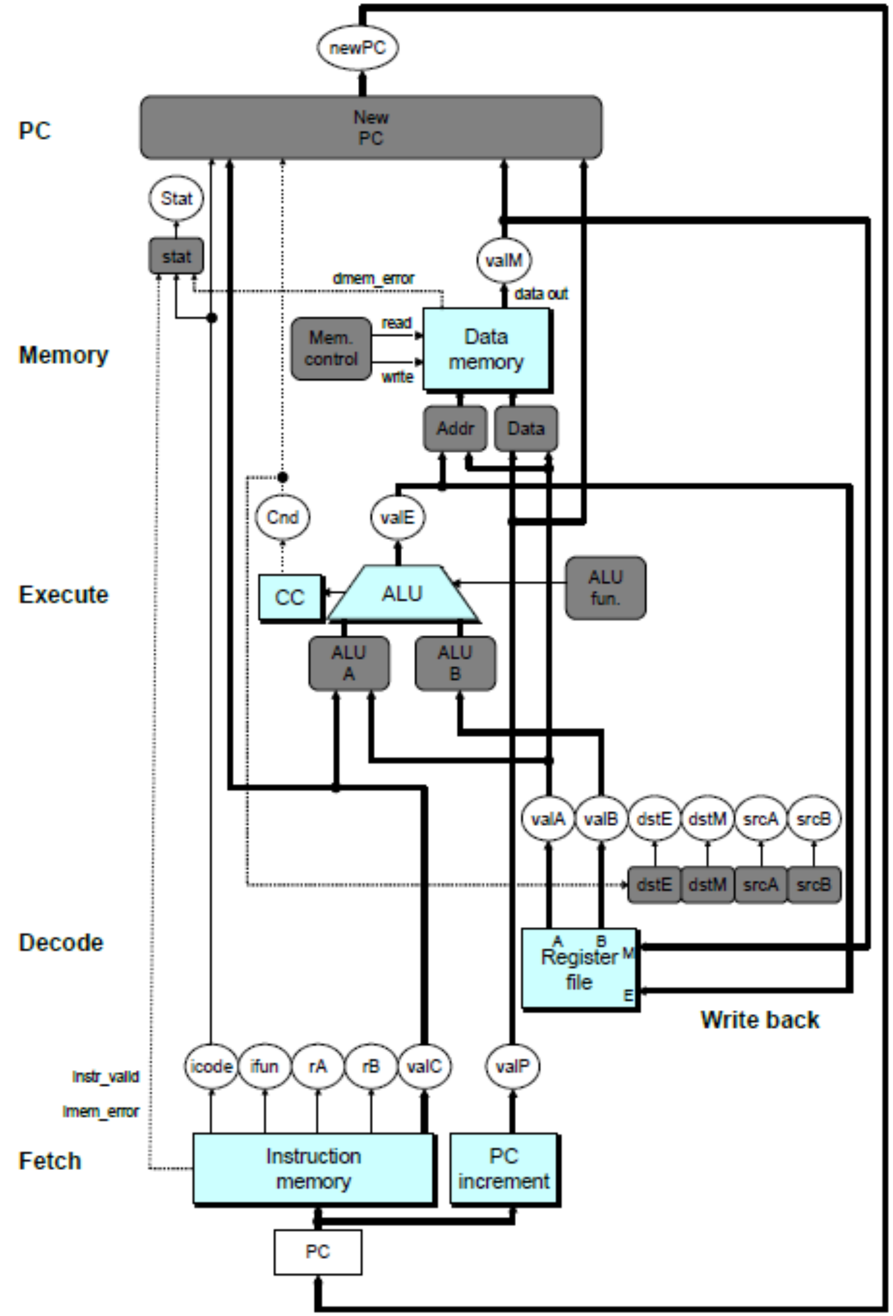
These slides adapted from materials provided by the textbook authors.

Processors & Pipelines

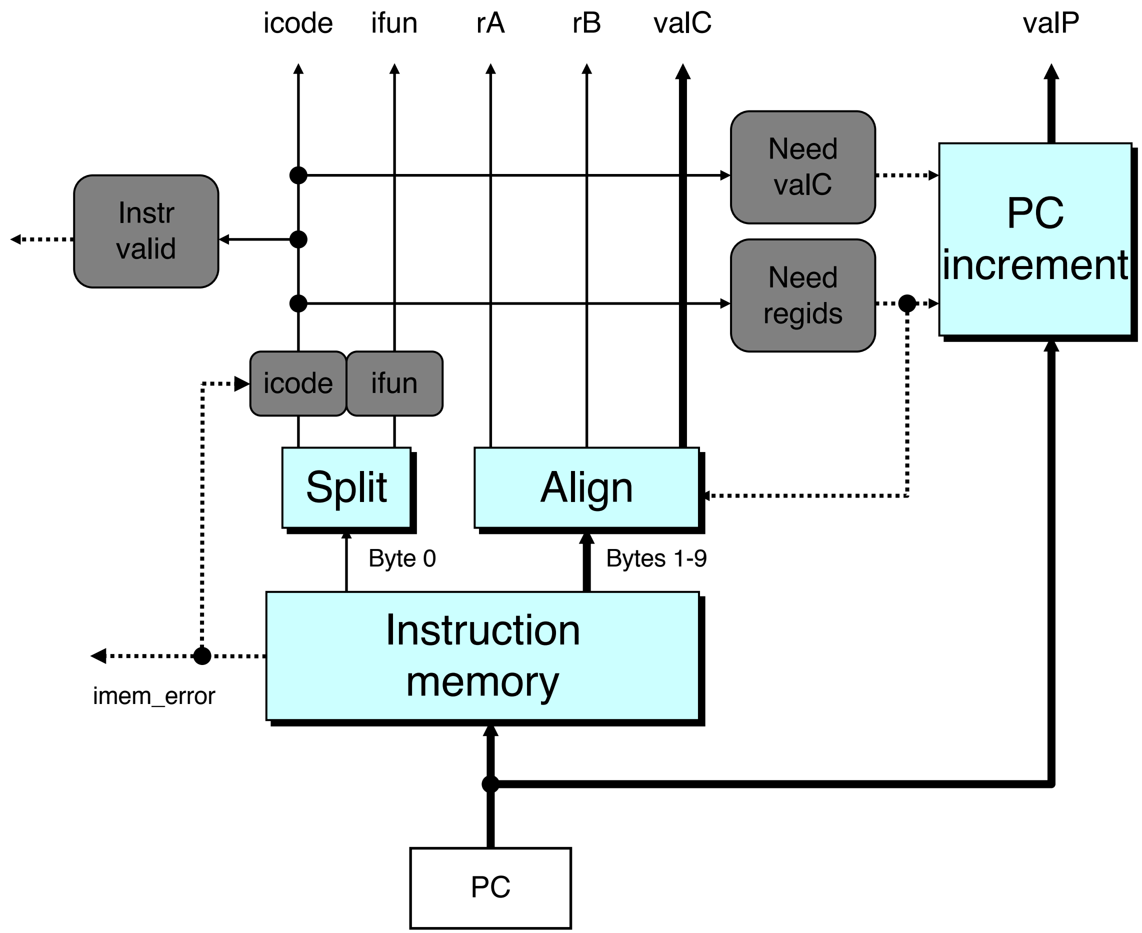
- **Process Architecture, in Brief**
- Pipelining (Instruction-Level Parallelism), Continued
- Dealing with Conditionals

SEQ Hardware

- **Key**
  - Blue boxes: predesigned hardware blocks
    - E.g., memories, ALU
  - Gray boxes: control logic (HCL)
  - White ovals: labels for signals
  - Thick lines: 64-bit word values
  - Thin lines: 4-8 bit values
  - Dotted lines: 1-bit values



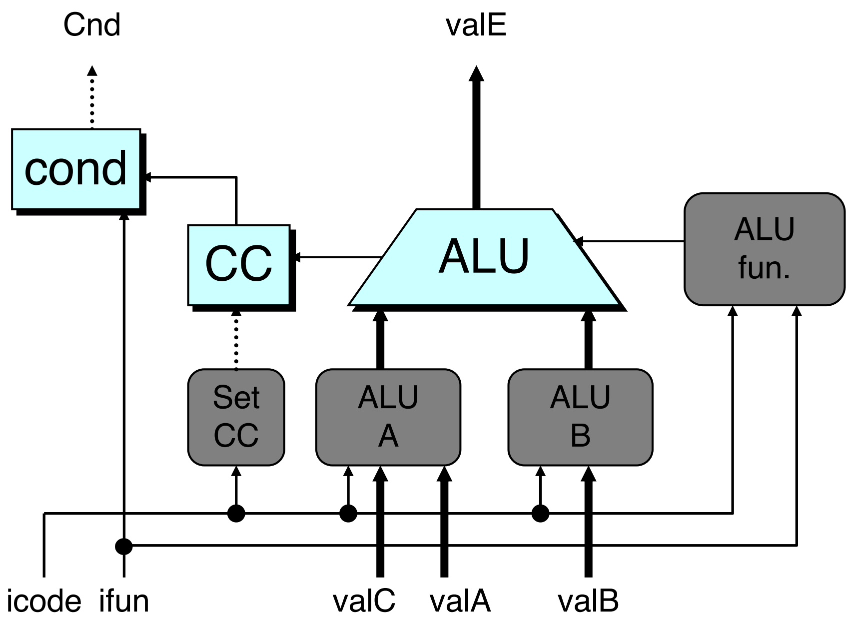
Fetch Logic



- **Control Logic**
  - Instr. Valid: Is this instruction valid?
  - icode, ifun: Generate no-op if invalid address
  - Need regids: Does this instruction have a register byte?
  - Need valC: Does this instruction have a constant word?

Execute Logic

- **Units**
  - ALU
    - Implements 4 required functions
    - Generates condition code values
  - CC
    - Register with 3 condition code bits
  - cond
    - Computes conditional jump/move flag
- **Control Logic**
  - Set CC: Should condition code register be loaded?
  - ALU A: Input A to ALU
  - ALU B: Input B to ALU
  - ALU fun: What function should ALU compute?



ALU Operation

	OPl rA, rB	
Execute	valE ← valB OP valA	Perform ALU operation

	cmovXX rA, rB	
Execute	valE ← 0 + valA	Pass valA through ALU

	`rmmovl` rA, D(rB)	
Execute	valE ← valB + valC	Compute effective address

	`popq` rA	
Execute	valE ← valB + 8	Increment stack pointer

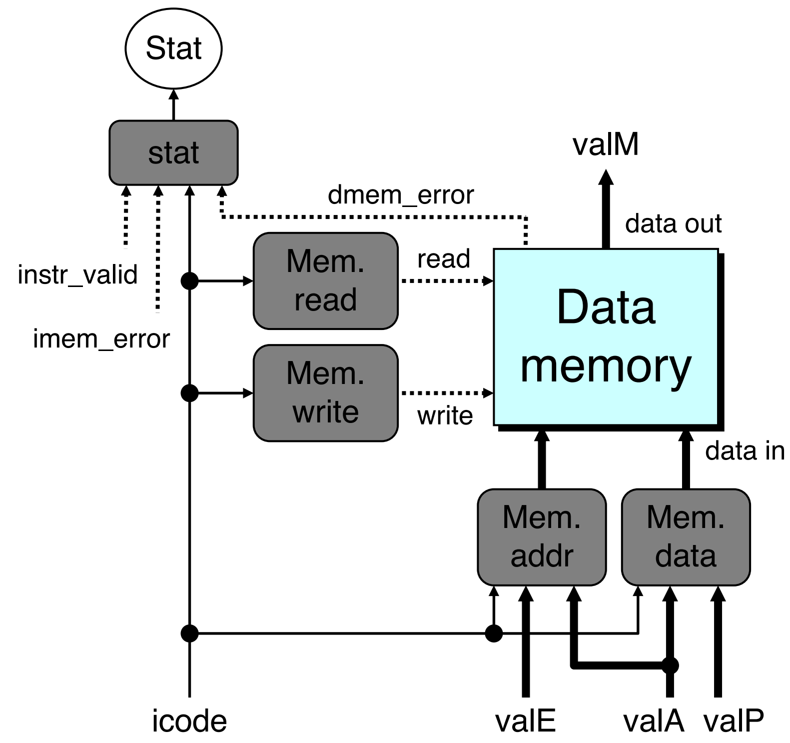
	jXX Dest	
Execute		No operation

	`call` Dest	
Execute	valE ← valB + –8	Decrement stack pointer

	`ret`	
Execute	valE ← valB + 8	Increment stack pointer

Memory Logic

- **Memory**
  - Reads or writes memory word
- **Control Logic**
  - stat: What is instruction status?
  - Mem. read: should word be read?
  - Mem. write: should word be written?
  - Mem. addr.: Select address
  - Mem. data.: Select data

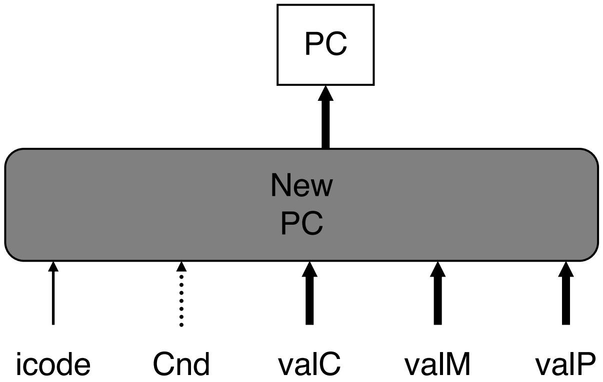


Memory Address

OPq rA, rB	Memory		No operation
`rmmovq` rA, D(rB)	Memory	$M_8[valE] \leftarrow valA$	Write value to memory
`popq` rA	Memory	$valM \leftarrow M_8[valA]$	Read from stack
jXX Dest	Memory		No operation
`call` Dest	Memory	$M_8[valE] \leftarrow valP$	Write return value on stack
`ret`	Memory	$valM \leftarrow M_8[valA]$	Read return address

PC Update Logic

- **New PC**
  - Select next value of PC

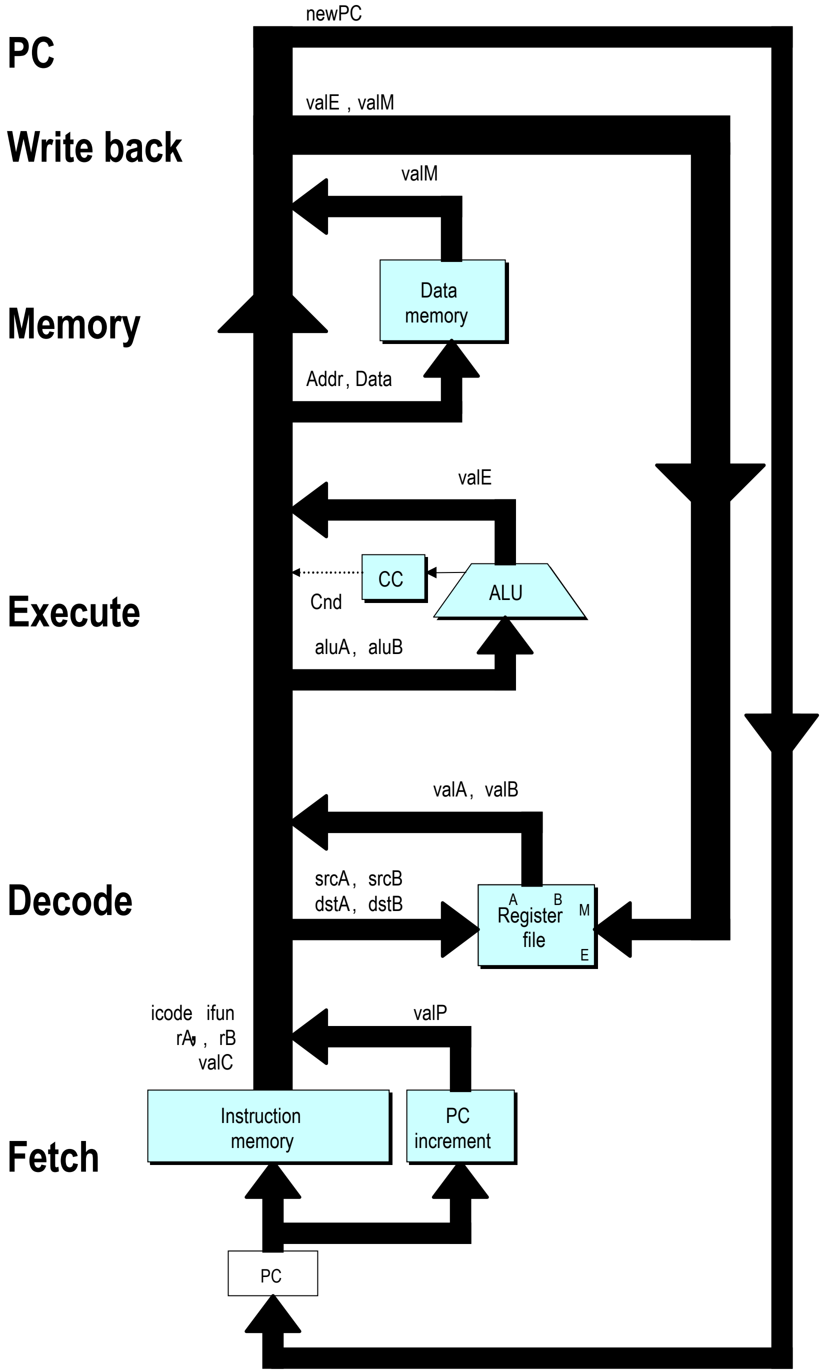


PC Update

	OPq rA, rB	
PC update	PC ← valP	Update PC
	`rmmovq` rA, D(rB)	
PC update	PC ← valP	Update PC
	`popq` rA	
PC update	PC ← valP	Update PC
	jXX Dest	
PC update	PC ← Cnd ? valC : valP	Update PC
	`call` Dest	
PC update	PC ← valC	Set PC to destination
	`ret`	
PC update	PC ← valM	Set PC to return address

SEQ Stages

- **Fetch**
  - Read instruction from instruction memory
- **Decode**
  - Read program registers
- **Execute**
  - Compute value or address
- **Memory**
  - Read or write data
- **Write Back**
  - Write program registers
- **PC**
  - Update program counter



SEQ Summary

- **Implementation**
  - Express every instruction as series of simple steps
  - Follow same general flow for each instruction type
  - Assemble registers, memories, predesigned combinational blocks
  - Connect with control logic
- **Limitations**
  - In one cycle, must propagate through instruction memory, register file, ALU, and data memory
  - Would need to run clock very slowly
  - Hardware units only active for fraction of clock cycle