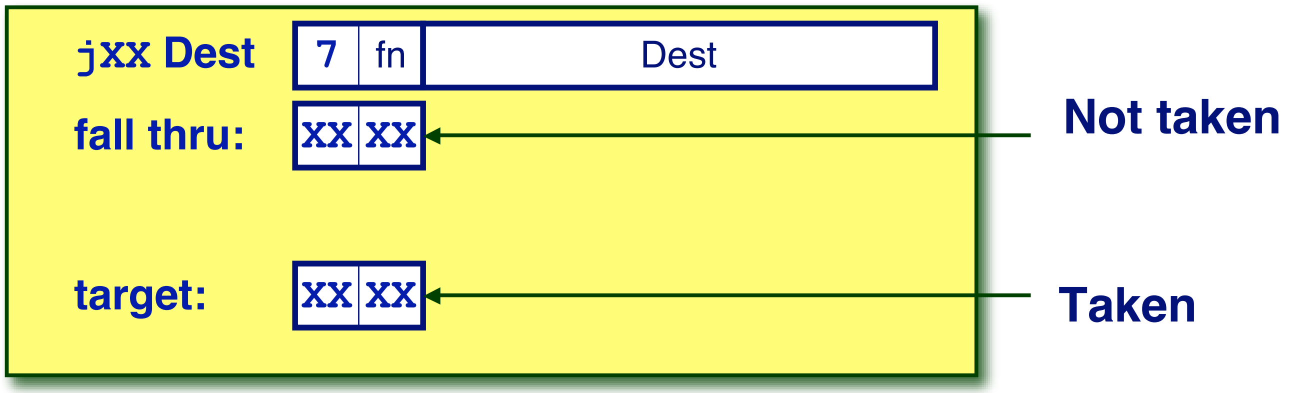
Systems I

Datapath Design II

Topics

- Control flow instructions
- Hardware for sequential machine (SEQ)

Executing Jumps



Fetch

- Read 5 bytes
- Increment PC by 5

Decode

- Do nothing

Execute

- Determine whether to take branch based on jump condition and condition codes

Memory

- Do nothing

Write back

- Do nothing

PC Update

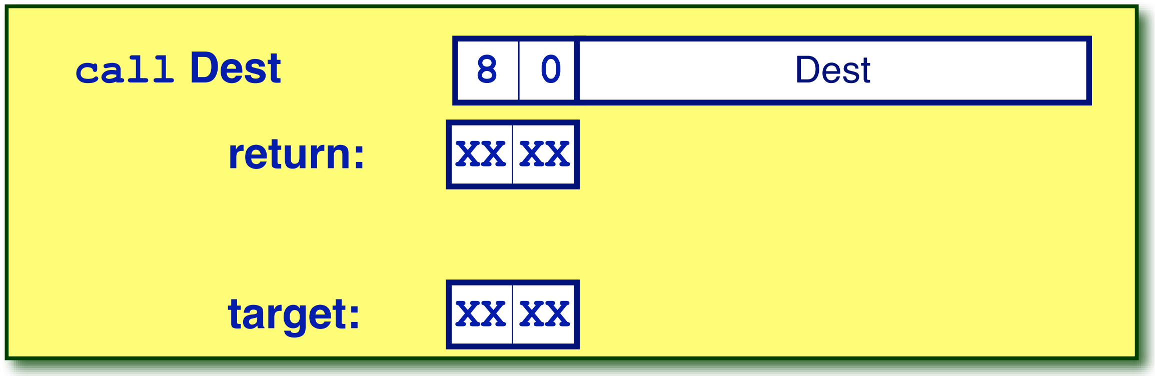
- Set PC to Dest if branch taken or to incremented PC if not branch

Stage Computation: Jumps

	jXX Dest	
Fetch	icode:ifun $\leftarrow$ M_1[PC]	Read instruction byte
	valC $\leftarrow$ M_4[PC+1]	Read destination address
	valP $\leftarrow$ PC+5	Fall through address
Decode		
Execute	Bch $\leftarrow$ Cond(CC,ifun)	Take branch?
Memory		
Write back		
PC update	PC $\leftarrow$ Bch ? valC : valP	Update PC

- Compute both addresses
- Choose based on setting of condition codes and branch condition

Executing `call`



Fetch

- Read 5 bytes
- Increment PC by 5

Decode

- Read stack pointer

Execute

- Decrement stack pointer by 4

Memory

- Write incremented PC to new value of stack pointer

Write back

- Update stack pointer

PC Update

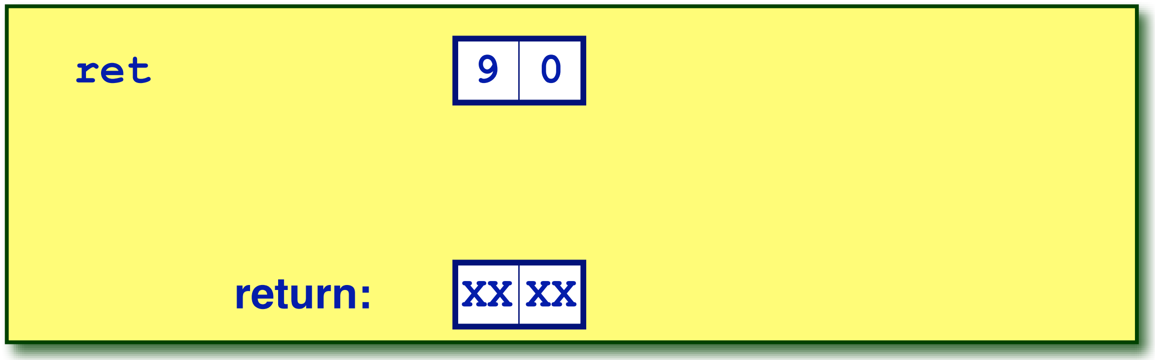
- Set PC to Dest

Stage Computation: `call`

	`call` Dest	
Fetch	icode:ifun ← M_1[PC]	Read instruction byte
	valC ← M_4[PC+1]	Read destination address
	valP ← PC+5	Compute return point
Decode	valB ← R[`%esp`]	Read stack pointer
Execute	valE ← valB + –4	Decrement stack pointer
Memory	M_4[valE] ← valP	Write return value on stack
Write back	R[`%esp`] ← valE	Update stack pointer
PC update	PC ← valC	Set PC to destination

- Use ALU to decrement stack pointer
- Store incremented PC

Executing `ret`



Fetch

- Read 1 byte

Decode

- Read stack pointer

Execute

- Increment stack pointer by 4

Memory

- Read return address from old stack pointer

Write back

- Update stack pointer

PC Update

- Set PC to return address

Stage Computation: `ret`

	`ret`	
Fetch	icode:ifun $\leftarrow$ M_1[PC]	Read instruction byte
Decode	valA $\leftarrow$ R[`%esp`] valB $\leftarrow$ R[`%esp`]	Read operand stack pointer Read operand stack pointer
Execute	valE $\leftarrow$ valB + 4	Increment stack pointer
Memory	valM $\leftarrow$ M_4[valA]	Read return address
Write back	R[`%esp`] $\leftarrow$ valE	Update stack pointer
PC update	PC $\leftarrow$ valM	Set PC to return address

- Use ALU to increment stack pointer
- Read return address from memory

Computation Steps

		OPl rA, rB	
Fetch	icode,ifun	icode:ifun $\leftarrow$ M_1[PC]	Read instruction byte
	rA,rB	rA:rB $\leftarrow$ M_1[PC+1]	Read register byte
	valC		[Read constant word]
	valP	valP $\leftarrow$ PC+2	Compute next PC
Decode	valA, srcA	valA $\leftarrow$ R[rA]	Read operand A
	valB, srcB	valB $\leftarrow$ R[rB]	Read operand B
Execute	valE	valE $\leftarrow$ valB OP valA	Perform ALU operation
	Cond code	Set CC	Set condition code register
Memory	valM		[Memory read/write]
Write back	dstE	R[rB] $\leftarrow$ valE	Write back ALU result
	dstM		[Write back memory result]
PC update	PC	PC $\leftarrow$ valP	Update PC

- All instructions follow same general pattern
- Differ in what gets computed on each step

Computation Steps

		`call` Dest	
Fetch	icode,ifun	icode:ifun $\leftarrow$ M_1[PC]	Read instruction byte
	rA,rB		[Read register byte]
	valC	valC $\leftarrow$ M_4[PC+1]	Read constant word
	valP	valP $\leftarrow$ PC+5	Compute next PC
Decode	valA, srcA		[Read operand A]
	valB, srcB	valB $\leftarrow$ R[`%esp`]	Read operand B
Execute	valE	valE $\leftarrow$ valB + –4	Perform ALU operation
	Cond code		[Set condition code reg.]
Memory	valM	M_4[valE] $\leftarrow$ valP	[Memory read/write]
Write back	dstE	R[`%esp`] $\leftarrow$ valE	[Write back ALU result]
	dstM		Write back memory result
PC update	PC	PC $\leftarrow$ valC	Update PC

- All instructions follow same general pattern
- Differ in what gets computed on each step

Computed Values

Fetch

icode	Instruction code
ifun	Instruction function
rA	Instr. Register A
rB	Instr. Register B
valC	Instruction constant
valP	Incremented PC

Decode

srcA	Register ID A
srcB	Register ID B
dstE	Destination Register E
dstM	Destination Register M
valA	Register value A
valB	Register value B

Execute

- valE ALU result
- Bch Branch flag

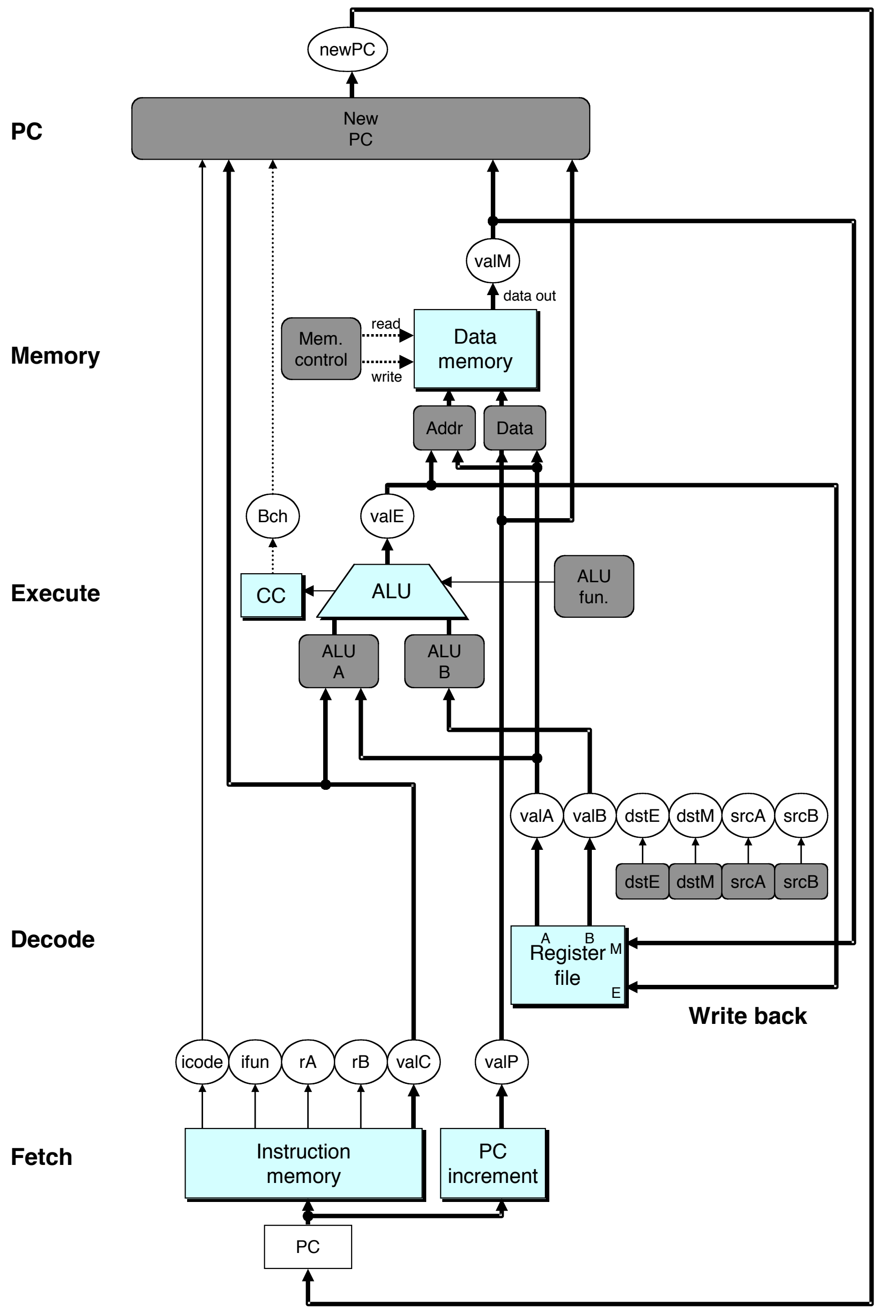
Memory

- valM Value from memory

SEQ Hardware

Key

- **Blue boxes: predesigned hardware blocks**
  - **E.g., memories, ALU**
- **Gray boxes: control logic**
  - **Describe in HCL**
- **White ovals: labels for signals**
- **Thick lines: 32-bit word values**
- **Thin lines: 4-8 bit values**
- **Dotted lines: 1-bit values**



Summary

Today

- Control flow instructions
- Hardware for sequential machine (SEQ)

Next time

- Control logic for instruction execution
- Timing and clocking