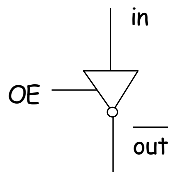
Computer Organization

- Computer design as an application of digital logic design procedures
- Computer = processing unit + memory system
- Processing unit = control + datapath
- Control = finite state machine
  - Inputs = machine instruction, datapath conditions
  - Outputs = register transfer control signals, ALU operation codes
  - Instruction interpretation = instruction fetch, decode, execute
- Datapath = functional units + registers
  - Functional units = ALU, multipliers, dividers, etc.
  - Registers = program counter, shifters, storage registers

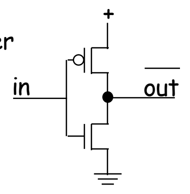
Tri-State Buffers

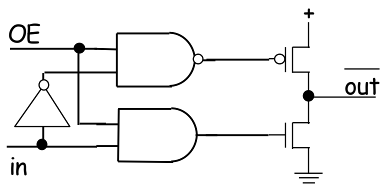
- 0, 1, Z (high impedance state)



if OE then Out = $\overline{\text{In}}$
else “disconnected”

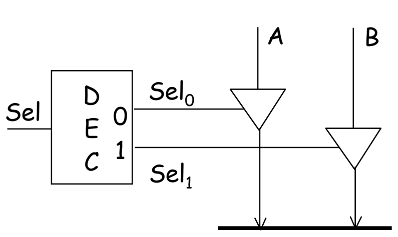
Basic Inverter

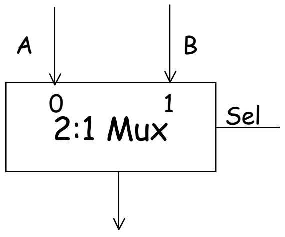




Inverting Buffer

Tri-States vs. Mux

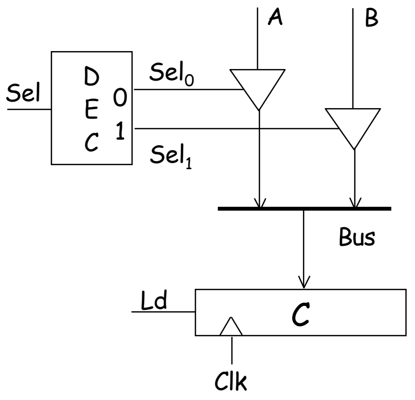




Buffer circuits simple!
Scales nicely for high fan-in
and wide bit widths!

Scales poorly for high fan-in
or wide bit widths

Register Transfer

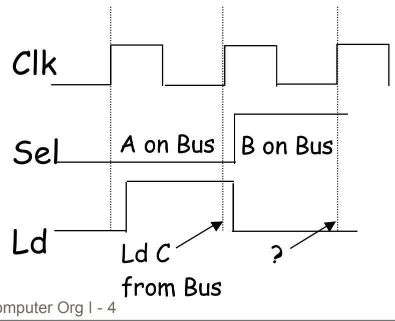


$C \leftarrow A$

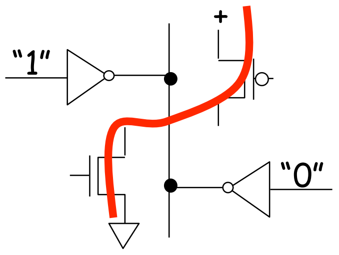
$Sel \leftarrow 0;\ Ld \leftarrow 1$

$C \leftarrow B$

$Sel \leftarrow 1;\ Ld \leftarrow 1$

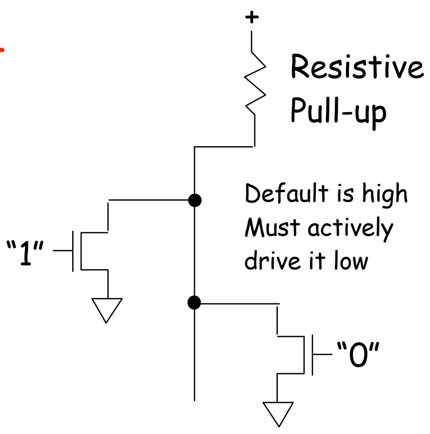


Open Collector Concept



Bad! Short circuit!

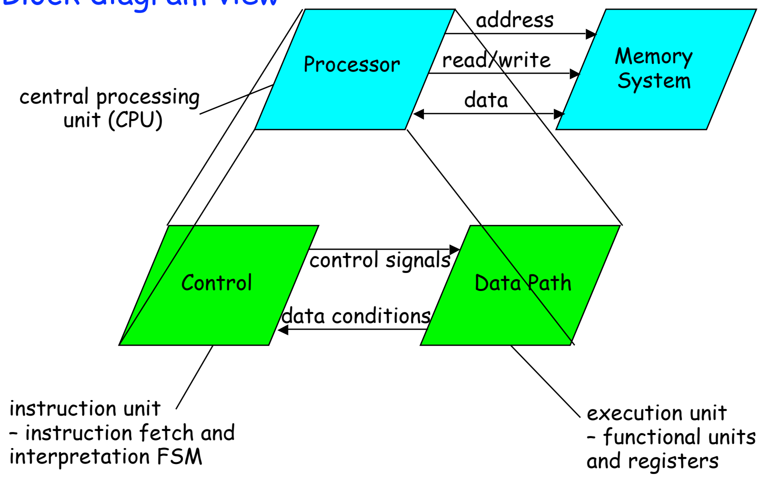
Low resistance path from Vdd to Gnd



Wired AND Configuration:
If *any* attached device wants wire to be "0", it wins
If *all* attached devices want wire to be "1", it is

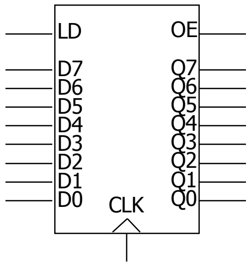
Structure of a Computer

- Block diagram view


Processor
Memory System
address
read/write
data
central processing unit (CPU)
Control
Data Path
control signals
data conditions
instruction unit – instruction fetch and interpretation FSM
execution unit – functional units and registers

Registers

- Selectively loaded – EN or LD input
- Output enable – OE input
- Multiple registers – group 4 or 8 in parallel

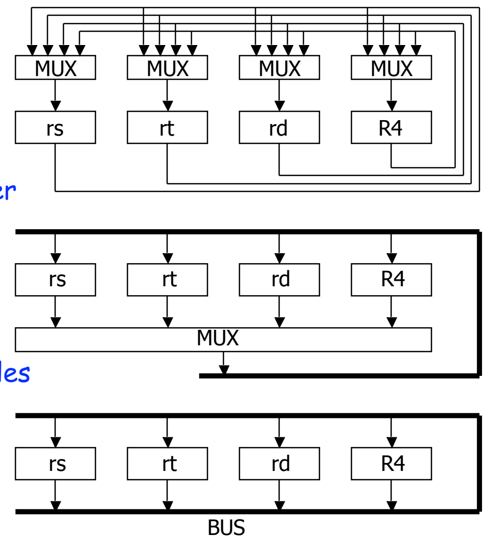


OE asserted causes FF state to be connected to output pins; otherwise they are left unconnected (high impedance)

LD asserted during a lo-to-hi clock transition loads new data into FFs

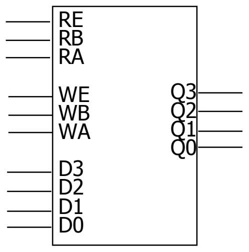
Register Transfer

- Point-to-point connection
  - Dedicated wires
  - Muxes on inputs of each register
- Common input from multiplexer
  - Load enables for each register
  - Control signals for multiplexer
- Common bus with output enables
  - Output enables and load enables for each register



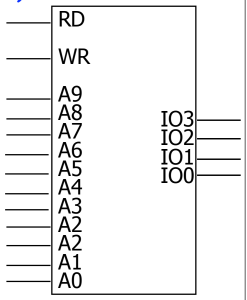
Register Files

- Collections of registers in one package
  - Two-dimensional array of FFs
  - Address used as index to a particular word
  - Separate read and write addresses so can do both at same time
- 4 by 4 register file
  - 16 D-FFs
  - Organized as four words of four bits each
  - Write-enable (load)
  - Read-enable (output enable)



Memories

- Larger Collections of Storage Elements
  - Implemented not as FFs but as much more efficient latches
  - High-density memories use 1-5 switches (transitors) per bit
- Static RAM – 1024 words each 4 bits wide
  - Once written, memory holds forever (not true for denser dynamic RAM)
  - Address lines to select word (10 lines for 1024 words)
  - Read enable
    - Same as output enable
    - Often called chip select
    - Permits connection of many chips into larger array
  - Write enable (same as load enable)
  - Bi-directional data lines
    - output when reading, input when writing



Instruction Sequencing

- Example – an instruction to add the contents of two registers (Rx and Ry) and place result in a third register (Rz)
- Step 1: Get the ADD instruction from memory into an instruction register
- Step 2: Decode instruction
  - Instruction in IR has the code of an ADD instruction
  - Register indices used to generate output enables for registers Rx and Ry
  - Register index used to generate load signal for register Rz
- Step 3: Execute instruction
  - Enable Rx and Ry output and direct to ALU
  - Setup ALU to perform ADD operation
  - Direct result to Rz so that it can be loaded into register

Instruction Types

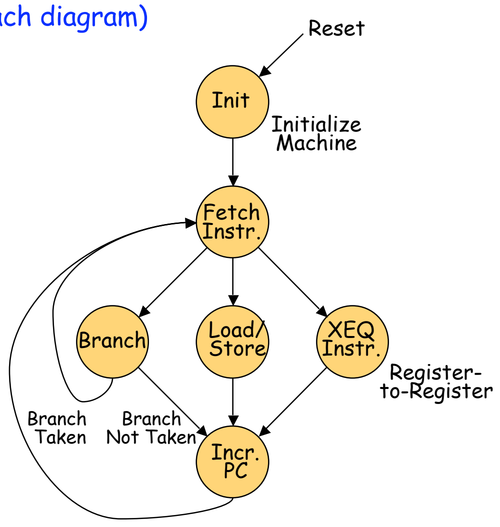
- Data Manipulation
  - Add, subtract
  - Increment, decrement
  - Multiply
  - Shift, rotate
  - Immediate operands
- Data Staging
  - Load/store data to/from memory
  - Register-to-register move
- Control
  - Conditional/unconditional branches in program flow
  - Subroutine call and return

Elements of the Control Unit (aka Instruction Unit)

- Standard FSM Elements
  - State register
  - Next-state logic
  - Output logic (datapath/control signaling)
  - Moore or synchronous Mealy machine to avoid loops unbroken by FF
- Plus Additional "Control" Registers
  - Instruction register (IR)
  - Program counter (PC)
- Inputs/Outputs
  - Outputs control elements of data path
  - Inputs from data path used to alter flow of program (test if zero)

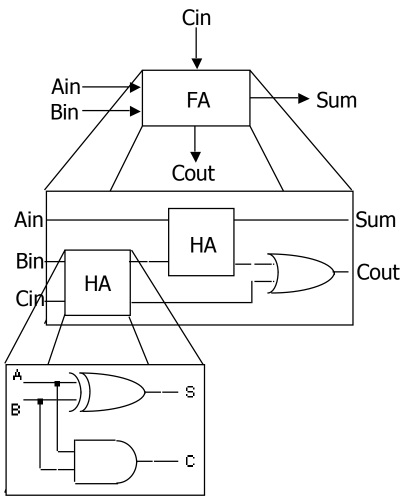
Instruction Execution

- Control State Diagram (for each diagram)
  - Reset
  - Fetch instruction
  - Decode
  - Execute
- Instructions partitioned into three classes
  - Branch
  - Load/store
  - Register-to-register
- Different sequence through diagram for each instruction type



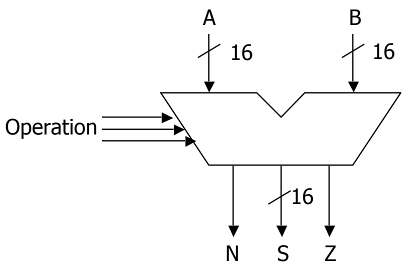
Data Path (Hierarchy)

- Arithmetic circuits constructed in hierarchical and iterative fashion
  - Each bit in datapath is functionally identical
  - 4-bit, 8-bit, 16-bit, 32-bit datapaths



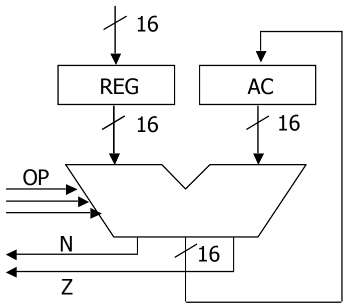
Data Path (ALU)

- ALU Block Diagram
  - Input: data and operation to perform
  - Output: result of operation and status information



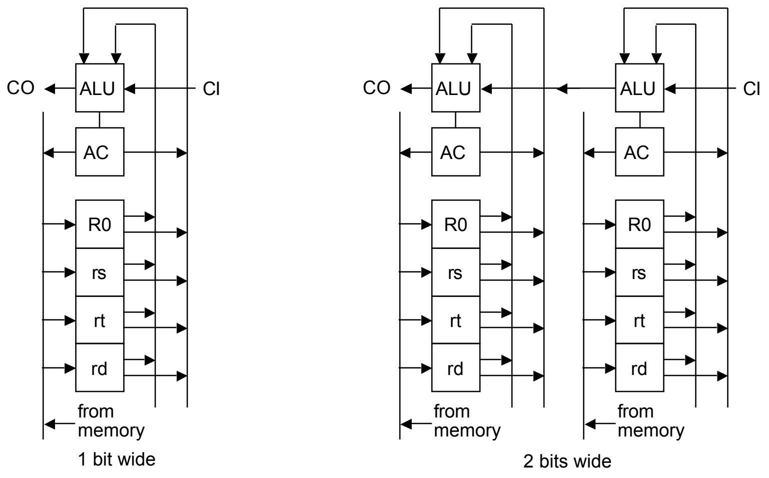
Data Path (ALU + Registers)

- Accumulator
  - Special register
  - One of the inputs to ALU
  - Output of ALU stored back in accumulator
- One-address instructions
  - Operation and address of one operand
  - Other operand and destination is accumulator register
  - AC <- AC op Mem[addr]
  - "Single address instructions" (AC implicit operand)
- Multiple registers
  - Part of instruction used to choose register operands



Data Path (Bit-slice)

- Bit-slice concept: iterate to build n-bit wide datapaths



Announcements

- Quizzes: Good news and bad news!
- Additional readings on-line: CLD 1ed Chapters 11, 12
- Lab Checkpoints and Project
  - Project is a marathon, not a sprint
  - Not as completely specified or as straightforward as the labs: creativity, team work as well as technical skill required
  - Do NOT fall behind … schedule may appear to look slack, but it probably won't be possible to catch up if you fall behind
  - Partner problems: Keep us informed! Don't let it fester!
  - Keep up with your TA design reviews. This is really important! Take them seriously!

Announcements

- After HW #6, length of hws will decrease …
- HW #5, Q2 Re-revised specification (reposted to web):
  - N-S: Green Arrow 16, Yellow Arrow 8, Red 88, Green 24, Yellow 8 (NOTE: Red overlaps Green and Yellow Arrows for 24 seconds, leaving 64 seconds with the arrows off but the Red light on)
  - E-W: Red 56, Green 56, Yellow 8
  - Suppose each column represents an 8 second interval:

N-S

```
GA GA YA RA RA RA RA RA RA RA RA RA RA RA RA
R  R  R  G  G  G  Y  R  R  R  R  R  R  R  R
```

E-W

```
R  R  R  R  R  R  R  G  G  G  G  G  G  G  Y
```

Instruction Path

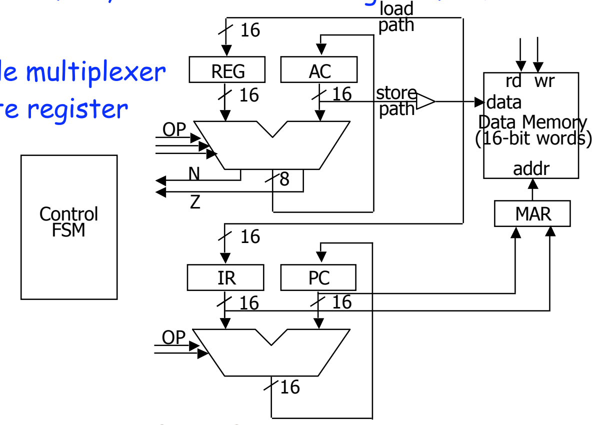
- Program Counter
  - Keeps track of program execution
  - Address of next instruction to read from memory
  - May have auto-increment feature or use ALU
- Instruction Register
  - Current instruction
  - Includes ALU operation and address of operand
  - Also holds target of jump instruction
  - Immediate operands
- Relationship to Data Path
  - PC may be incremented through ALU
  - Contents of IR may also be required as input to ALU

Data Path (Memory Interface)

- Memory
  - Separate data and instruction memory (Harvard architecture)
    - Two address busses, two data busses
  - Single combined memory (Princeton architecture)
    - Single address bus, single data bus
- Separate memory
  - ALU output goes to data memory input
  - Register input from data memory output
  - Data memory address from instruction register
  - Instruction register from instruction memory output
  - Instruction memory address from program counter
- Single memory
  - Address from PC or IR
  - Memory output to instruction and data registers
  - Memory input from ALU output

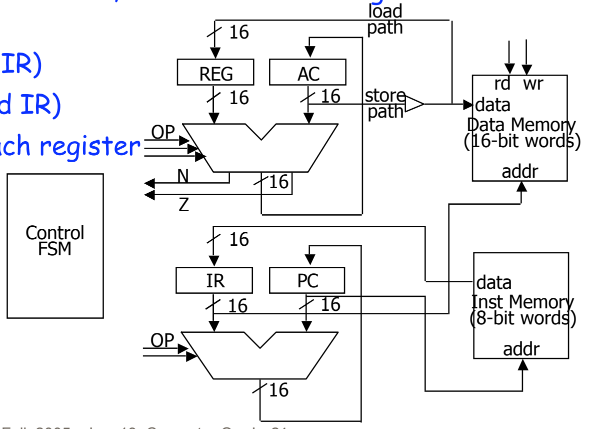
Block Diagram of Processor

- Register Transfer View of Princeton Architecture
  - Which register outputs are connected to which register inputs
  - Arrows represent data-flow, other are control signals from control FSM
  - MAR may be a simple multiplexer rather than separate register
  - MBR is split in two (REG and IR)
  - Load control for each register



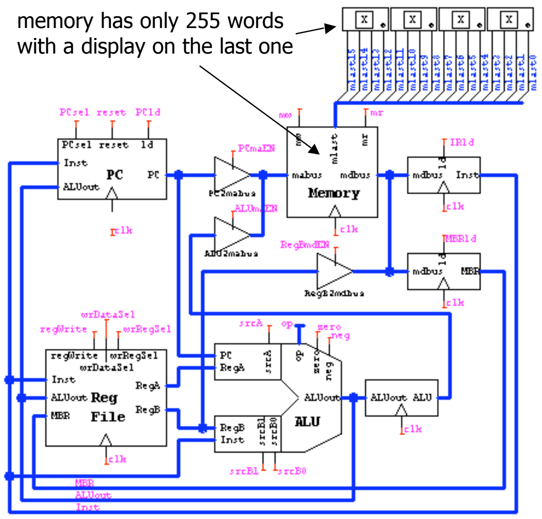
Block Diagram of Processor

- Register transfer view of Harvard architecture
  - Which register outputs are connected to which register inputs
  - Arrows represent data-flow, other are control signals from control FSM
  - Two MARs (PC and IR)
  - Two MBRs (REG and IR)
  - Load control for each register



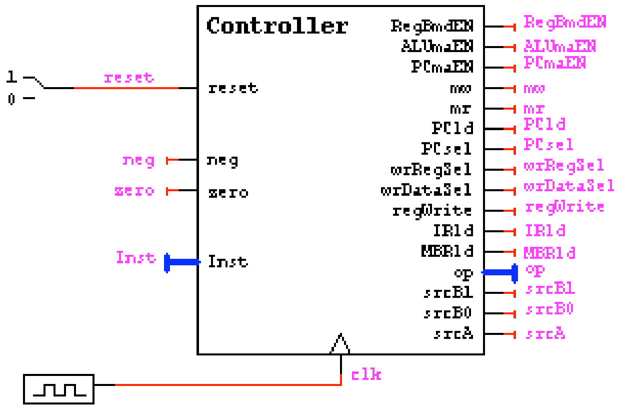
A Simplified Processor Data-path and Memory

- Princeton architecture
- Register file
- Instruction register
- PC incremented through ALU
- Modeled after MIPS rt000 (used in 61C textbook by Patterson & Hennessy)
  - Really a 32 bit machine
  - We'll do a 16 bit version



Processor Control

- Synchronous Mealy machine
- Multiple cycles per instruction



Processor Instructions

- Three principal types (16 bits in each instruction)

type	op	rs	rt	rd	funct
R(egister)	3	3	3	3	4
I(mmediate)	3	3	3	7	
J(ump)	3	13			

- Some of the instructions

R	add	0	rs	rt	rd	0	rd = rs + rt
	sub	0	rs	rt	rd	1	rd = rs - rt
	and	0	rs	rt	rd	2	rd = rs & rt
	or	0	rs	rt	rd	3	rd = rs | rt
	slt	0	rs	rt	rd	4	rd = (rs < rt)
I	lw	1	rs	rt	offset		rt = mem[rs + offset]
	sw	2	rs	rt	offset		mem[rs + offset] = rt
	beq	3	rs	rt	offset		pc = pc + offset, if (rs == rt)
	addi	4	rs	rt	offset		rt = rs + offset
J	j	5	target address				pc = target address
	halt	7	-				stop execution until reset

Tracing an Instruction's Execution

- Instruction: r3 = r1 + r2

R	0	rs=r1	rt=r2	rd=r3	funct=0

- 1. Instruction fetch
  - Move instruction address from PC to memory address bus
  - Assert memory read
  - Move data from memory data bus into IR
  - Configure ALU to add 1 to PC
  - Configure PC to store new value from ALUout
- 2. Instruction decode
  - Op-code bits of IR are input to control FSM
  - Rest of IR bits encode the operand addresses (rs and rt)
    - These go to register file

Tracing an Instruction's Execution (cont'd)

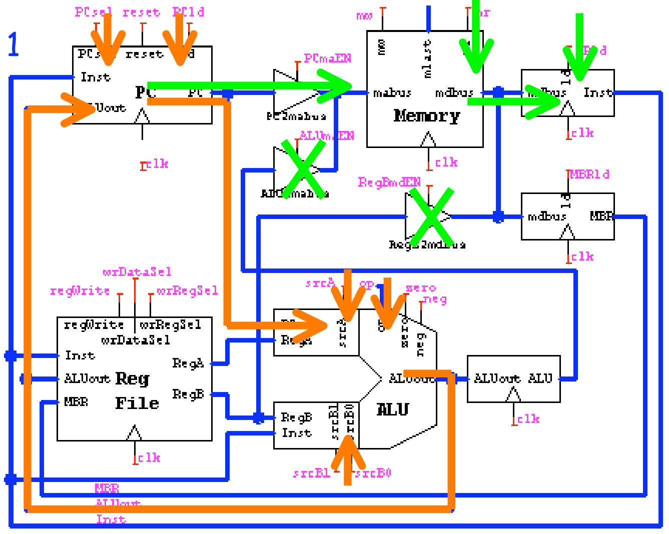
- Instruction: r3 = r1 + r2

R	0	rs=r1	rt=r2	rd=r3	funct=0

- 3. Instruction execute
  - Set up ALU inputs
  - Configure ALU to perform ADD operation
  - Configure register file to store ALU result (rd)

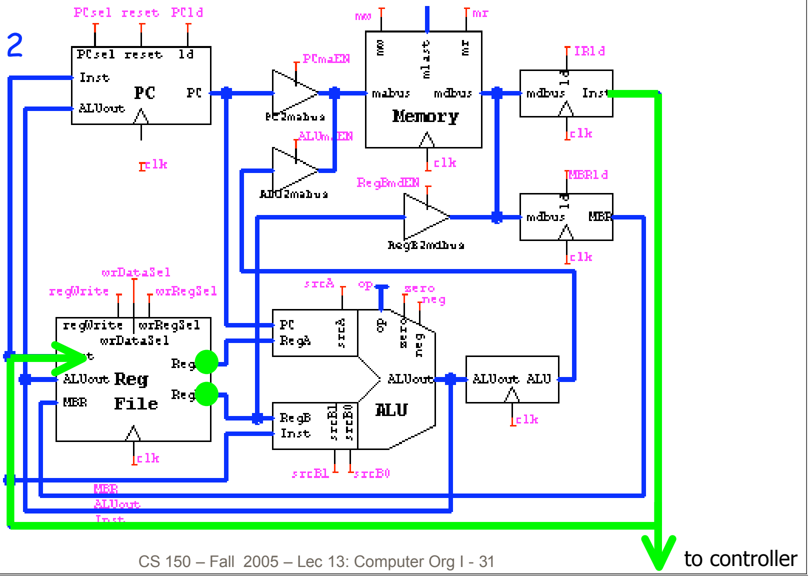
Tracing an Instruction's Execution (cont'd)

- Step 1



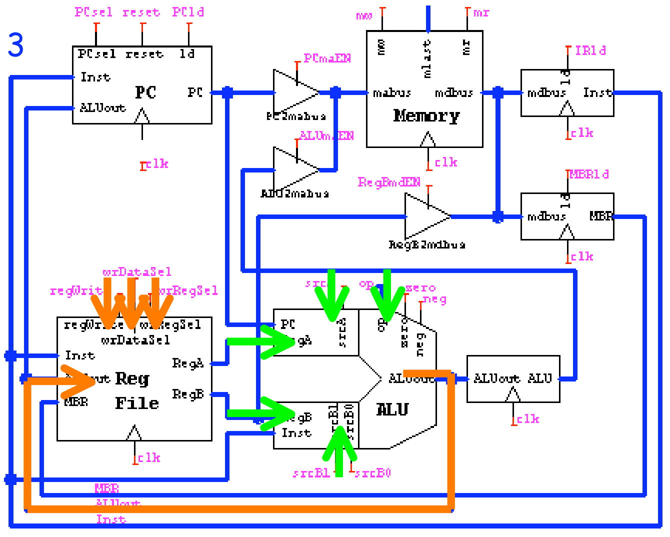
Tracing an Instruction's Execution (cont'd)

- Step 2



Tracing an Instruction's Execution (cont'd)

- Step 3



Register-Transfer-Level Description

- Control
  - Transfer data btwn registers by asserting appropriate control signals
- Register transfer notation: work from register to register
  - Instruction fetch:

```
    mabus ← PC;      - move PC to memory address bus (PCmaEN, ALUmaEN)
    memory read;     - assert memory read signal (mr, RegBmdEN)
    IR ← memory;     - load IR from memory data bus (IRld)
    op ← add         - send PC into A input, 1 into B input, add
                          (srcA, srcB0, scrB1, op)
    PC ← ALUout      - load result of incrementing in ALU into PC (PCld, PCsel)
```

  - Instruction decode:

```
    IR to controller
    values of A and B read from register file (rs, rt)
```

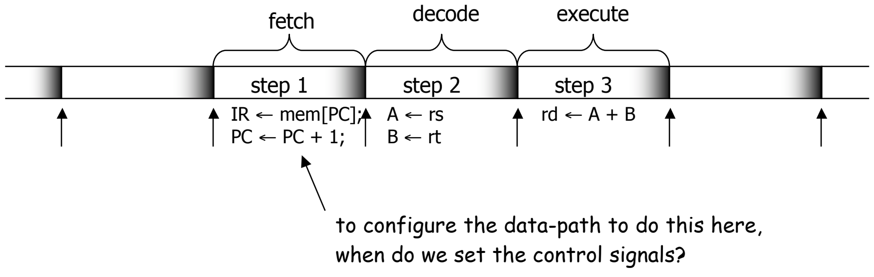
  - Instruction execution:

```
    op ← add         - send regA into A input, regB into B input, add
                          (srcA, srcB0, scrB1, op)
    rd ← ALUout      - store result of add into destination register
                          (regWrite, wrDataSel, wrRegSel)
```

Register-Transfer-Level Description (cont'd)

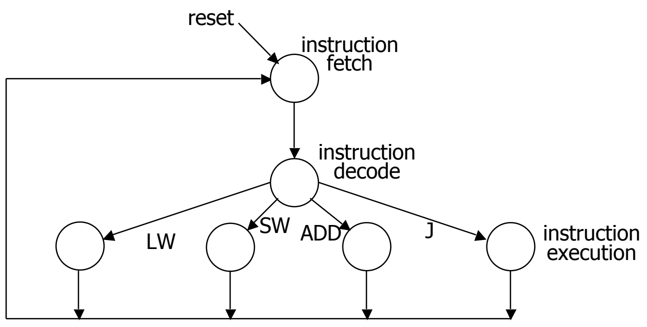
- How many states are needed to accomplish these transfers?
  - Data dependencies (where do values that are needed come from?)
  - Resource conflicts (ALU, busses, etc.)
- In our case, it takes three cycles
  - One for each step
  - All operation within a cycle occur between rising edges of the clock
- How do we set all of the control signals to be output by the state machine?
  - Depends on the type of machine (Mealy, Moore, synchronous Mealy)

Review of FSM Timing



FSM Controller for CPU (skeletal Moore FSM)

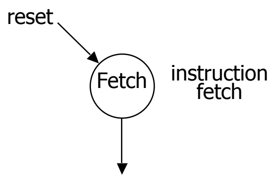
- First pass at deriving the state diagram (Moore Machine)
  - These will be further refined into sub-states



FSM Controller for CPU (reset and instruction fetch)

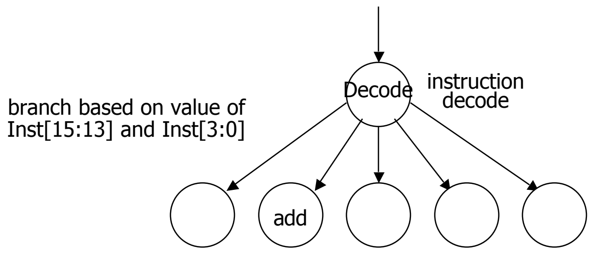
- Assume Moore Machine
  - Outputs associated with states rather than arcs
- Reset state and instruction fetch sequence
- On reset (go to Fetch state)
  - Start fetching instructions
  - PC will set itself to zero

mabus ← PC;
memory read;
IR ← memory data bus;
PC ← PC + 1;



FSM Controller for CPU (decode)

- Operation Decode State
  - Next state branch based on operation code in instruction
  - Read two operands out of register file
    - What if the instruction doesn't have two operands?

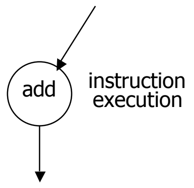


FSM Controller for CPU (Instruction Execution)

- For add instruction
  - Configure ALU and store result in register

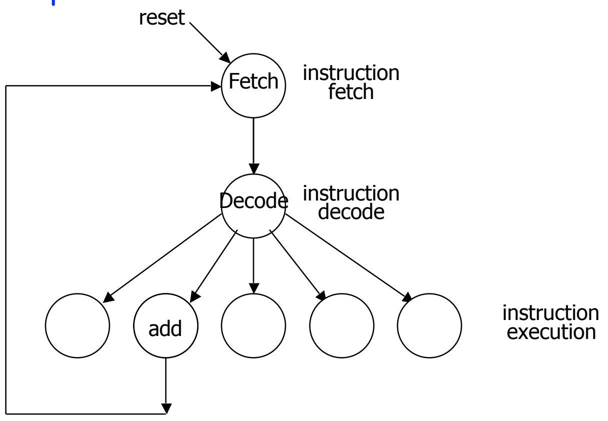
    rd ← A + B

  - Other instructions may require multiple cycles



FSM Controller for CPU (Add Instruction)

- Putting it all together and closing the loop
  - the famous instruction fetch decode execute cycle



FSM Controller for CPU

- Now we need to repeat this for all the instructions of our processor
  - Fetch and decode states stay the same
  - Different execution states for each instruction
    - Some may require multiple states if available register transfer paths require sequencing of steps