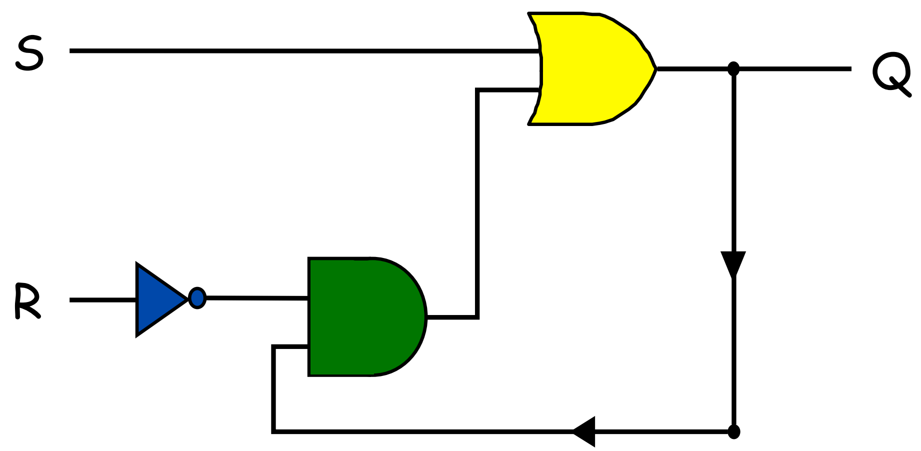
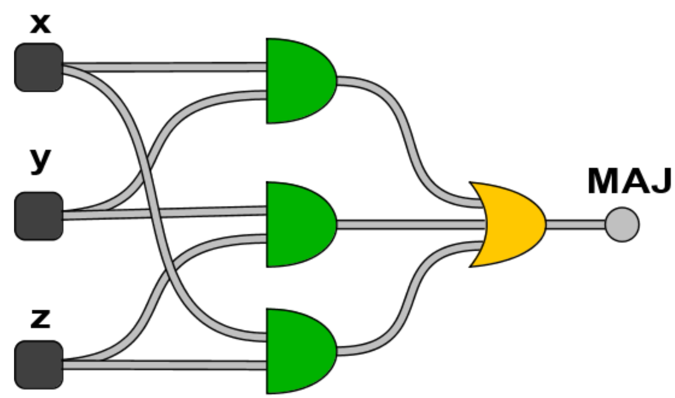
More Sequential Circuits, plus Architecture



Sequential vs. Combinational Circuits

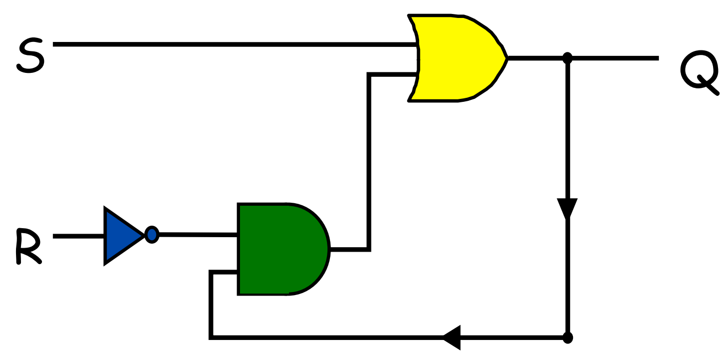
Combinational circuits.

- Output determined solely by inputs.
- Can draw solely with left-to-right signal paths.



Sequential circuits.

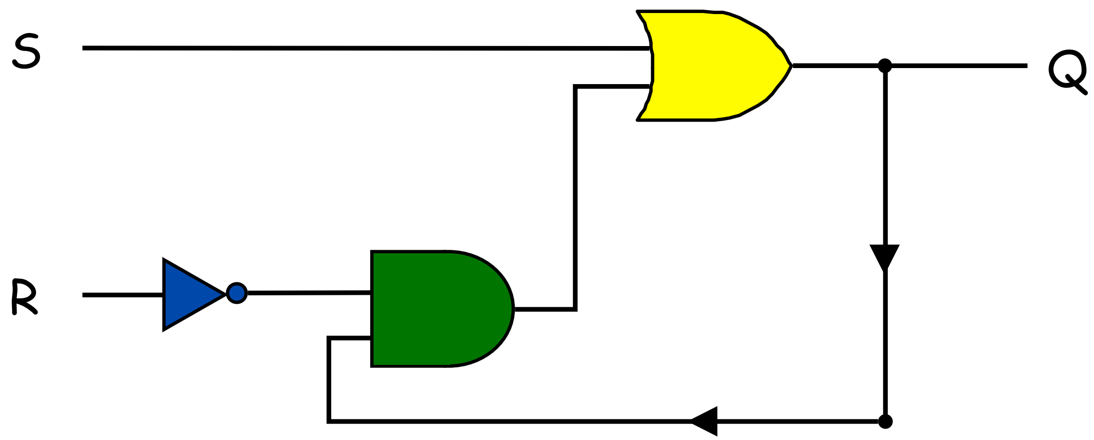
- Output determined by inputs AND previous outputs.
- Feedback loop.



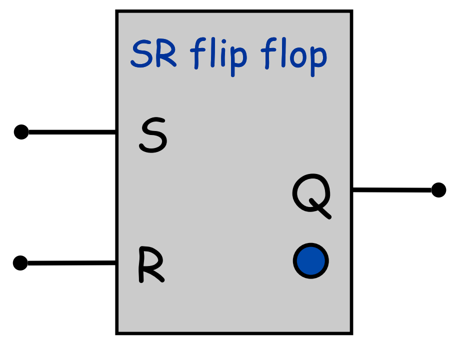
SR Flip-Flop

SR Flip-Flop.

- S = 1, R = 0 (set) ⇒ "Flips" bit on.
- S = 0, R = 1 (reset) ⇒ "Flops" bit off.
- S = R = 0 ⇒ Status quo.
- S = R = 1 ⇒ Not allowed.



Implementation

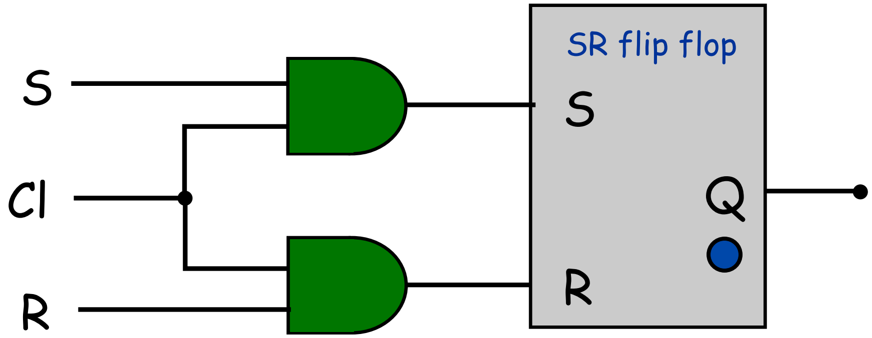


Interface

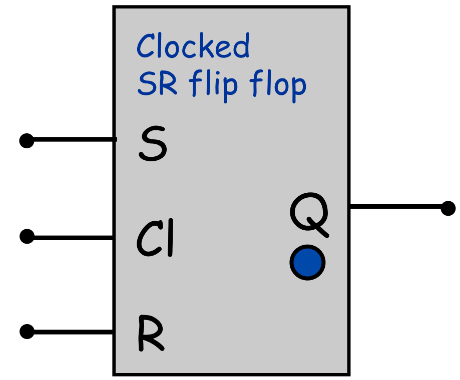
Clocked SR Flip-Flop

Clocked SR Flip-Flop.

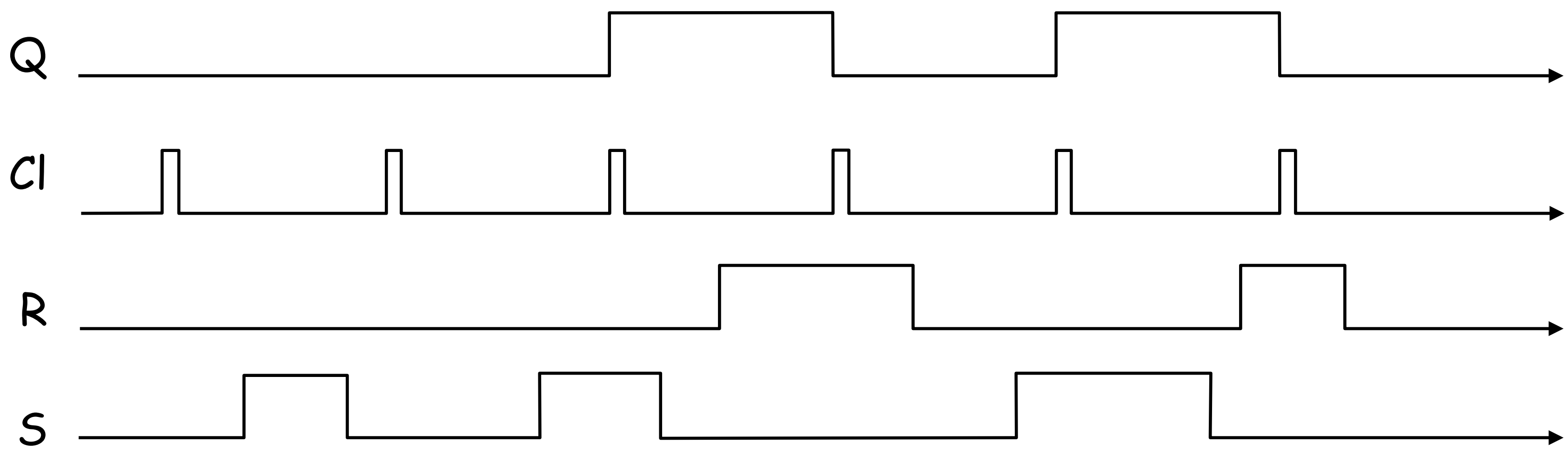
- Same as SR flip-flop except S and R only active when clock is 1.



Implementation



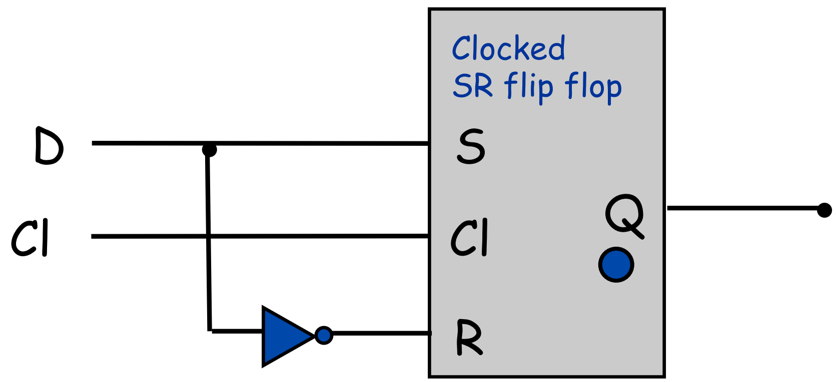
Interface



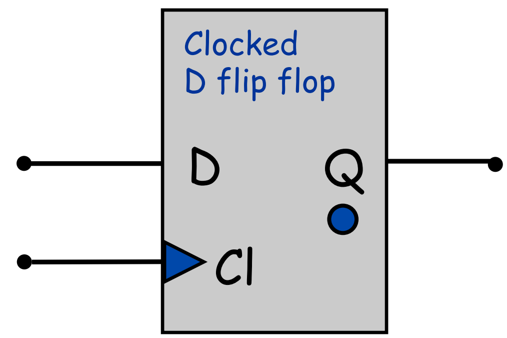
Clocked D Flip-Flop

Clocked D Flip-Flop.

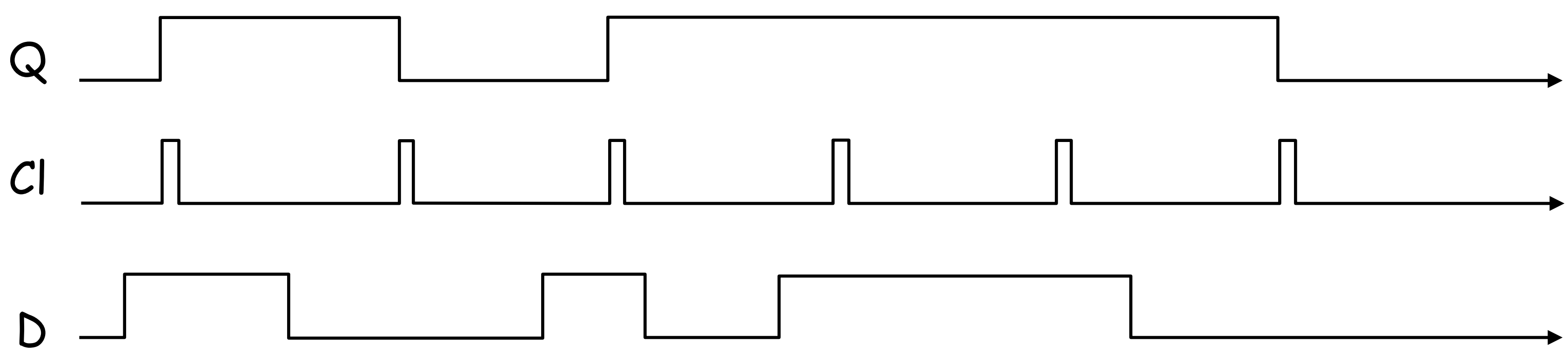
- Output follows D input while clock is 1.
- Output is remembered while clock is 0.



Implementation



Interface



Memory Overview

Computers and TOY have many types of memory.

- Program counter.
- Registers.
- Main memory.

We implement each bit of memory with a clocked D flip-flop.

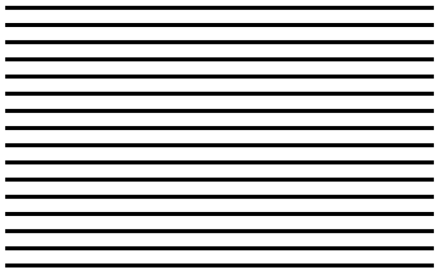
Need mechanism to organize and manipulate GROUPS of related bits.

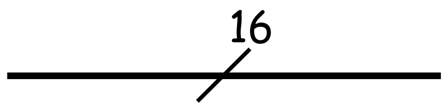
- TOY has 16-bit words.
- Memory hierarchy makes architecture manageable.

Bus

16-bit bus.

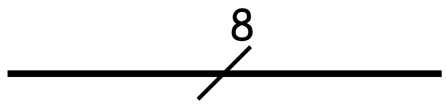
- Bundle of 16 wires.
- Memory transfer, register transfer.





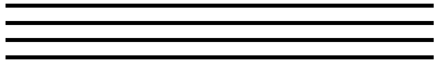
8-bit bus.

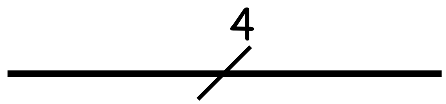
- Bundle of 8 wires.
- TOY memory address.



4-bit bus.

- Bundle of 4 wires.
- TOY register address.



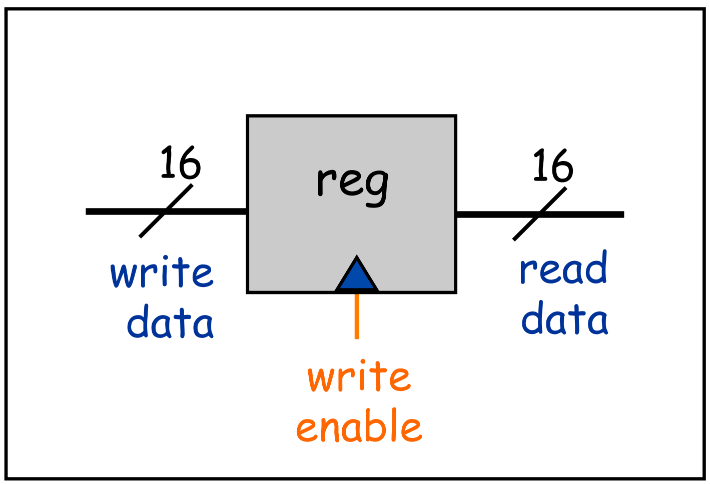


Stand-Alone Register

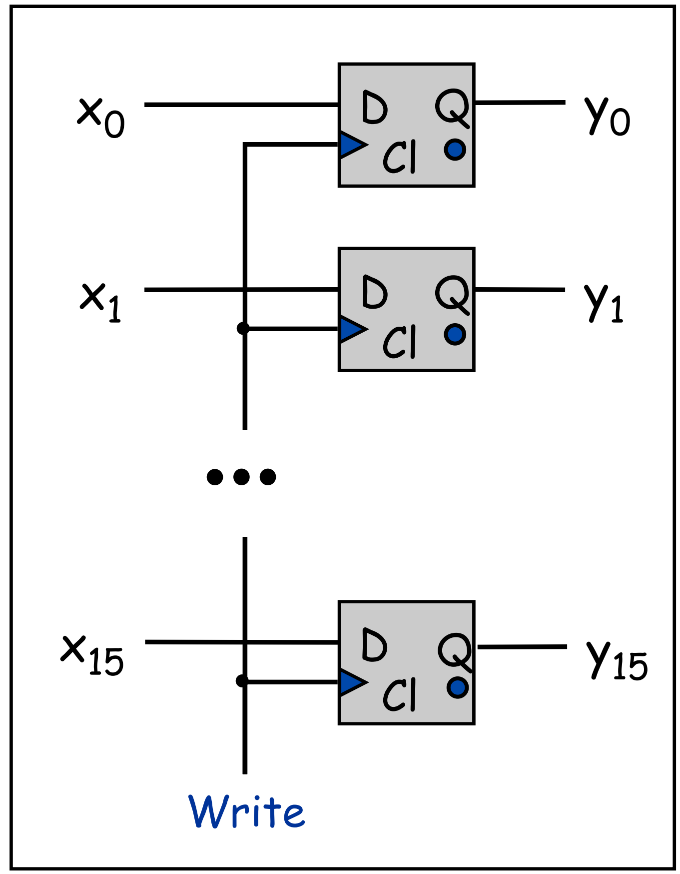
k-bit register.

- Stores k bits.
- Register contents always available on output.
- If write enable is asserted, k input bits get copied into register.

Ex: Program Counter, 16 TOY registers, 256 TOY memory locations.



16-bit Register Interface



16-bit Register Implementation

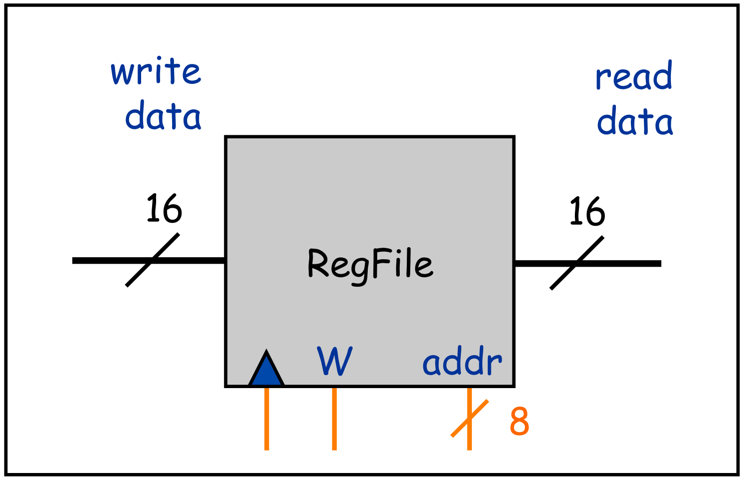
Register File Interface

n-by-k register file.

- Bank of n registers; each stores k bits.
- Read and write information to *one* of n registers.
  - $\log_2 n$ address inputs specifies which one
- Addressed bits always appear on output.
- If write enable and clock are asserted, k input bits are copied into addressed register.

Examples.

- TOY registers: n = 16, k = 16.
- TOY main memory: n = 256, k = 16.
- Real computer: n = 256 million, k = 32.
  - 1 GB memory
  - (1 Byte = 8 bits)

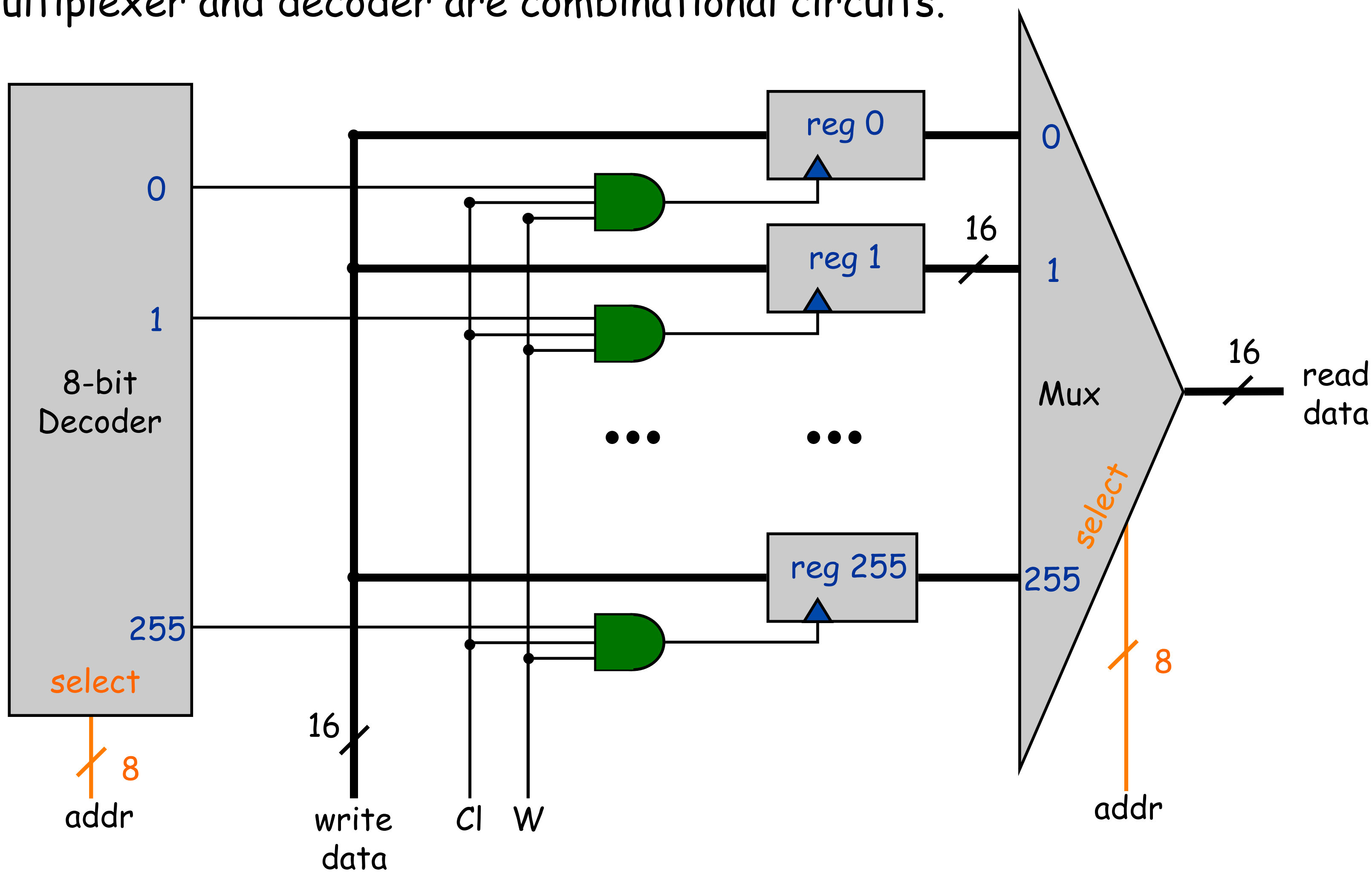


256 x 16 Register File Interface

Register File Implementation

Implementation example: TOY main memory.

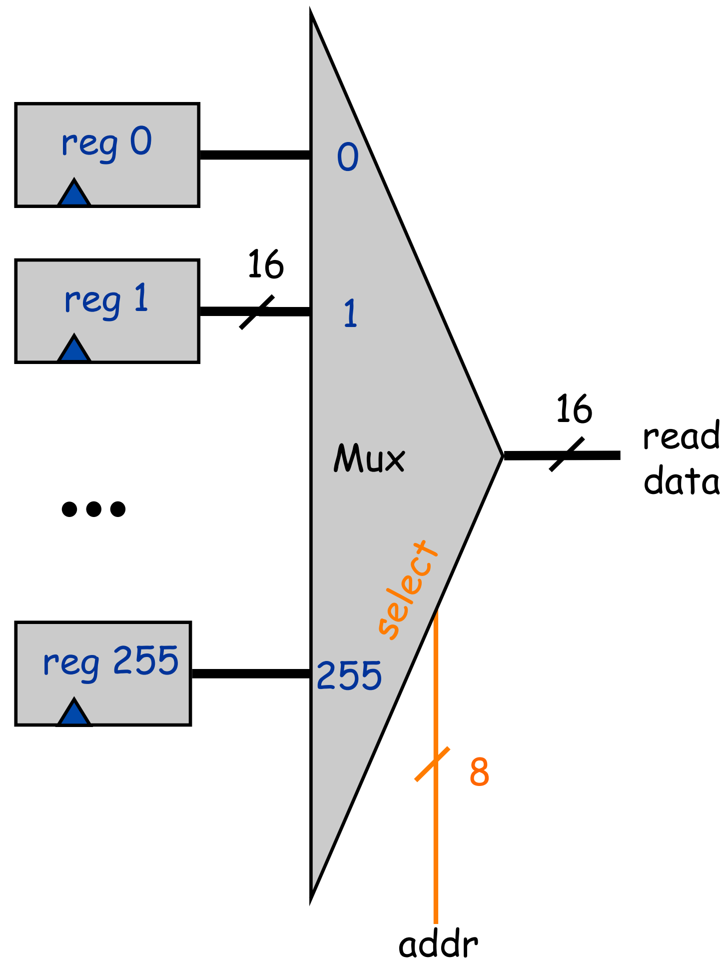
- Use 256 16-bit registers.
- Multiplexer and decoder are combinational circuits.



Register File Implementation: Reading

Implementation example: TOY main memory.

- Use 256 16-bit registers.
- Multiplexer is combinational circuit.

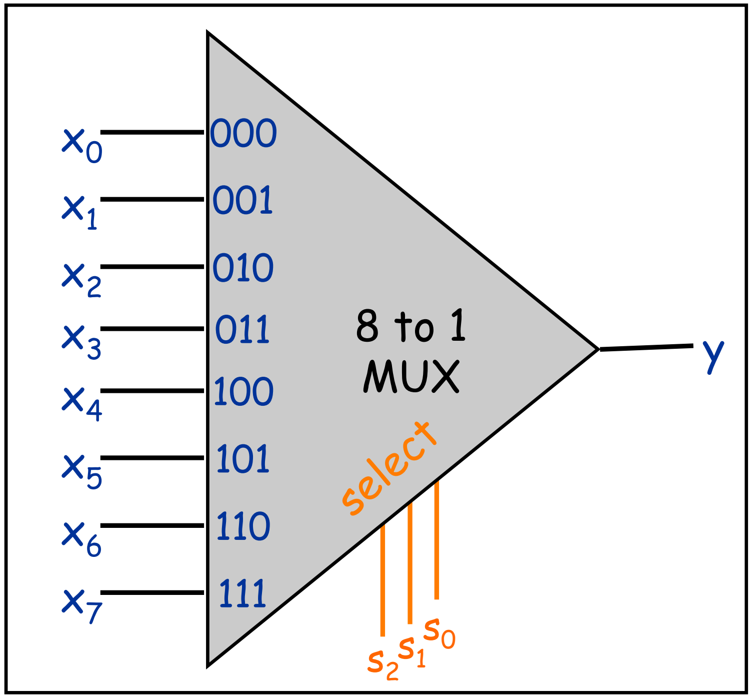


2^n-to-1 Multiplexer

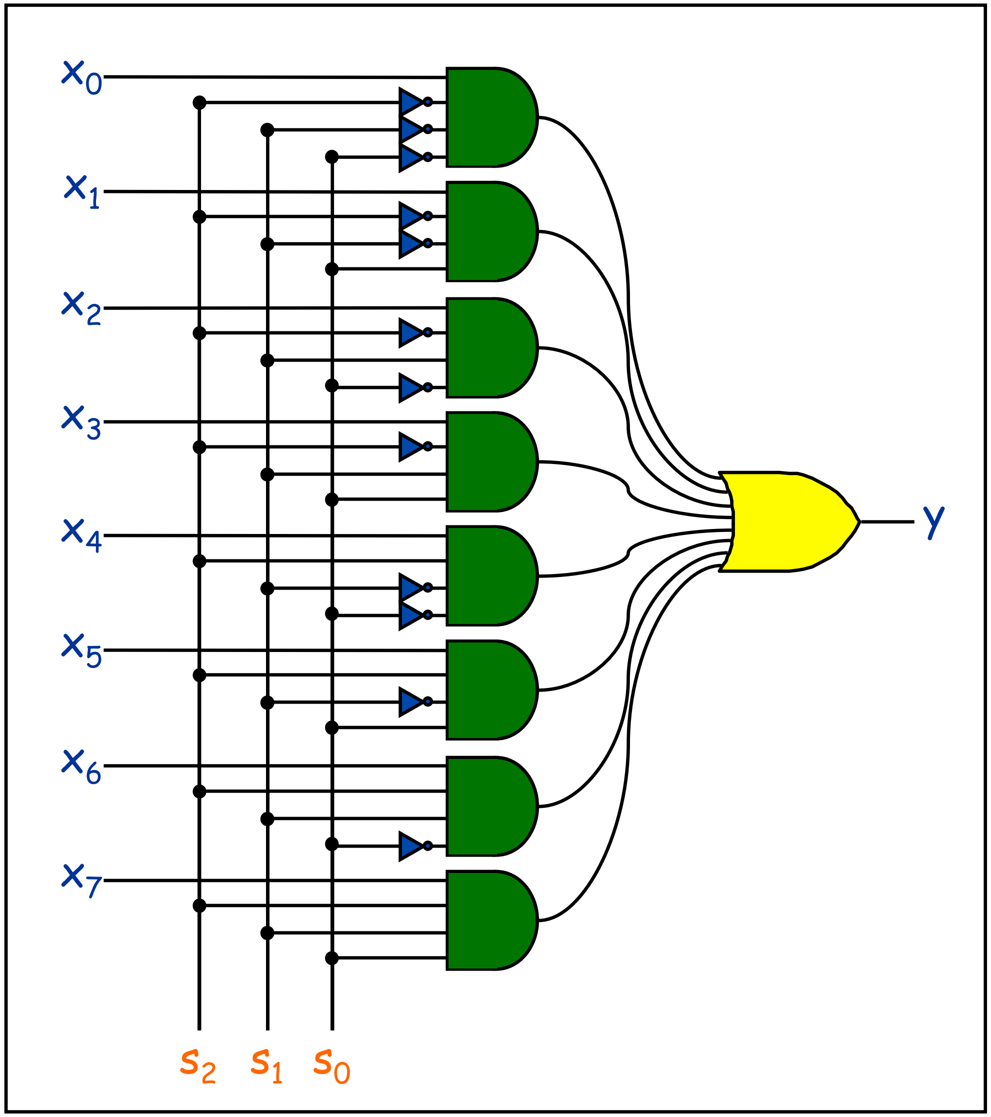
n = 8 for main memory

2^n-to-1 multiplexer.

- n select inputs, 2^n data inputs, 1 output.
- Copies "selected" data input bit to output.



8-to-1 Mux Interface



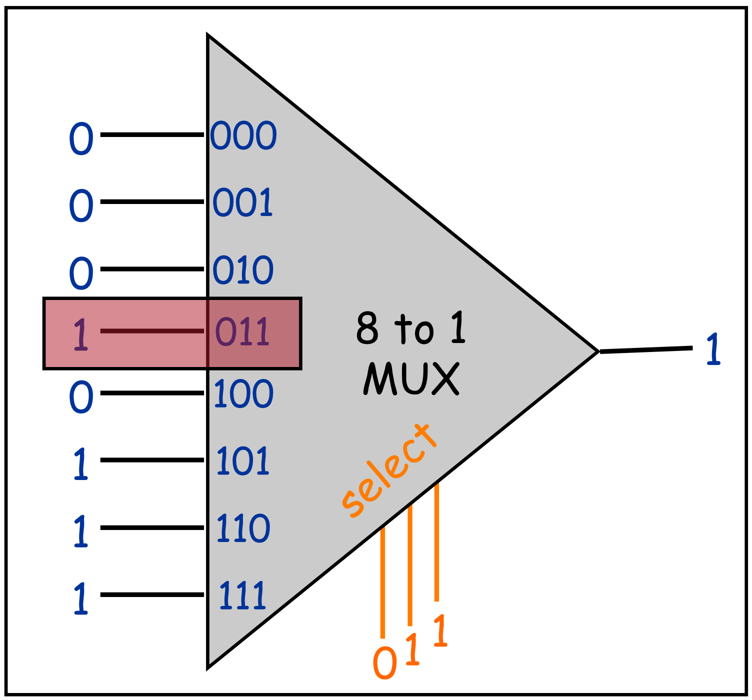
8-to-1 Mux Implementation

2^n-to-1 Multiplexer

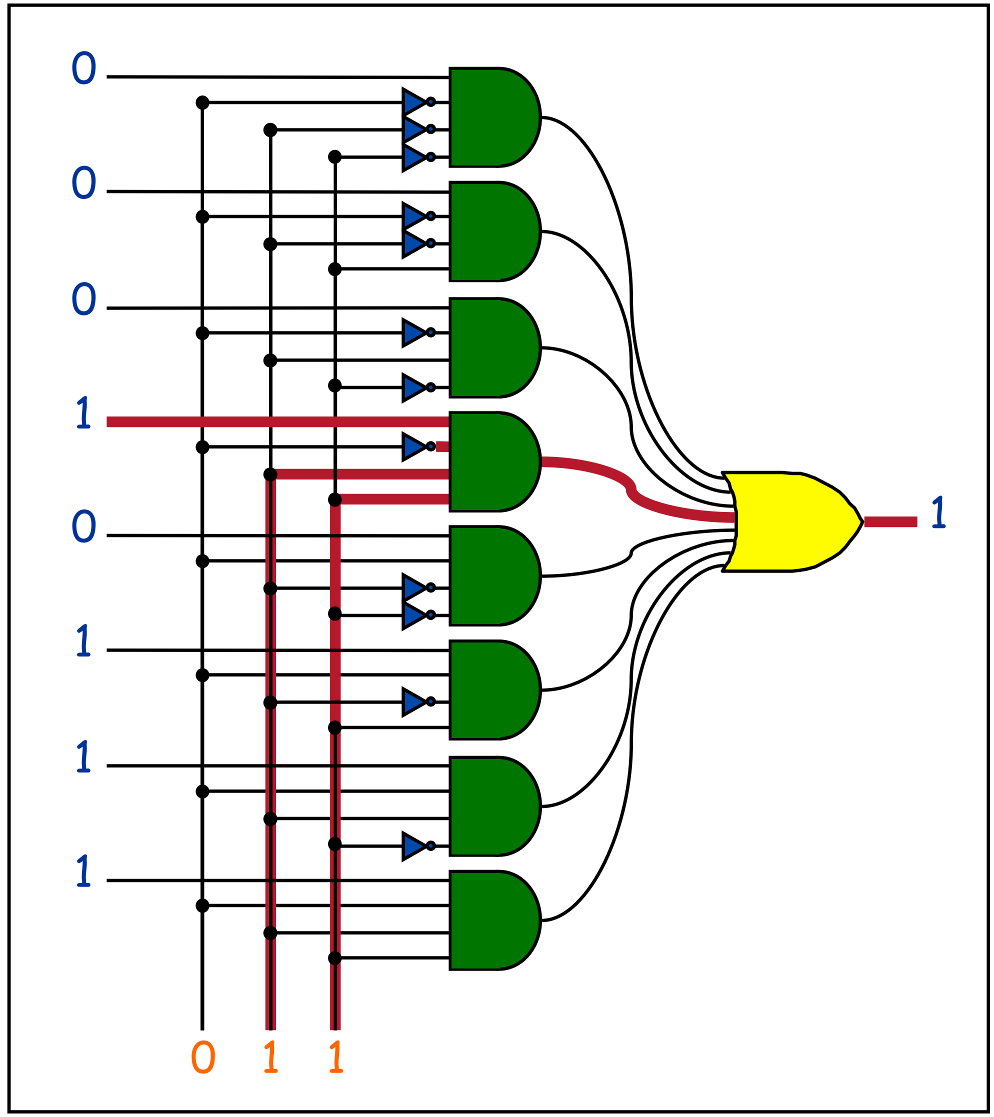
n = 8 for main memory

2^n-to-1 multiplexer.

- n select inputs, 2^n data inputs, 1 output.
- Copies "selected" data input bit to output.



8-to-1 Mux Interface



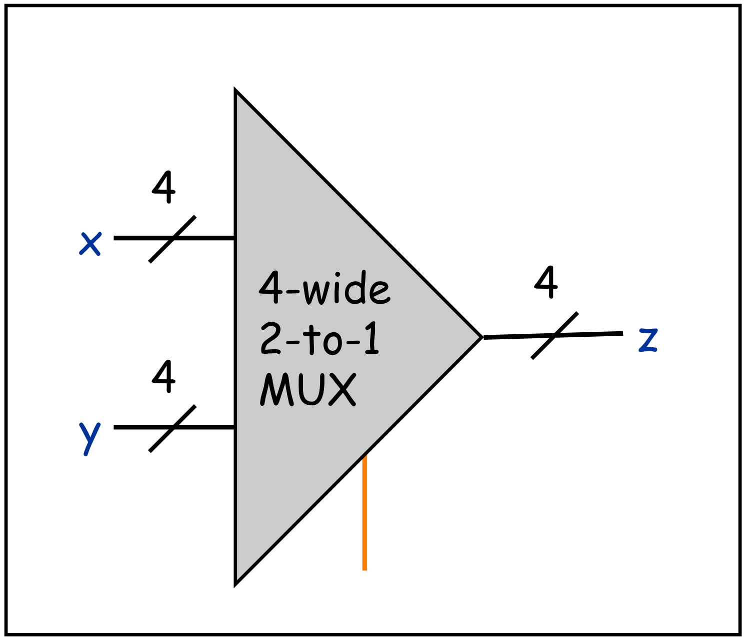
8-to-1 Mux Implementation

2^n-to-1 Multiplexer, Width = k

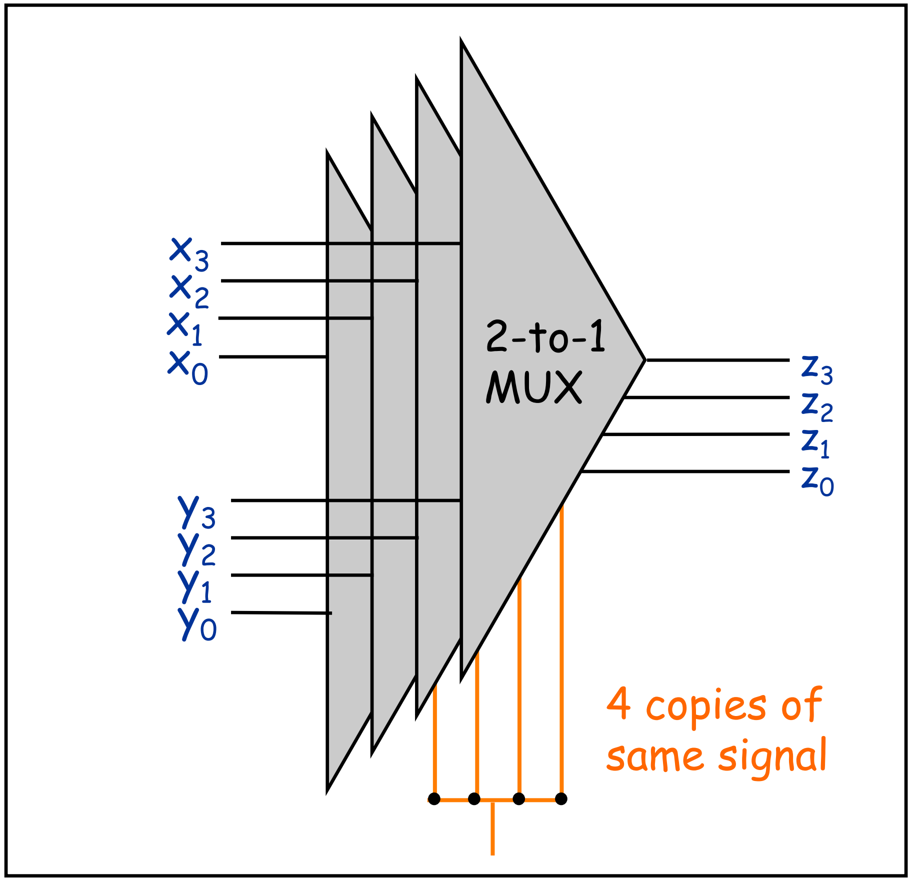
n = 8, k = 16 for main memory

2^n-to-1 multiplexer, width = k.

- Select from one of 2^n k-bit buses.
- Copies k "selected" data bits to output.
- Layering k 2^n-to-1 multiplexers.



Interface for 2-to-1 MUX, width = 4

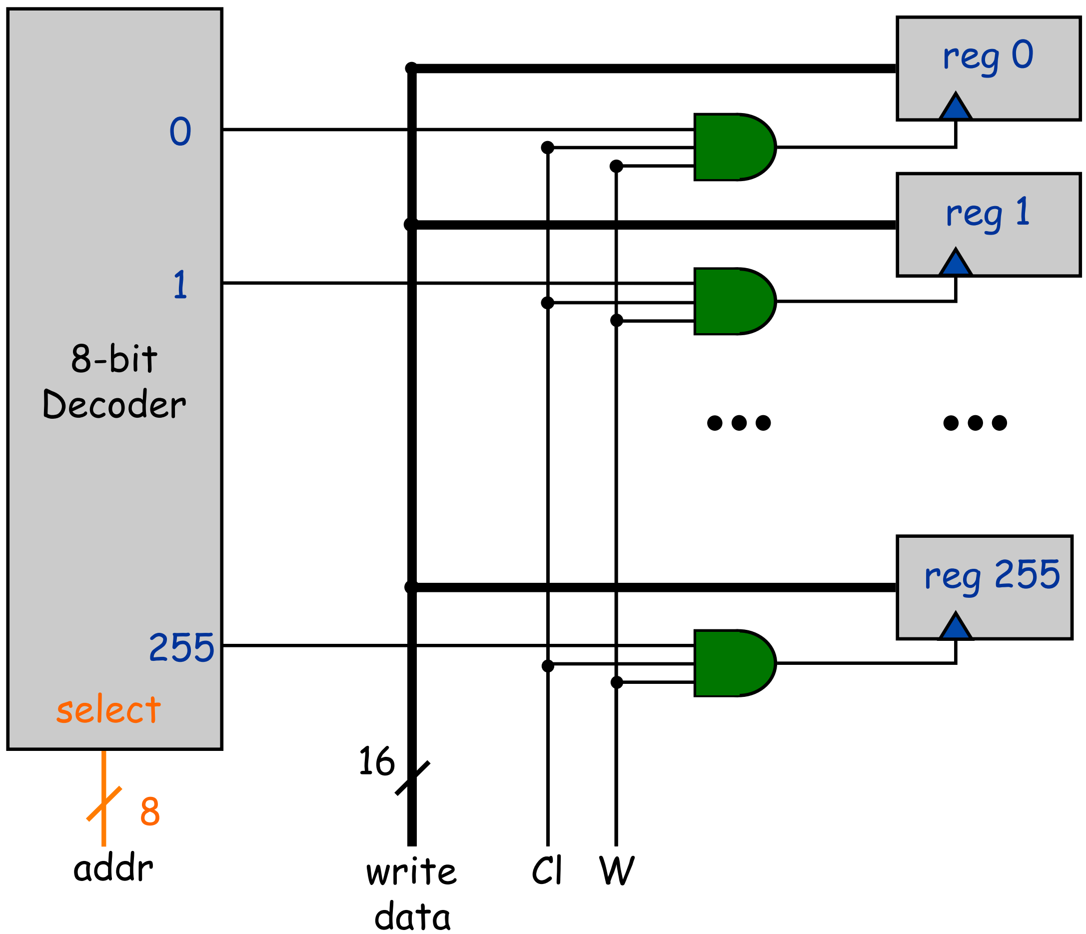


Implementation for 2-to-1 MUX, width = 4

Register File Implementation: Writing

Implementation example: TOY main memory.

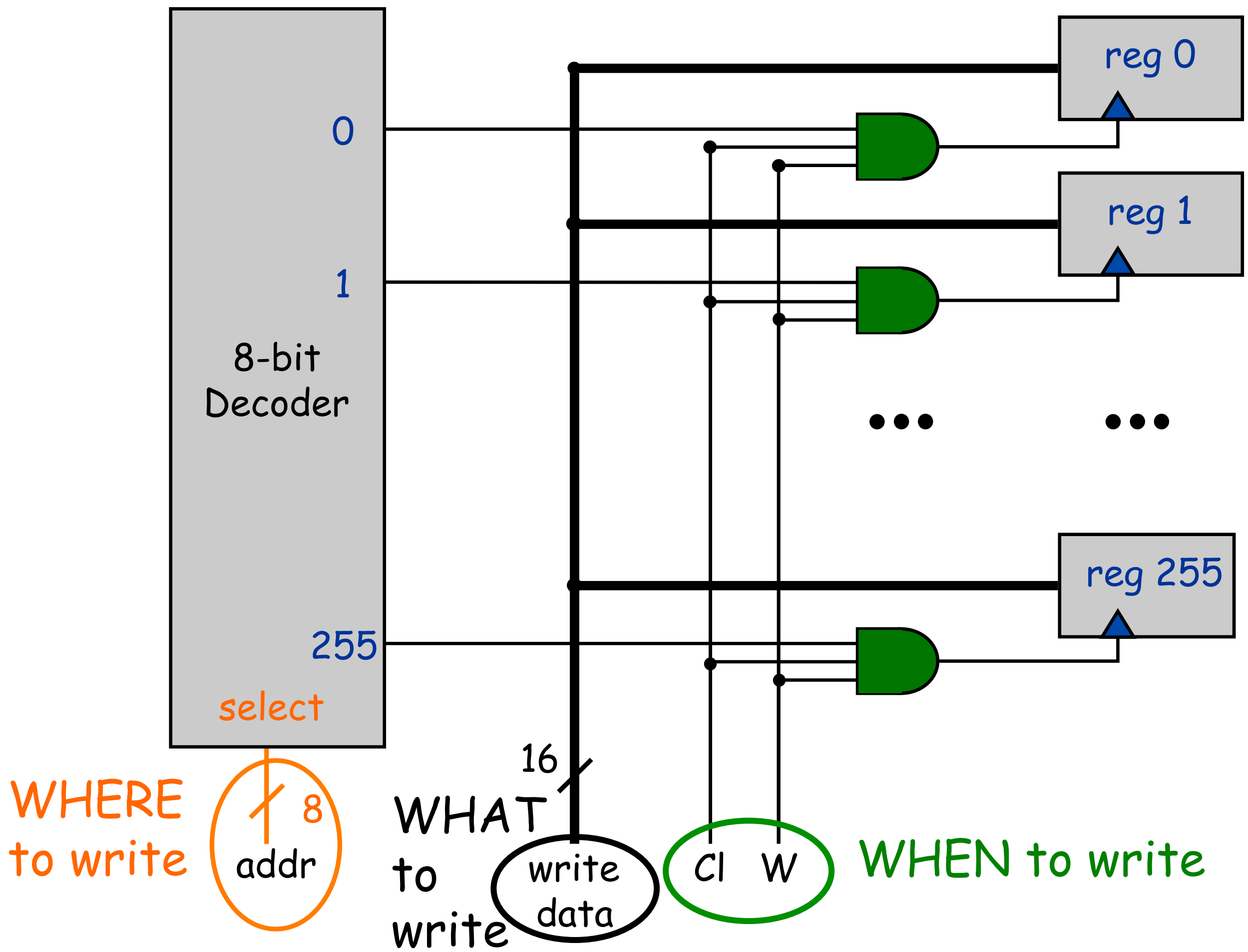
- Use 256 16-bit registers.
- Decoder is combinational circuit.



Register File Implementation: Writing

Implementation example: TOY main memory.

- Use 256 16-bit registers.
- Decoder is combinational circuit.

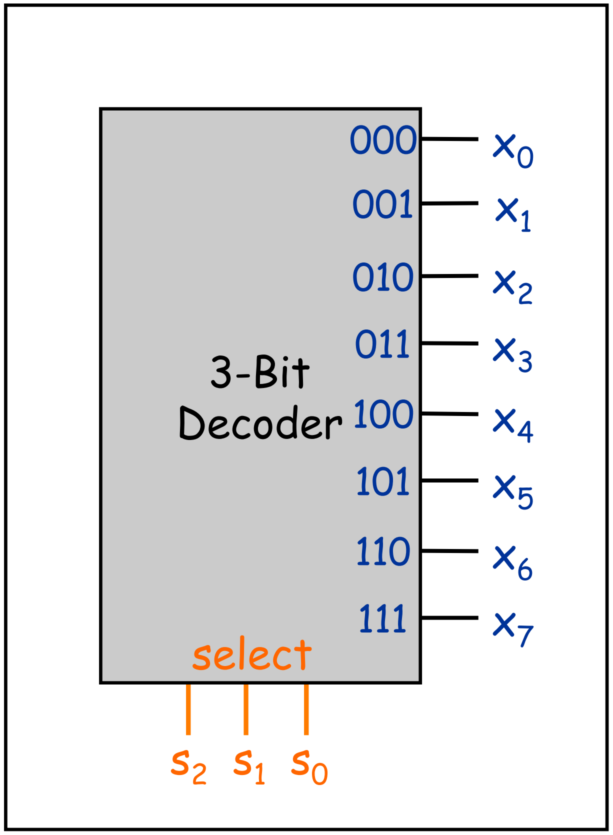


n-Bit Decoder

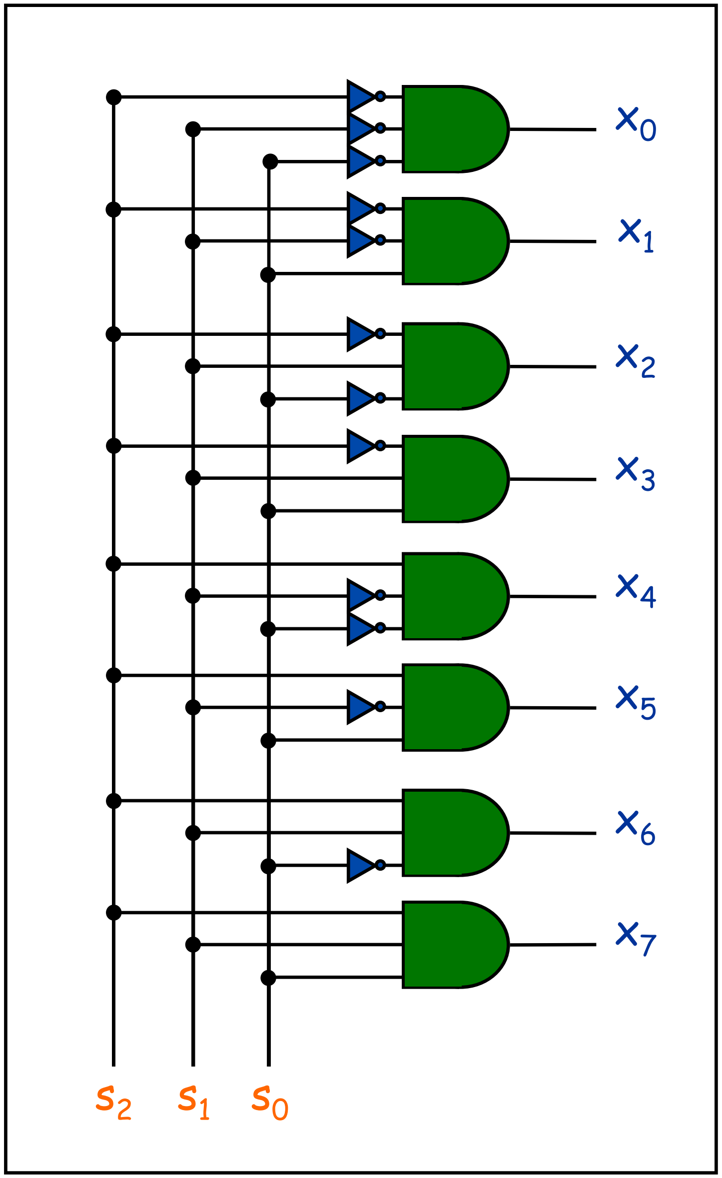
n = 8 for main memory

n-bit decoder.

- n address inputs, 2^n data outputs.
- Addressed output bit is 1; others are 0.



3-Bit Decoder Interface



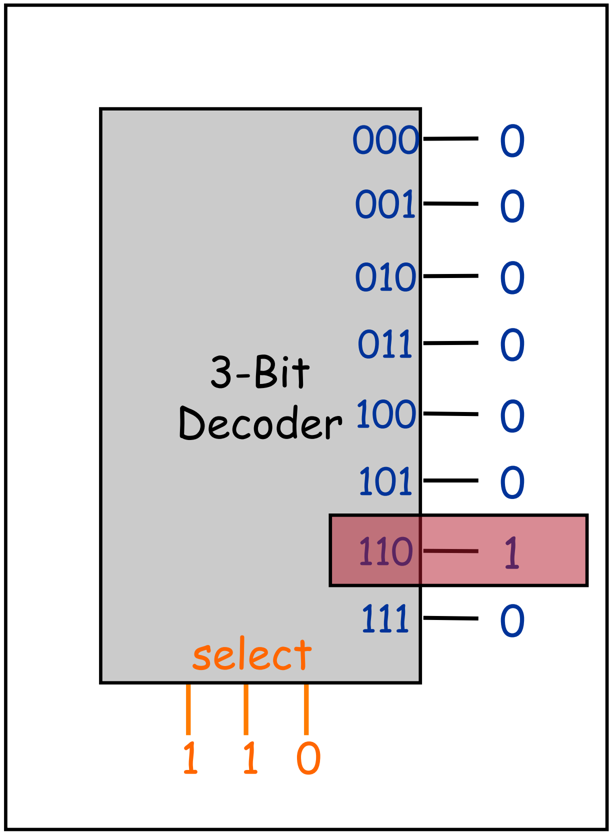
3-Bit Decoder Implementation

n-Bit Decoder

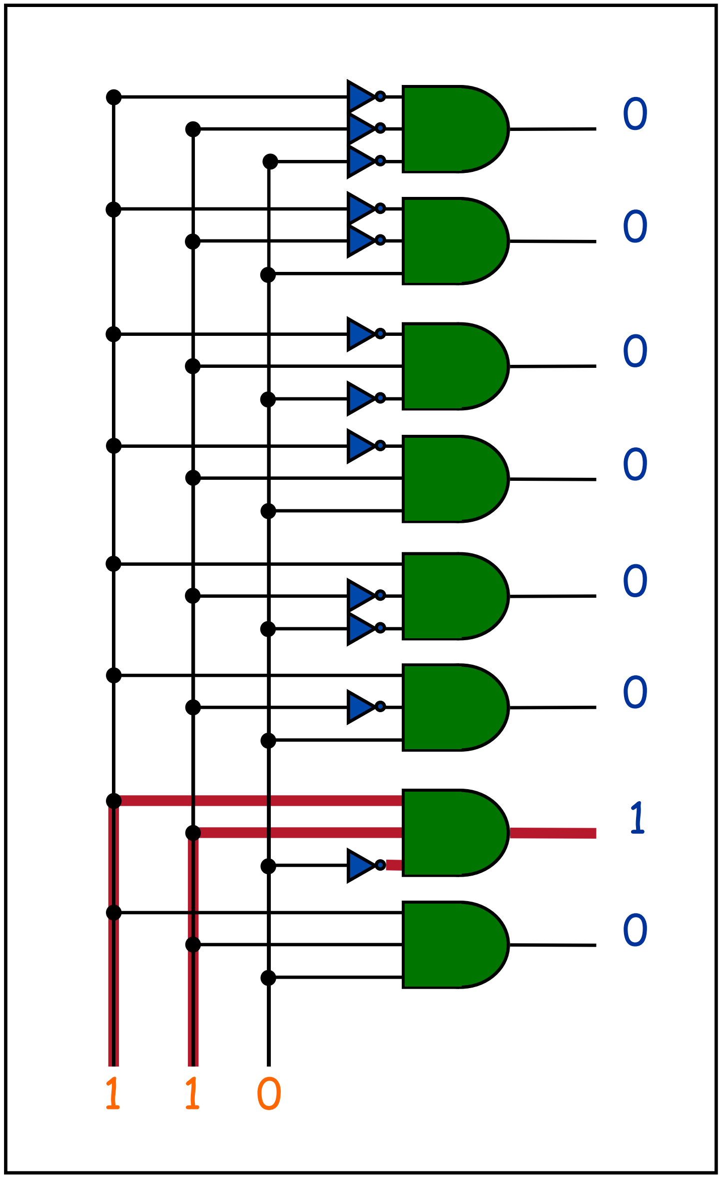
n = 8 for main memory

n-bit decoder.

- n address inputs, 2^n data outputs.
- Addressed output bit is 1; others are 0.



3-Bit Decoder Interface

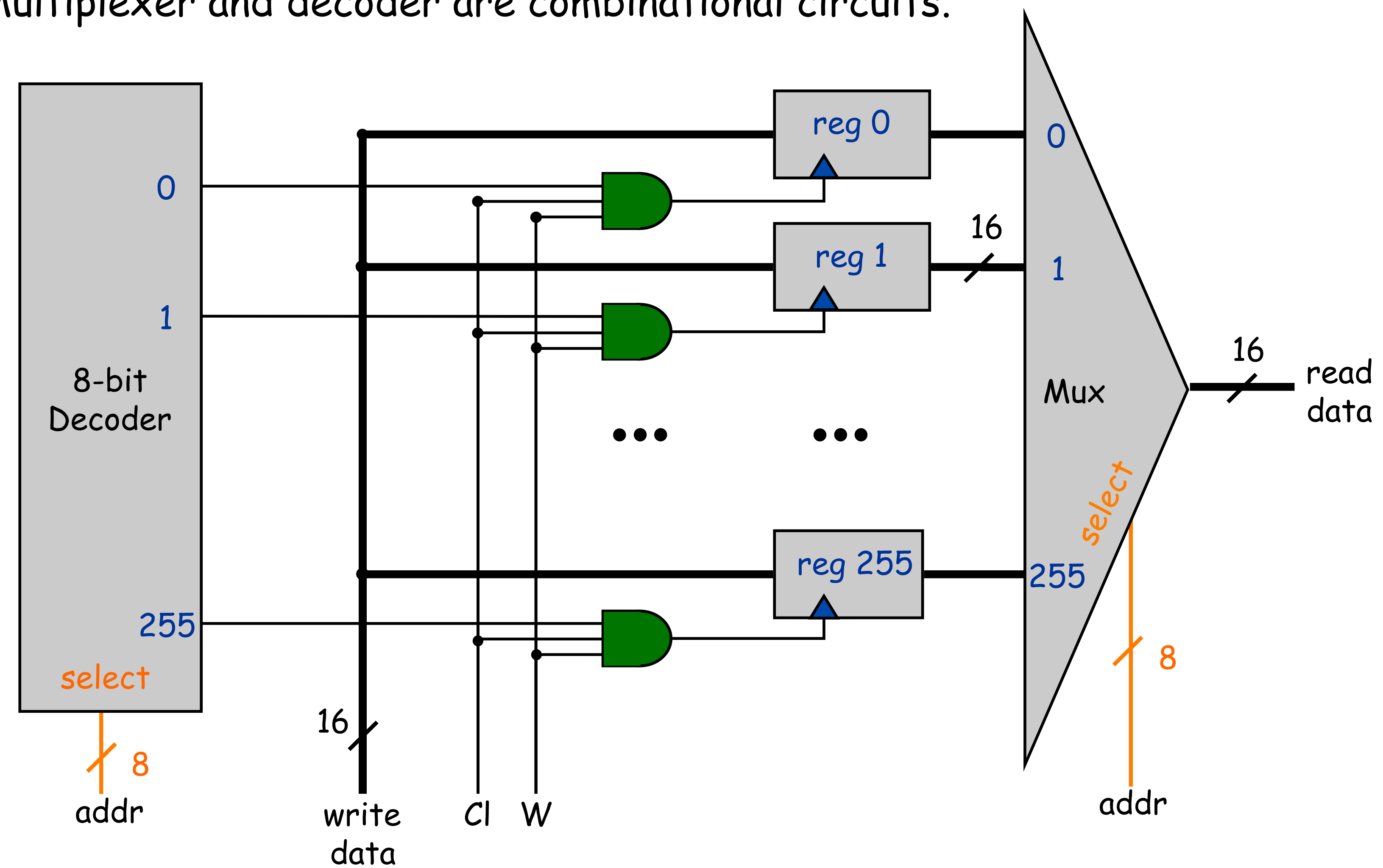


3-Bit Decoder Implementation

Register File Implementation: Reading and Writing

Implementation example: TOY main memory.

- Use 256 16-bit registers.
- Multiplexer and decoder are combinational circuits.



Register File Variations

Read address can be different from Write address

- Not in Main Memory (one address from instruction or PC)
- But definitely in TOY registers (read from and write to different registers)

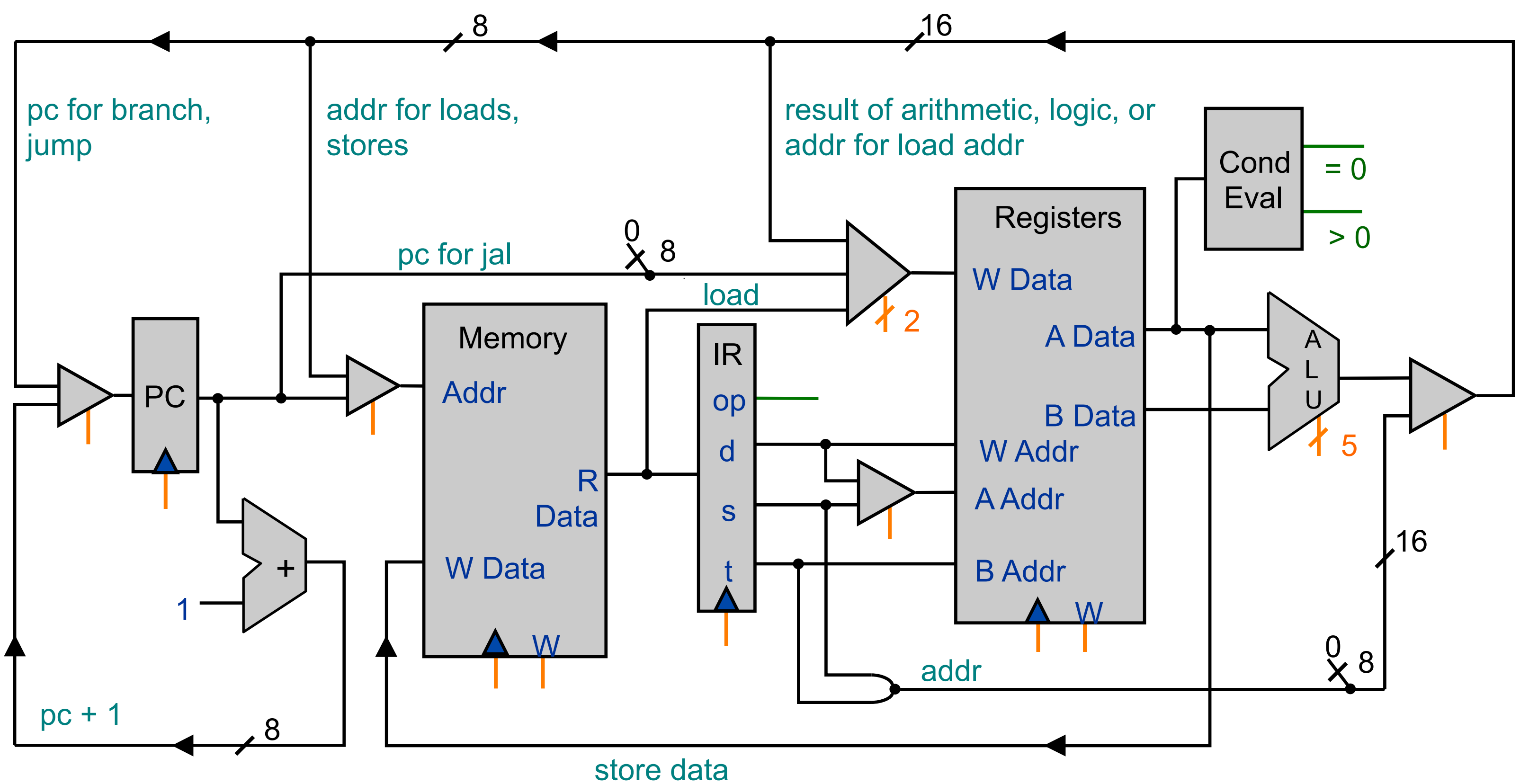
Can have multiple "ports"

- TOY registers supply TWO values per instruction
- How? Just get another set of 16-to-1, 16-wide multiplexors (and one more 4-bit address)

Actual technologies for register and memory are different.

- Register files are relatively small and very fast (expensive per bit)
- Memories are relatively large and pretty fast (very cheap per bit)
- Drastic evolution of technology over time (Moore's Law)

6.3: TOY Machine Architecture



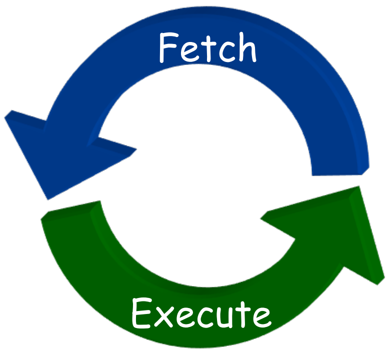
The TOY Machine

TOY machine.

- 256 16-bit words of memory.
- 16 16-bit registers.
- 1 8-bit program counter.
- 16 instructions types.

What we've done.

- Written programs for the TOY machine.
- Software implementation of fetch-execute cycle.
  - TOY simulator.



Our goal today.

- Hardware implementation of fetch-execute cycle.
  - TOY computer.

Designing a Processor

How to build a microprocessor?

- Develop instruction set architecture (ISA).
  - 16-bit words, 16 TOY machine instructions
- Determine major components.
  - ALU, memory, registers, program counter
- Determine datapath requirements.
  - "flow" of bits
- Establish clocking methodology.
  - 2-cycle design: fetch, execute
- Analyze how to implement each instruction.
  - determine settings of control signals

Instruction Set Architecture

Instruction set architecture (ISA).

- 16-bit words, 256 words of memory, 16 registers.
- Determine set of primitive instructions.
  - too narrow ⇒ cumbersome to program
  - too broad ⇒ cumbersome to build hardware
- TOY machine: 16 instructions.

Instructions	
0:	halt
1:	add
2:	subtract
3:	and
4:	xor
5:	shift left
6:	shift right
7:	load address

Instructions	
8:	load
9:	store
A:	load indirect
B:	store indirect
C:	branch zero
D:	branch positive
E:	jump register
F:	jump and link

Designing a Processor

How to build a microprocessor?

- Develop instruction set architecture (ISA).
  - 16-bit words, 16 TOY machine instructions
- Determine major components.
  - ALU, memory, registers, program counter
- Determine datapath requirements.
  - "flow" of bits
- Establish clocking methodology.
  - 2-cycle design: fetch, execute
- Analyze how to implement each instruction.
  - determine settings of control signals

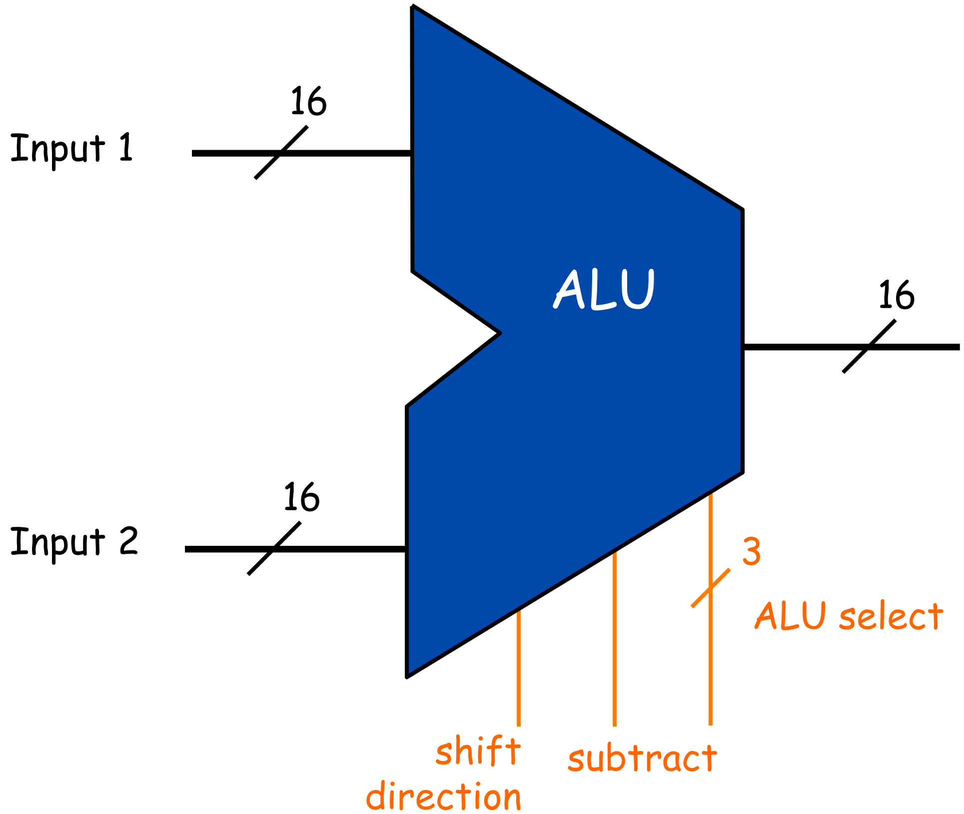
Arithmetic Logic Unit

TOY ALU.

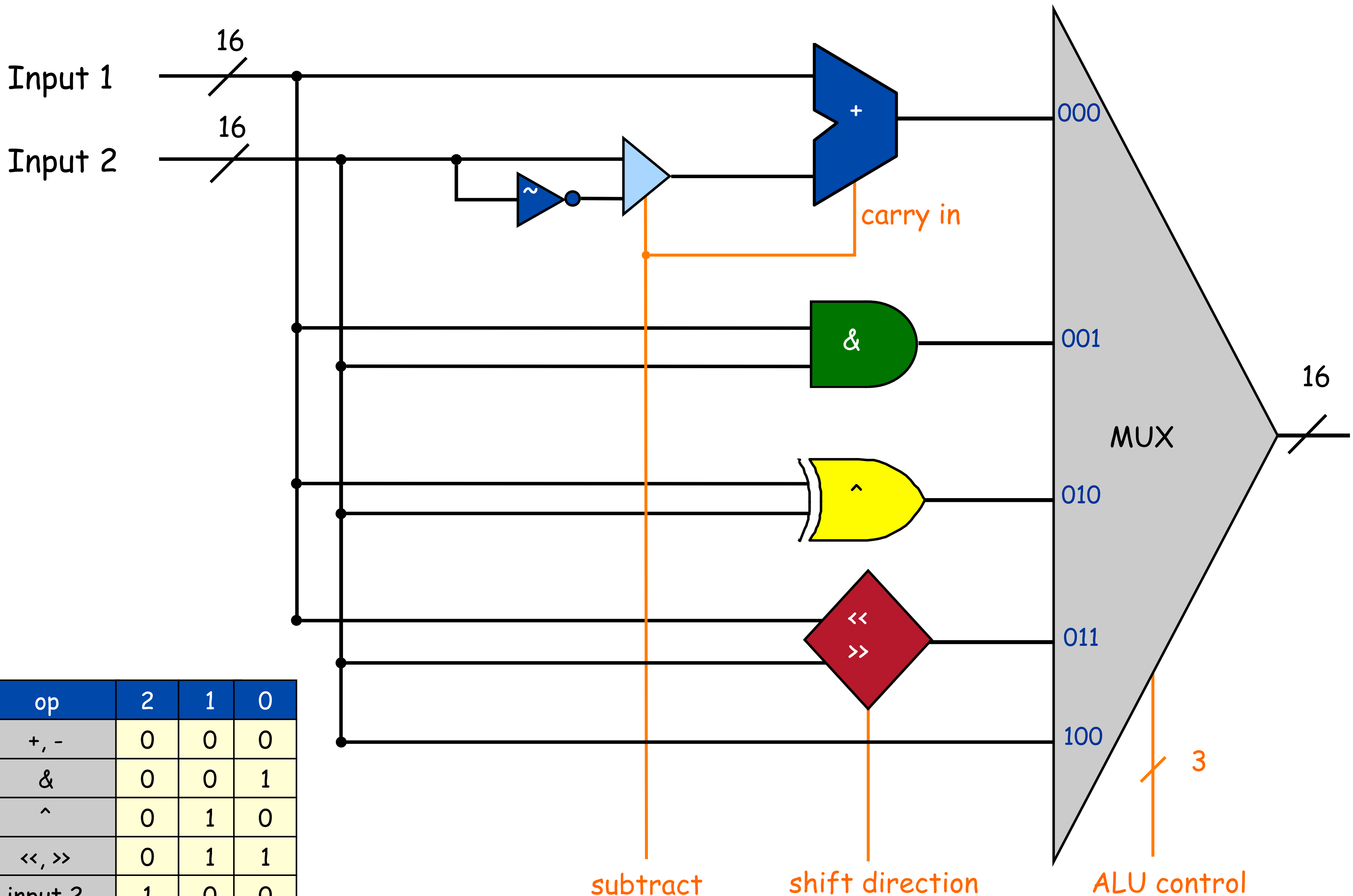
- Big combinational circuit.
- 16-bit buses for inputs and output.
- Add, subtract, and, xor, shift left, shift right, copy input 2.

technical hack

op	2	1	0
+, -	0	0	0
&	0	0	1
^	0	1	0
<<, >>	0	1	1
input 2	1	0	0



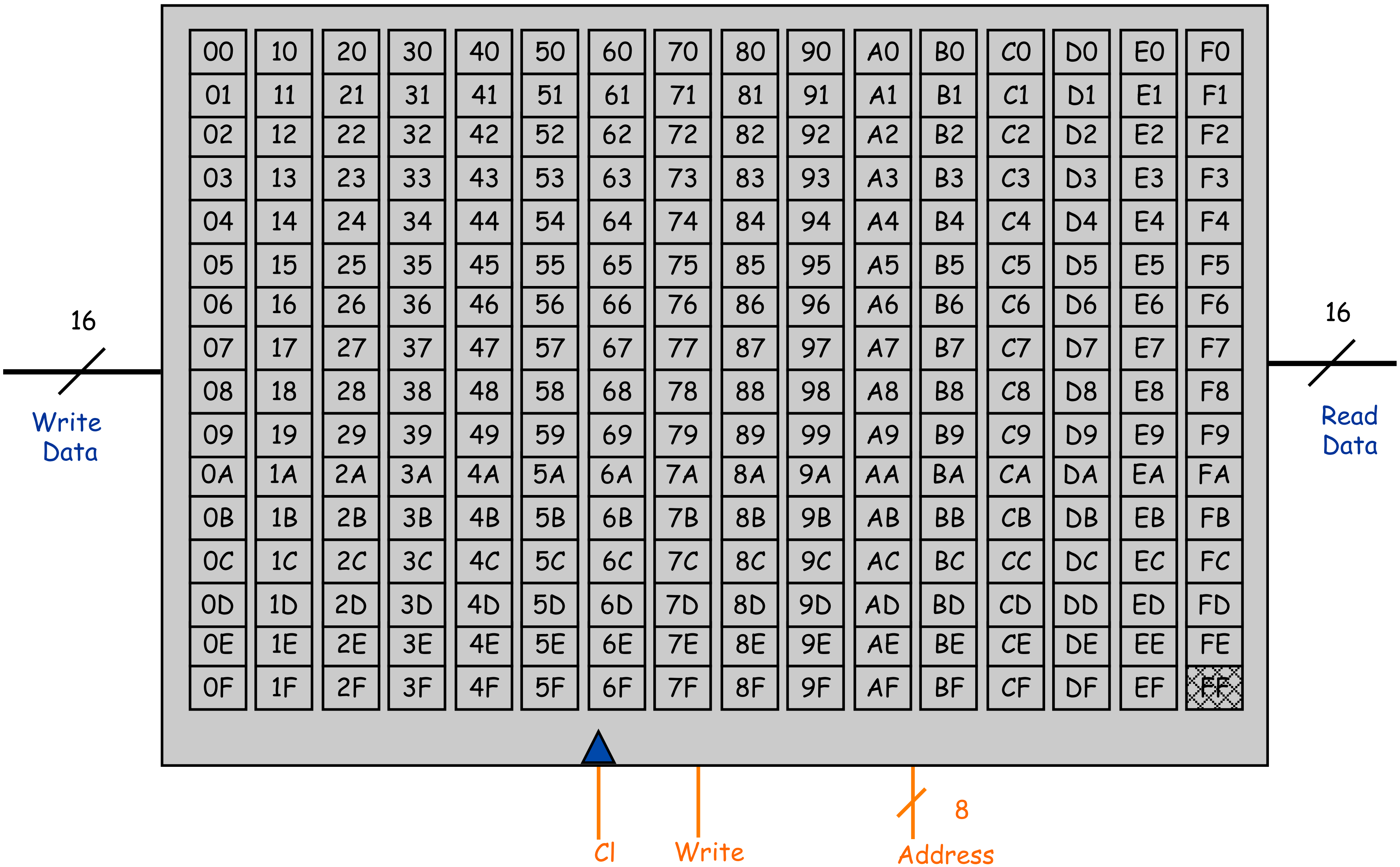
Arithmetic Logic Unit: Implementation



op	2	1	0
+, -	0	0	0
&	0	0	1
^	0	1	0
<<, >>	0	1	1
input 2	1	0	0

Main Memory

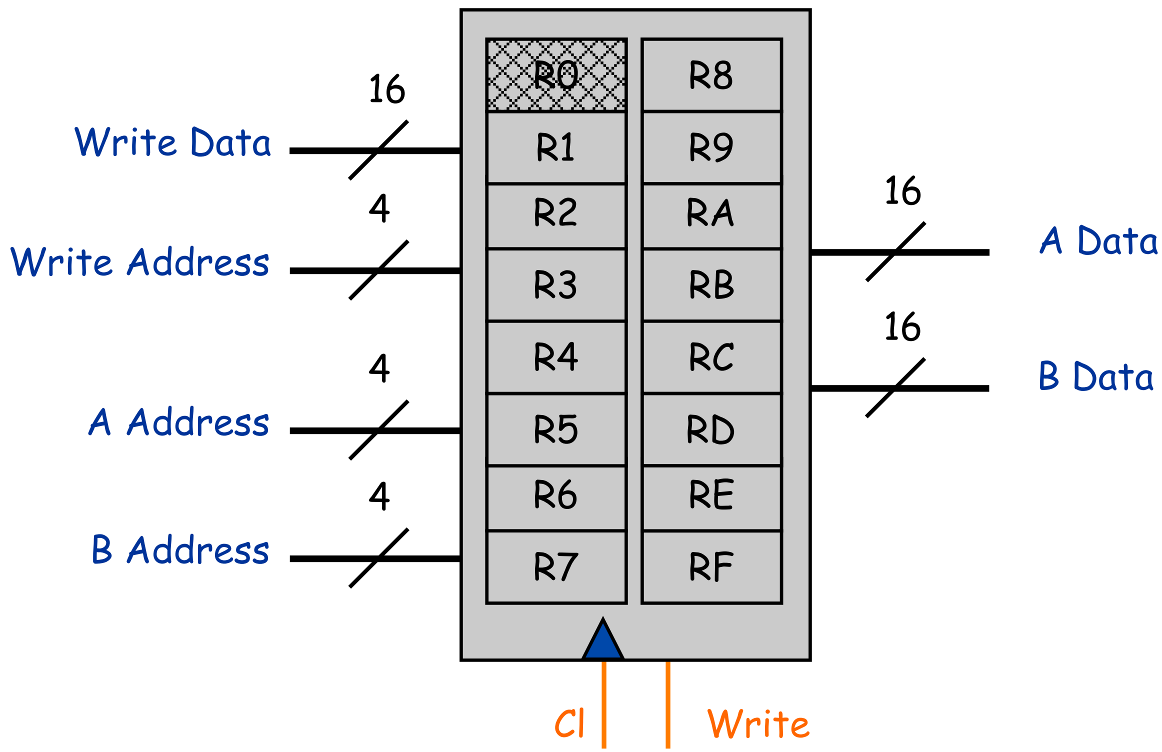
TOY main memory: 256 x 16-bit register file.



Registers

TOY registers: fancy 16 x 16-bit register file.

- Want to be able to read two registers, and write to a third in the same instructions: `R1 ← R2 + R3`.
- 3 address inputs, 1 data input, 2 data outputs.
- Add decoders and muxes for additional ports.



Designing a Processor

How to build a microprocessor?

- Develop instruction set architecture (ISA).
  - 16-bit words, 16 TOY machine instructions
- Determine major components.
  - ALU, memory, registers, program counter
- Determine datapath requirements.
  - "flow" of bits
- Establish clocking methodology.
  - 2-cycle design: fetch, execute
- Analyze how to implement each instruction.
  - determine settings of control signals

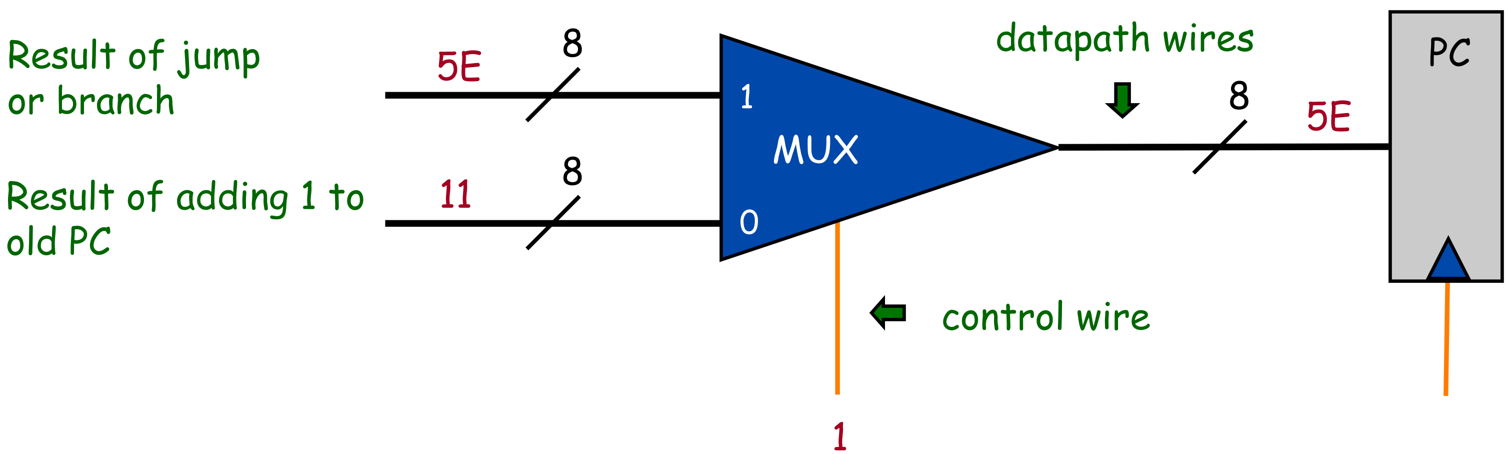
Datapath and Control

Datapath.

- Layout and interconnection of components.
- Must accommodate all instruction types.

Control.

- Choreographs the "flow" of information on the datapath.
- Depending on instruction, different control wires are turned on.



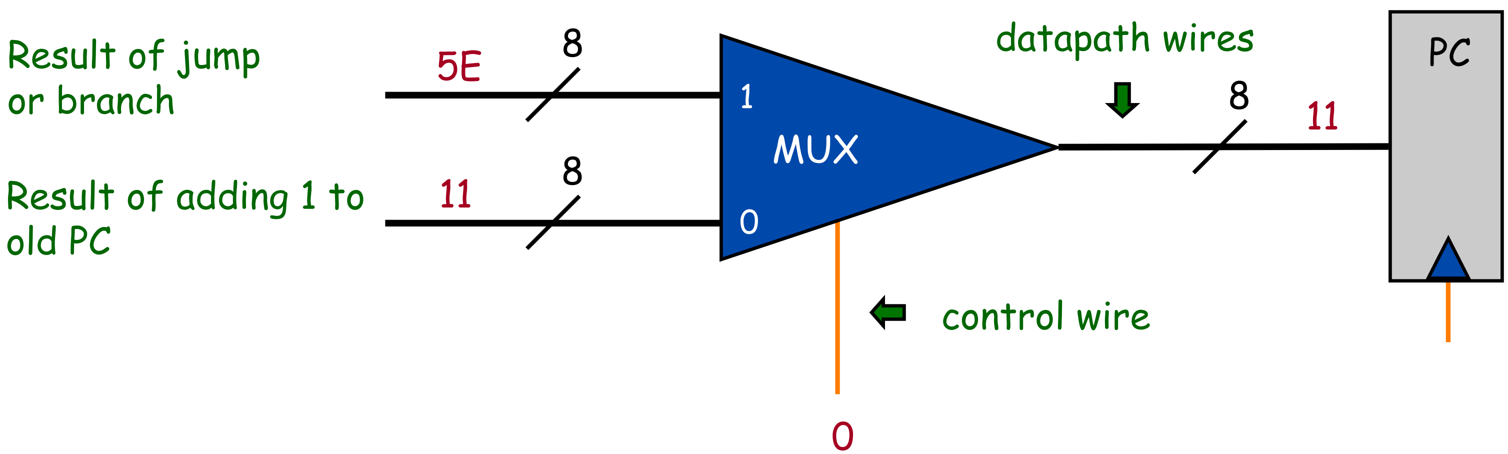
Datapath and Control

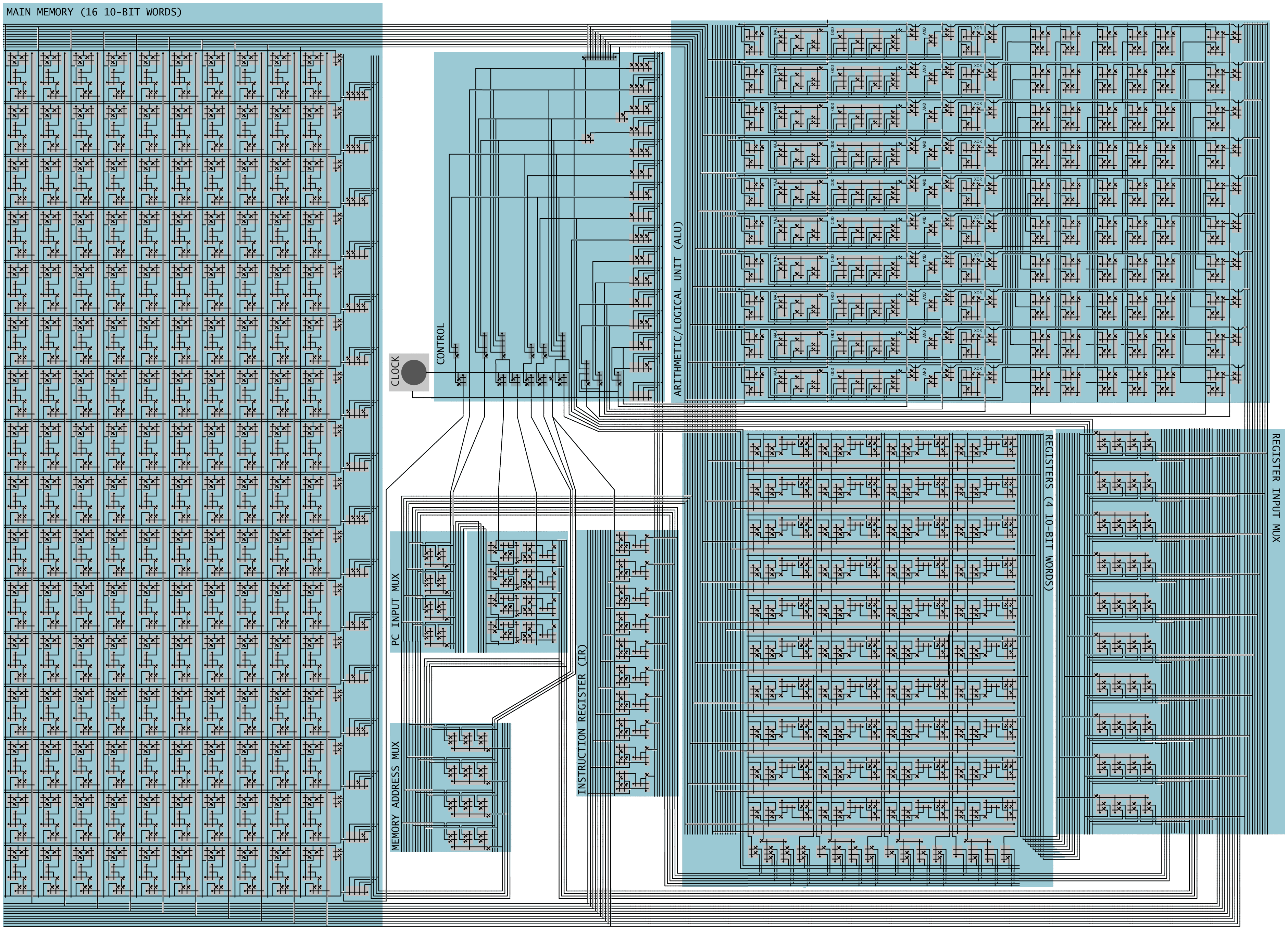
Datapath.

- Layout and interconnection of components.
- Must accommodate all instruction types.

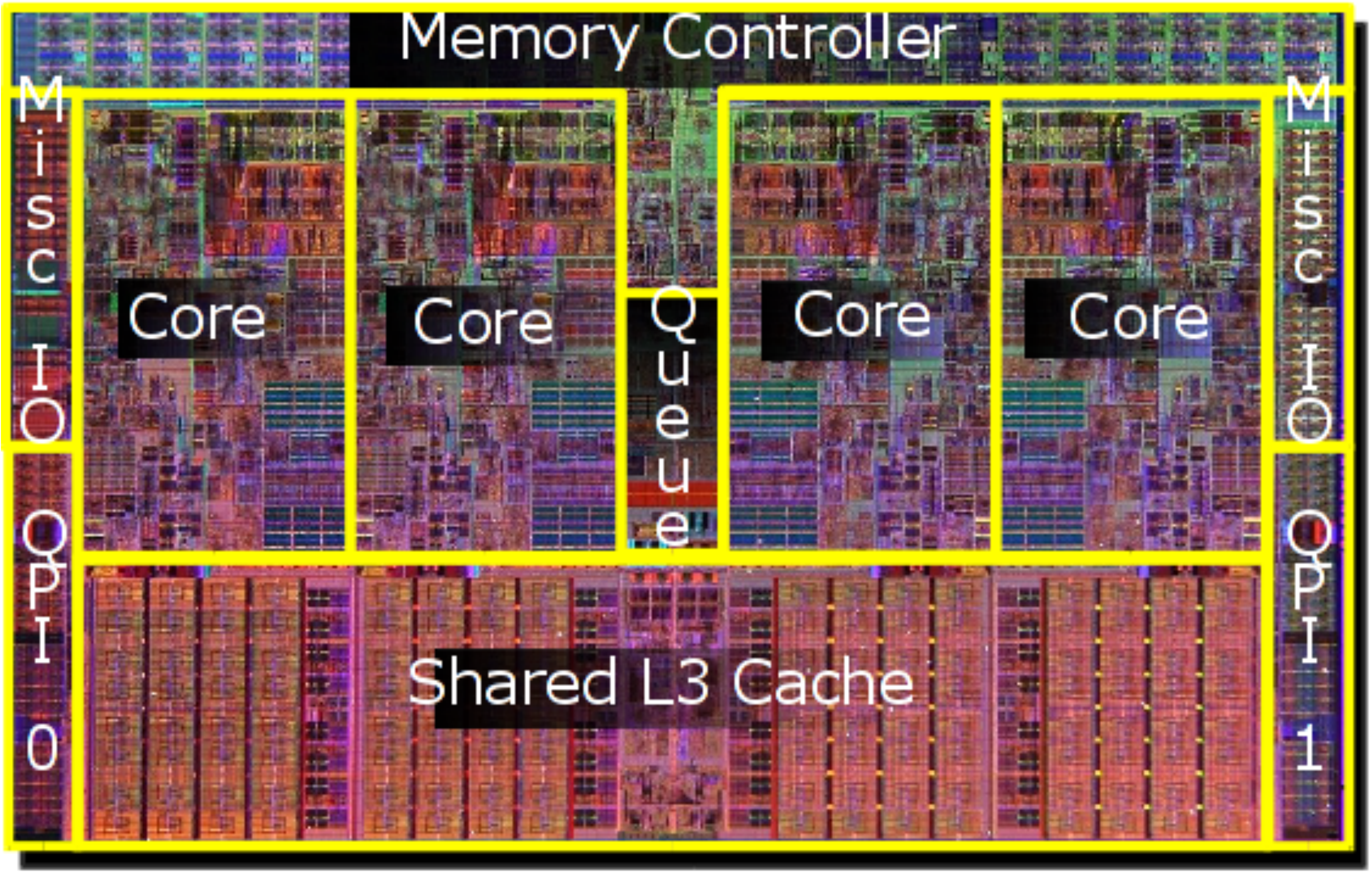
Control.

- Choreographs the "flow" of information on the datapath.
- Depending on instruction, different control wires are turned on.

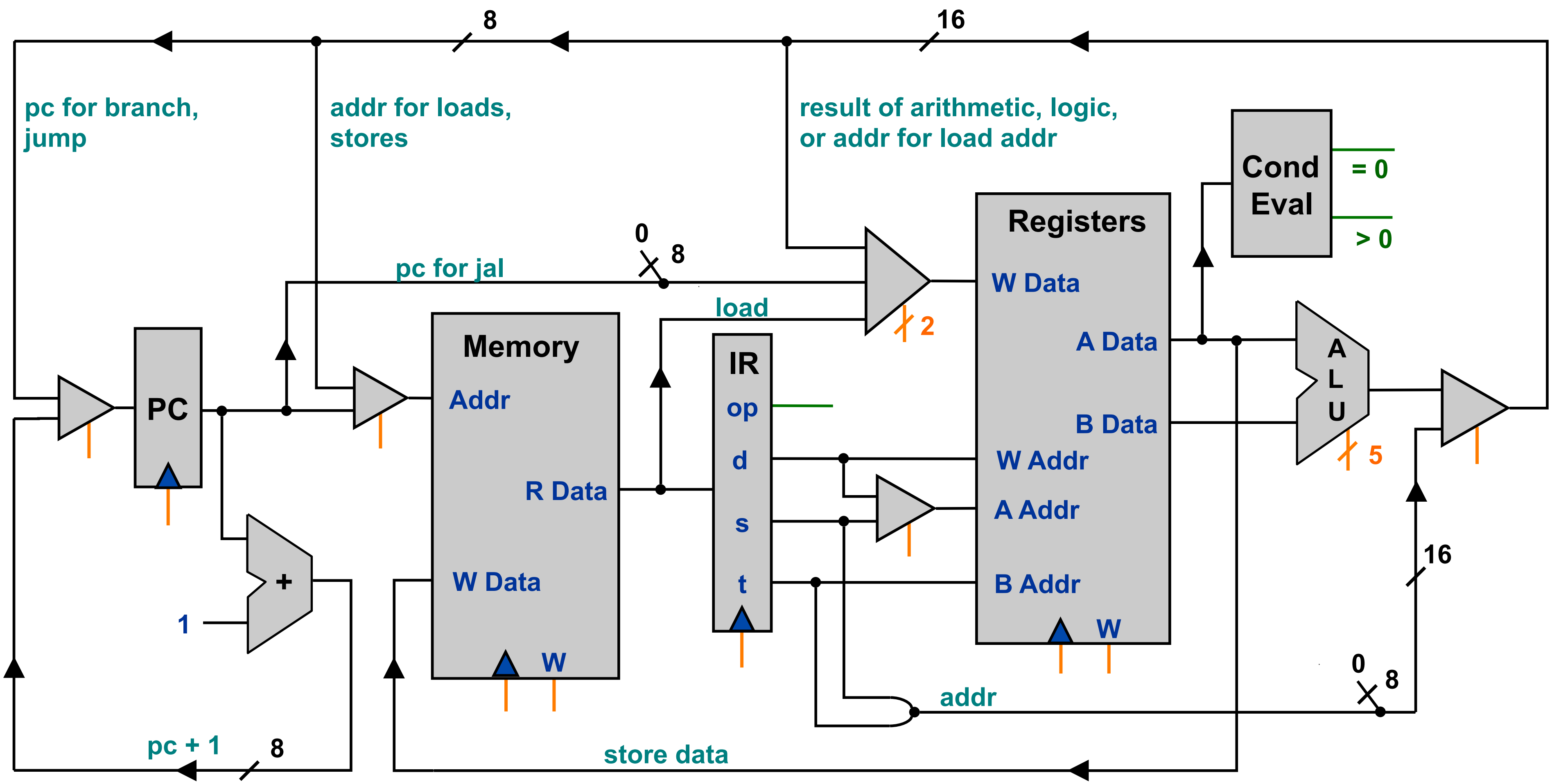



MAIN MEMORY (16 10-BIT WORDS)
CLOCK
CONTROL
ARITHMETIC/LOGICAL UNIT (ALU)
REGISTERS (4 10-BIT WORDS)
REGISTER INPUT MUX
PC INPUT MUX
INSTRUCTION REGISTER (IR)
MEMORY ADDRESS MUX

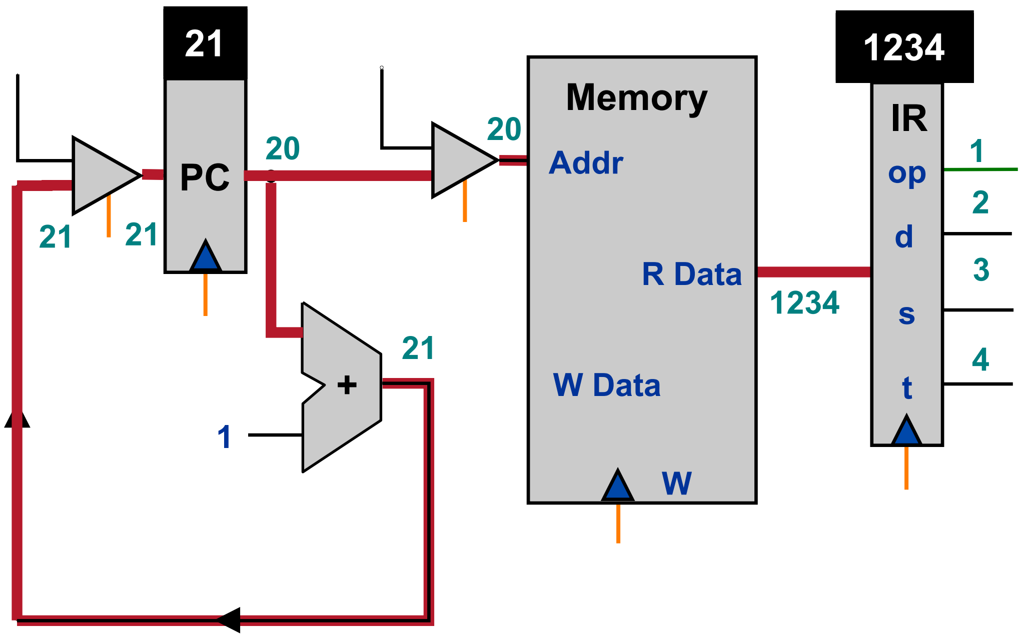
Real Microprocessor Chip (Intel Nehalem)



The TOY Datapath



The TOY Datapath: Add



Before fetch:

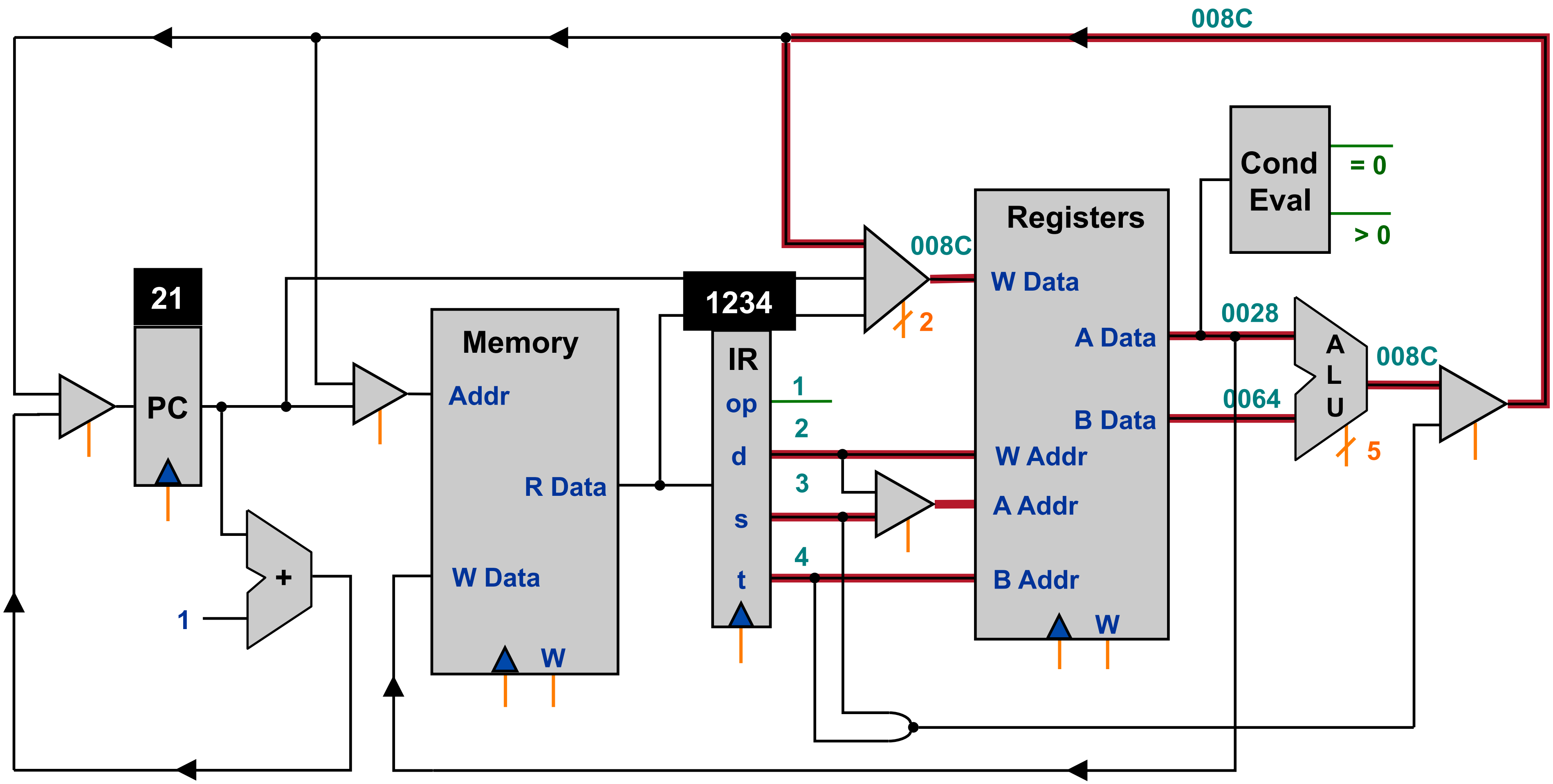
`pc = 20, mem[20] = 1234`

After fetch:

`pc = 21`

`IR = 1234: R[2] ← R[3] + R[4]`

The TOY Datapath: Add



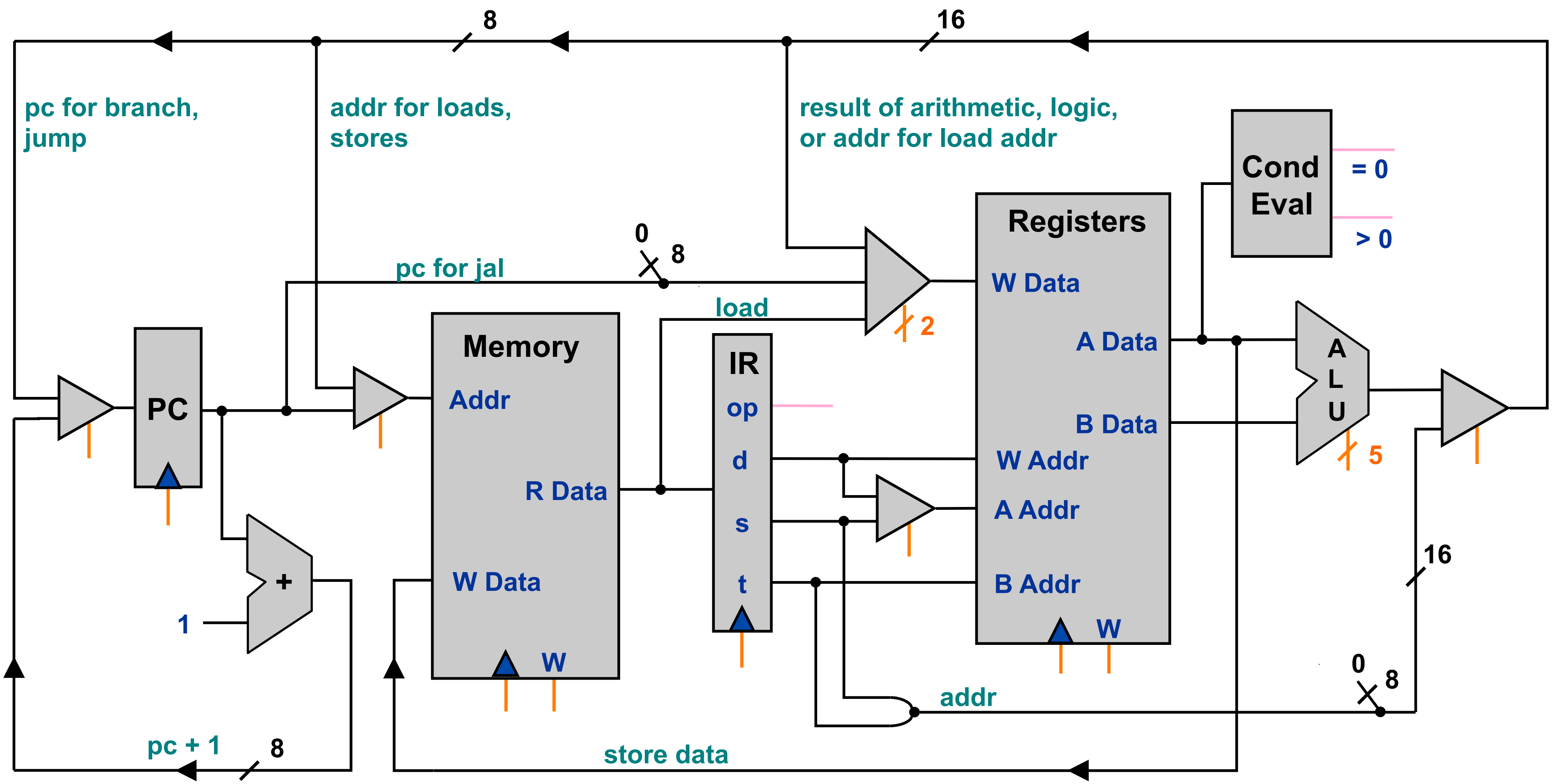
<u>Before execute:</u>

`pc = 21`
`IR = 1234: R[2] ← R[3] + R[4]`
`R[3] = 0028, R[4] = 0064`

<u>After execute:</u>

`pc = 21`
`R[2] = 008C`

Do Try This At Home



Trace the flow of some other instructions through the datapath picture.

Designing a Processor

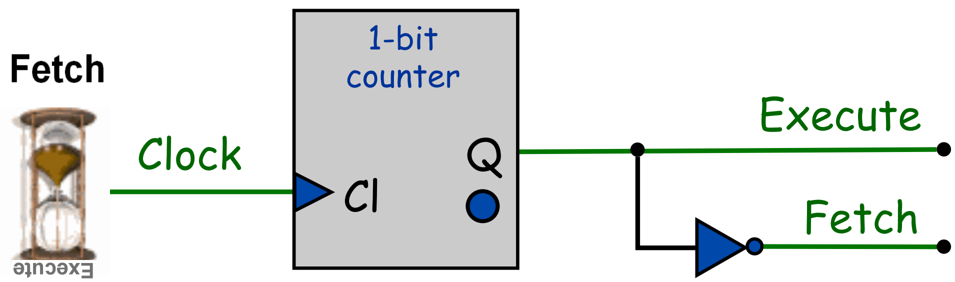
How to build a microprocessor?

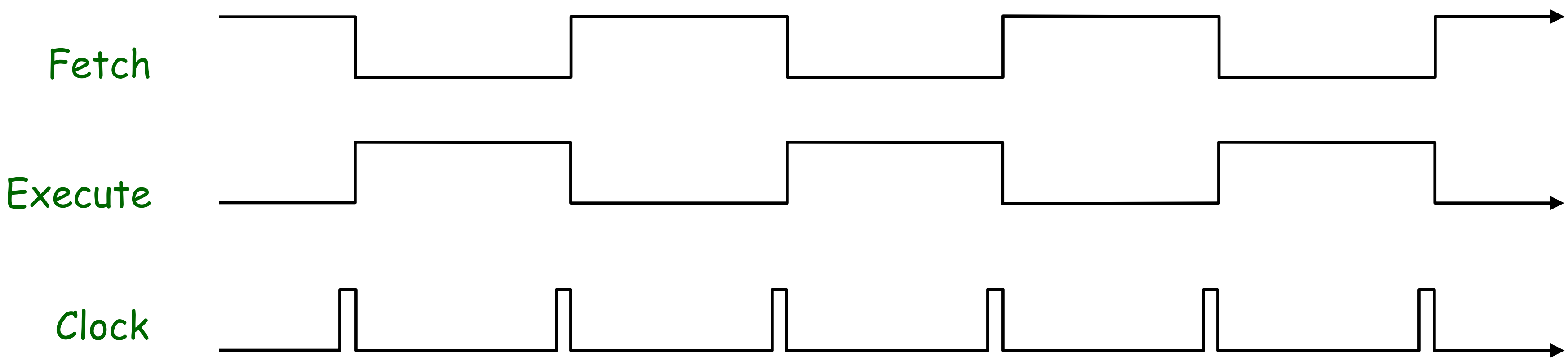
- Develop instruction set architecture (ISA).
  - 16-bit words, 16 TOY machine instructions
- Determine major components.
  - ALU, memory, registers, program counter
- Determine datapath requirements.
  - "flow" of bits
- Establish clocking methodology.
  - 2-cycle design: fetch, execute
- Analyze how to implement each instruction.
  - determine settings of control signals

Clocking Methodology

Two cycle design (fetch and execute).

- Use 1-bit counter to distinguish between 2 cycles.
- Use two cycles since fetch and execute phases each access memory and alter program counter.

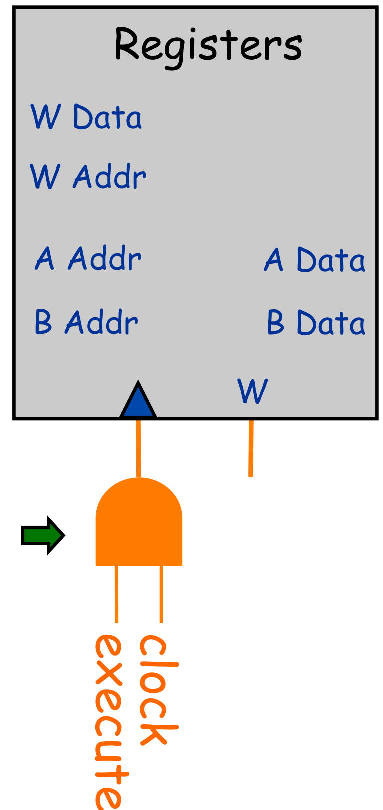




Clocking Methodology

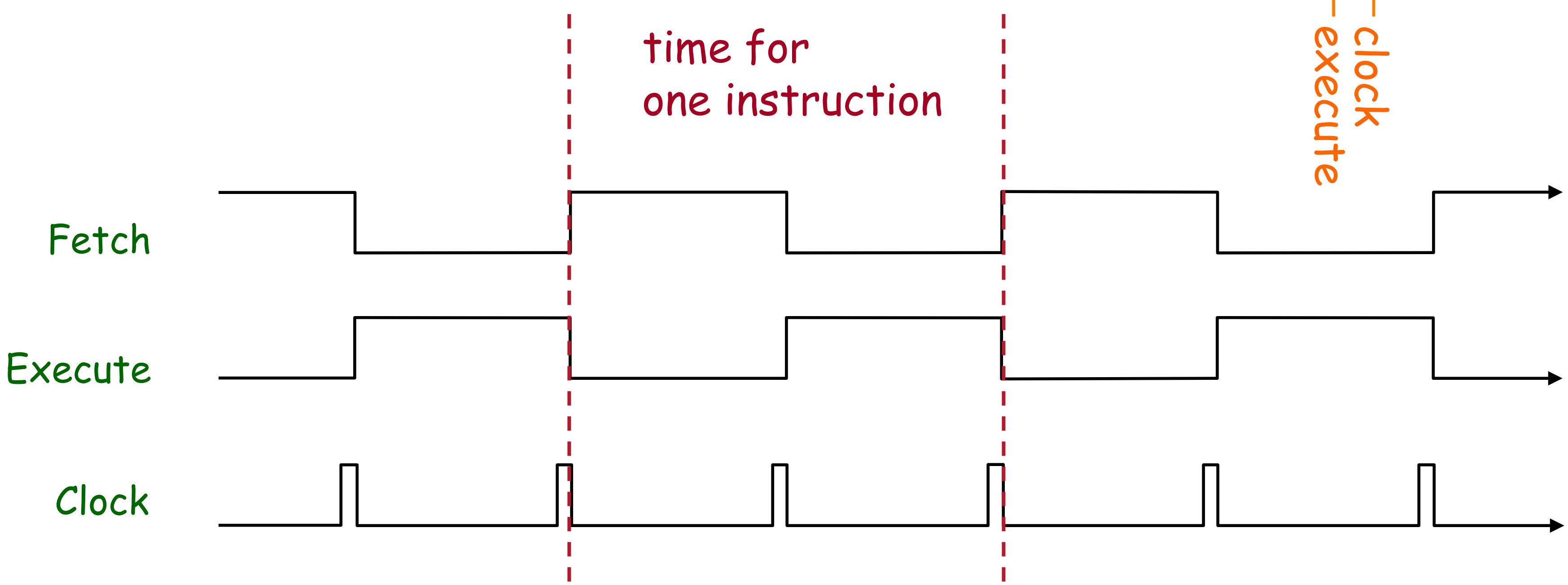
4 distinguishable epochs.

- During fetch phase.
- At very end of fetch phase.
- During execute phase.
- At very end of execute phase.



Ex: can only write at very end of execute phase.

- `R1 ← R1 + R1`



Designing a Processor

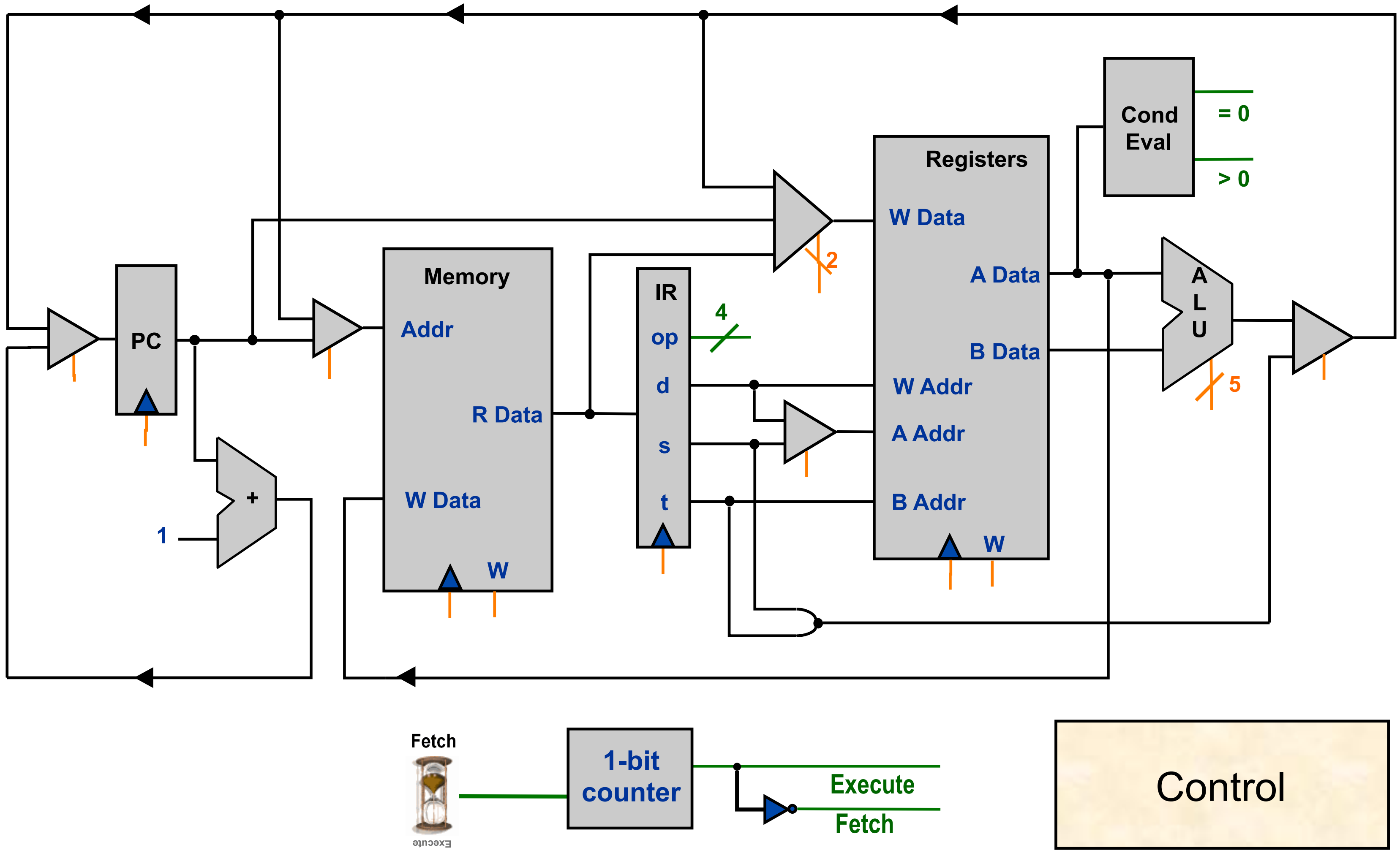
How to build a microprocessor?

- Develop instruction set architecture (ISA).
  - 16-bit words, 16 TOY machine instructions
- Determine major components.
  - ALU, memory, registers, program counter
- Determine datapath requirements.
  - "flow" of bits
- Establish clocking methodology.
  - 2-cycle design: fetch, execute
- Analyze how to implement each instruction.
  - determine settings of control signals

Control

Control: controls components, enables connections.

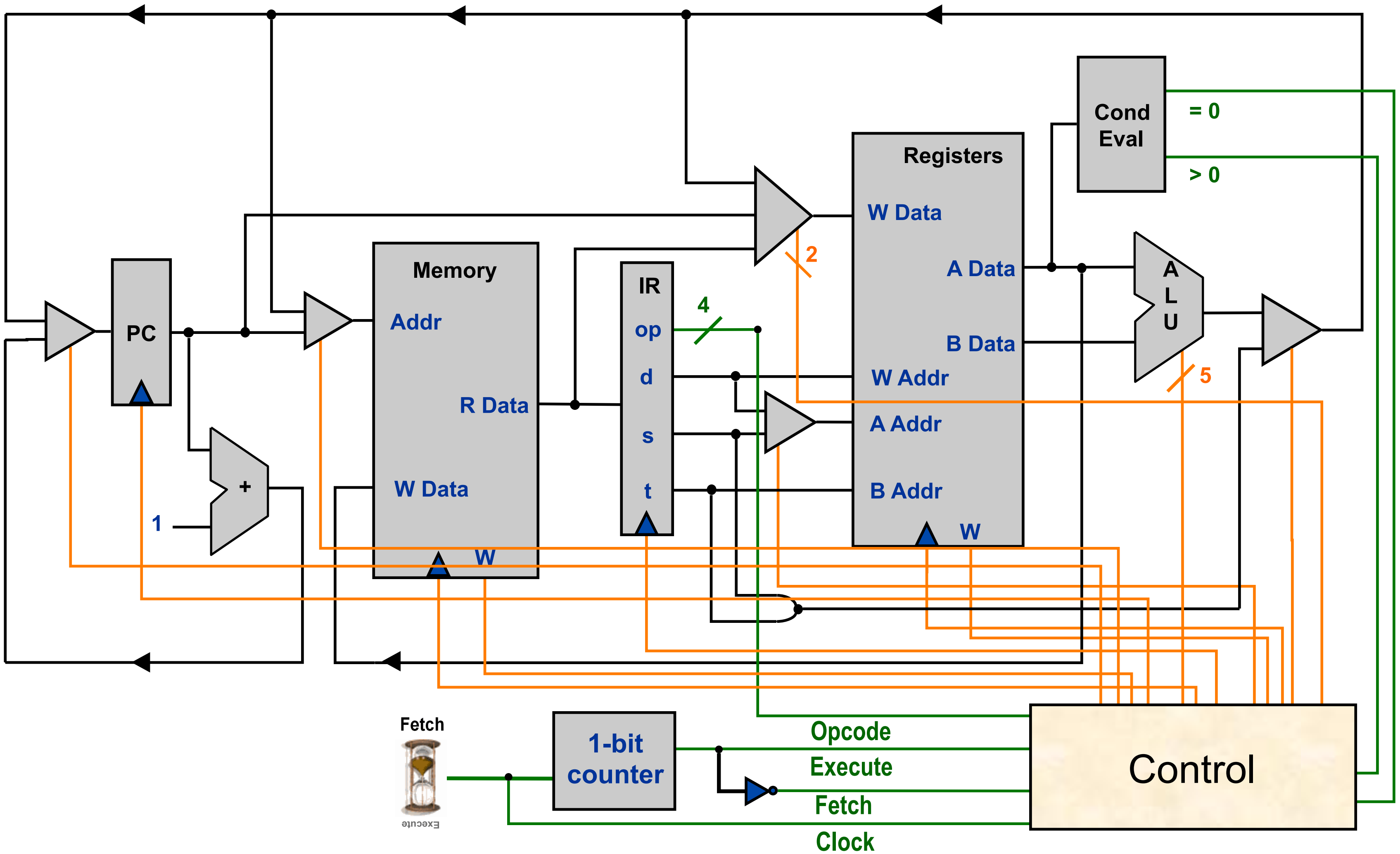
- Input: opcode, clock, conditional evaluation. (green)
- Output: control wires. (orange)



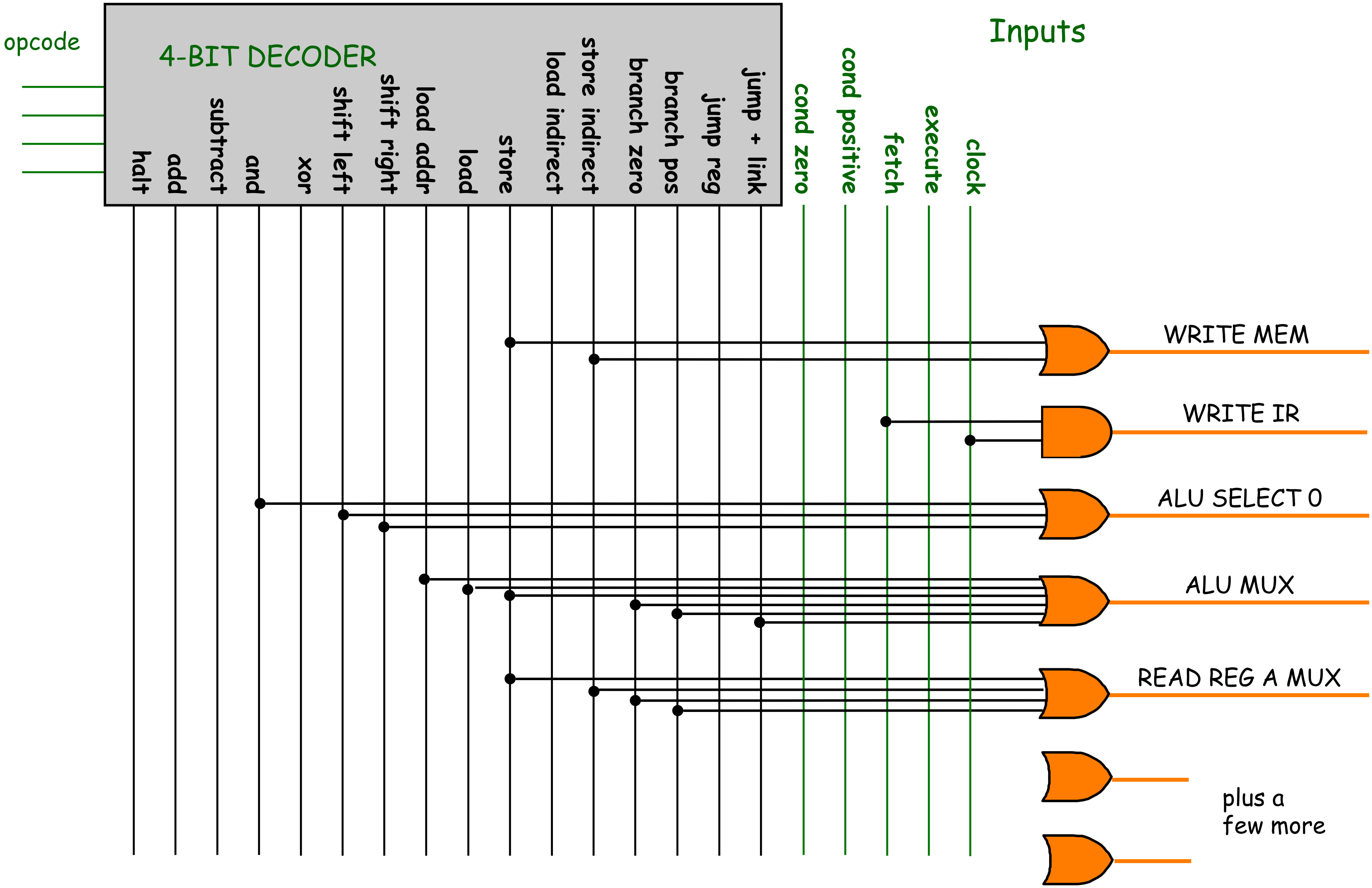
Control

Control: controls components, enables connections.

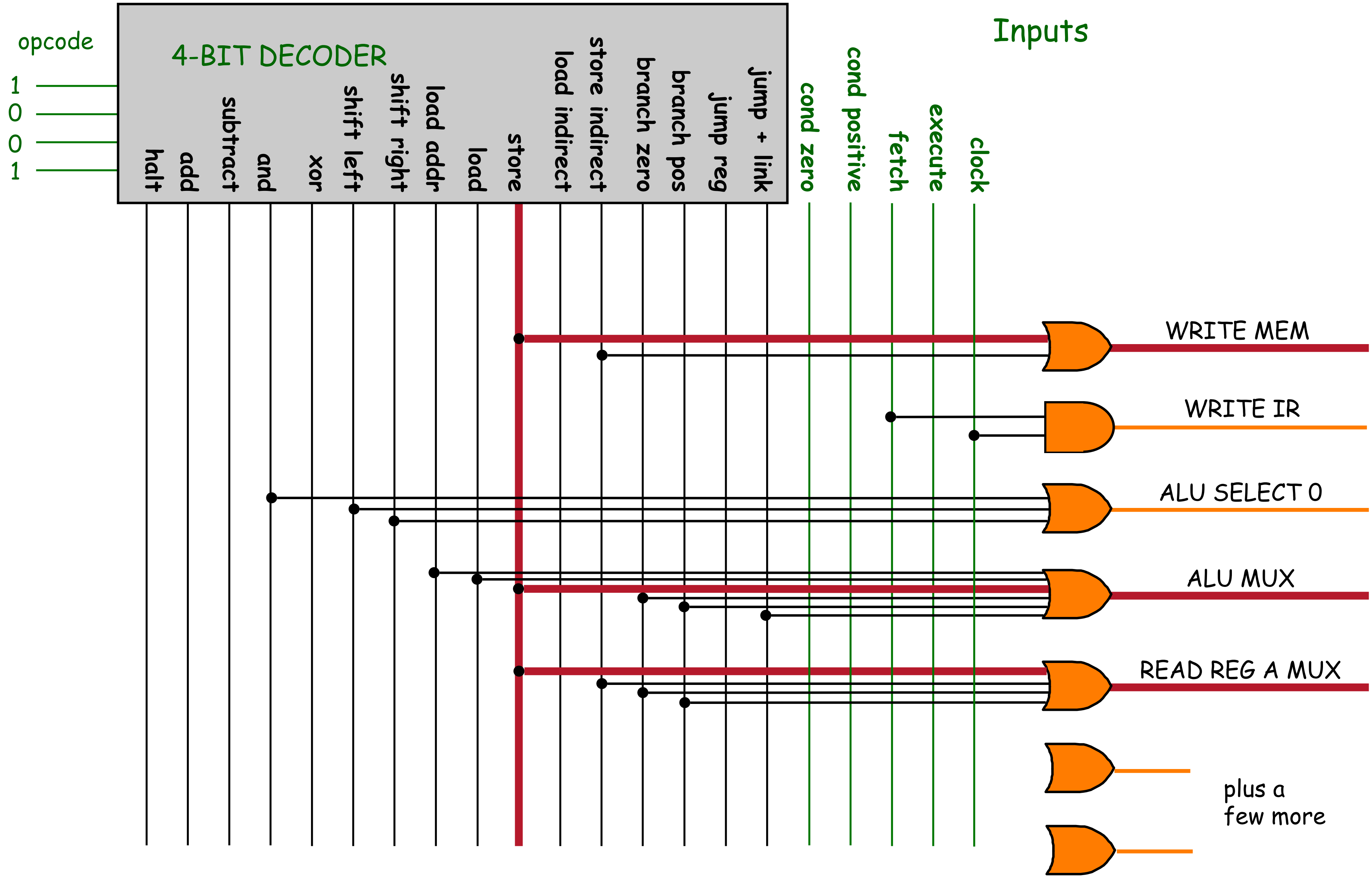
- Input: opcode, clock, conditional evaluation. (green)
- Output: control wires. (orange)



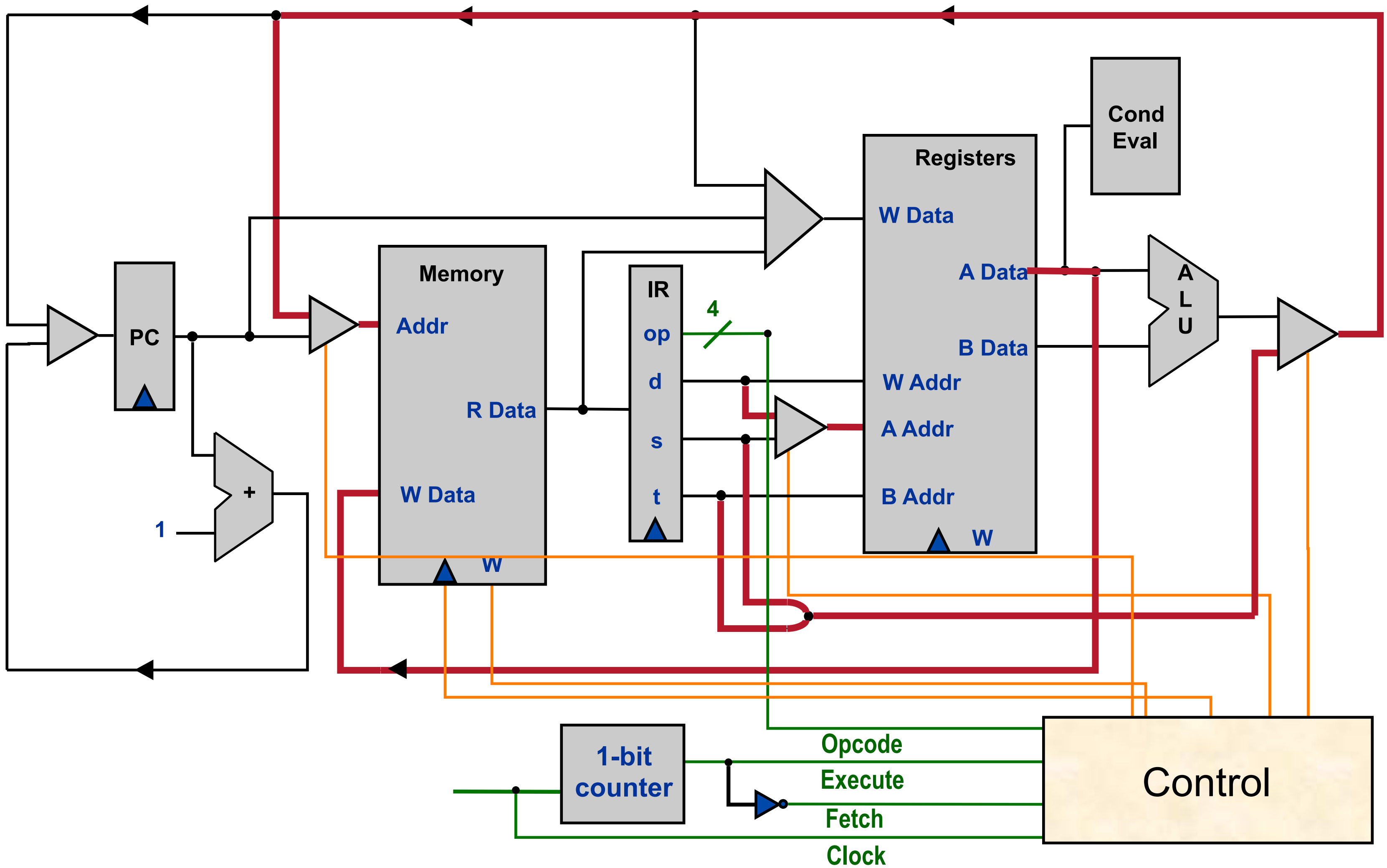
Implementation of Control



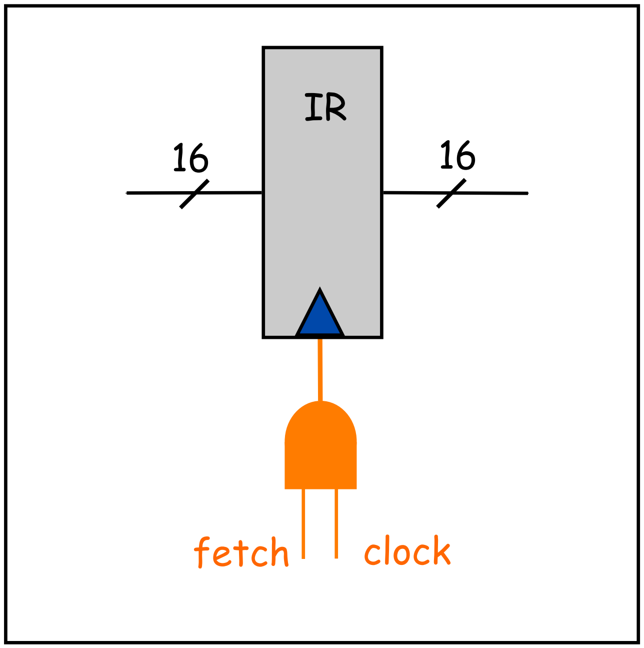
Implementation of Control: Store



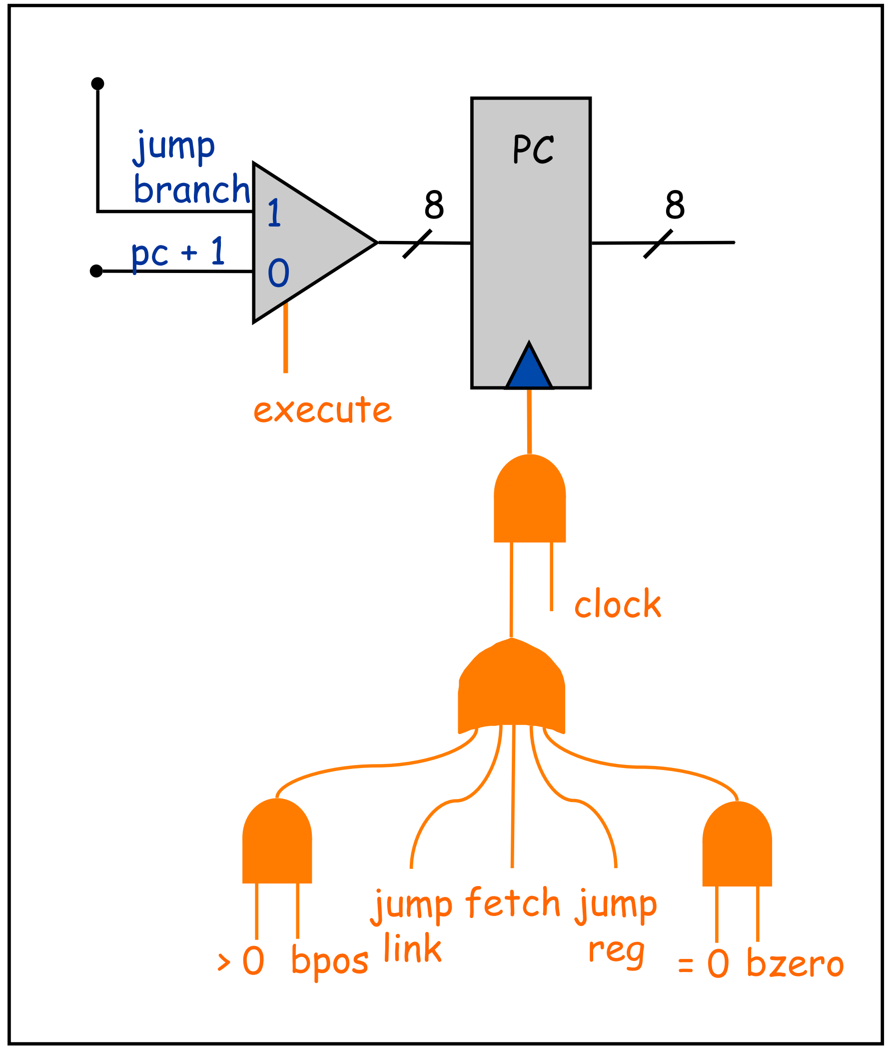
Control: Execute Phase of Store



Stand-Alone Registers



Instruction Register



Program Counter

Pipelining

Pipelining.

- At any instant, processor is either fetching instructions or executing them (and so half of circuitry is idle).
- Why not fetch next instruction while current instruction is executing?
  - Analogy: washer / dryer.

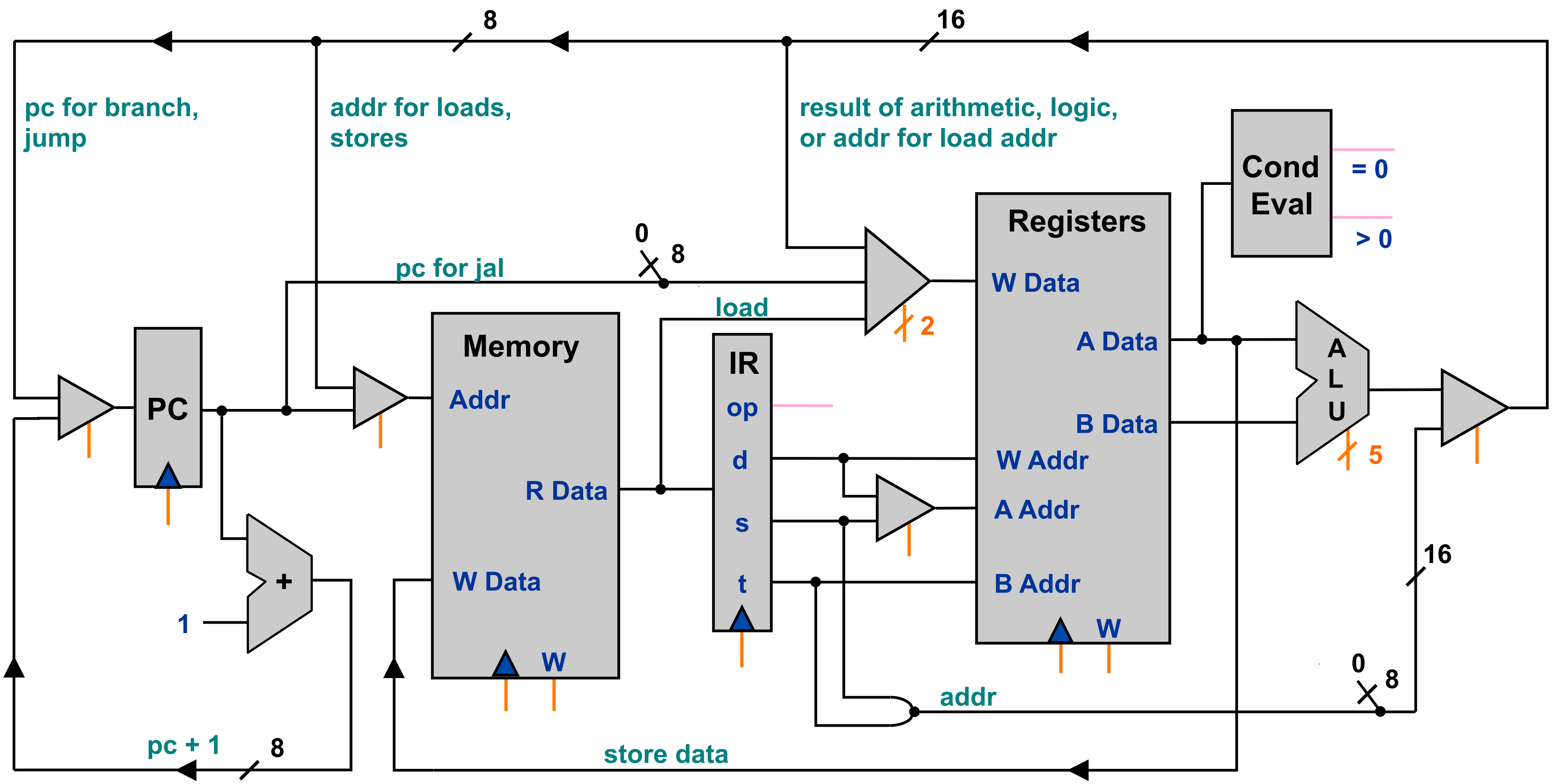
Issues.

- Jump and branch instructions change PC.
  - "Prefetch" next instruction.
- Fetch and execute cycles may need to access same memory.
  - Solution: use two memory “caches”.

Result.

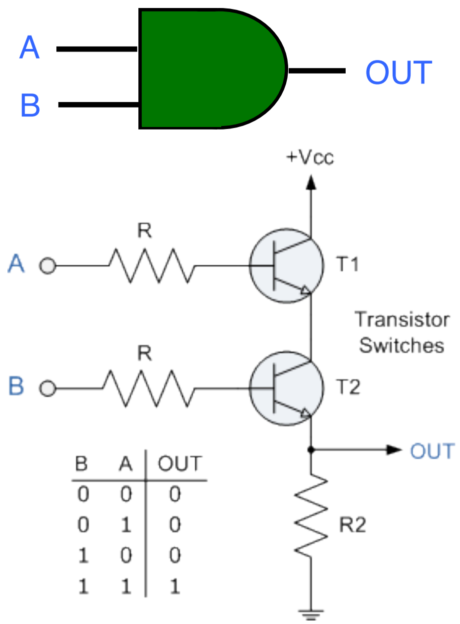
- Better utilization of hardware.
- Can double speed of processor.

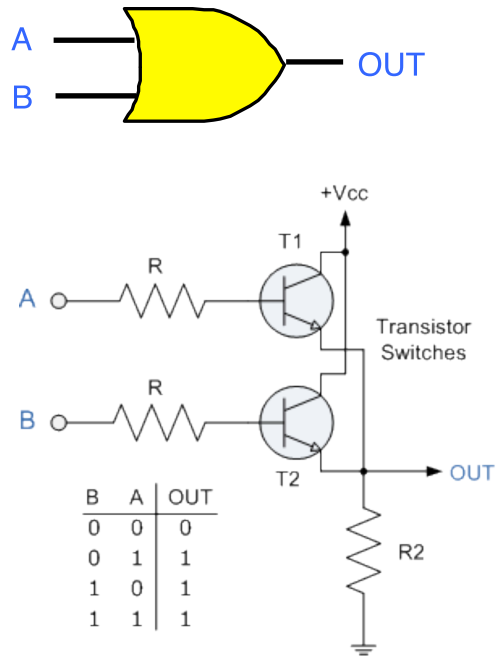
Goodbye, TOY

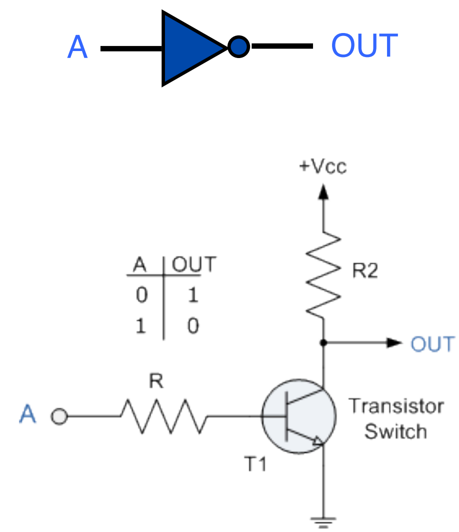


8
16
pc for branch, jump
addr for loads, stores
result of arithmetic, logic, or addr for load addr
Cond Eval
= 0
> 0
Registers
W Data
A Data
B Data
W Addr
A Addr
B Addr
W
pc for jal
0
8
load
2
Memory
Addr
R Data
W Data
W
IR
op
d
s
t
PC
A L U
5
+
1
16
0
8
addr
pc + 1
8
store data

The final secret







All three of our logic primitives can be made
using a *single** type of electronic primitive: the *transistor* !

*not counting the passive resistors