CSEE 3827: Fundamentals of Computer Systems, Spring 2022

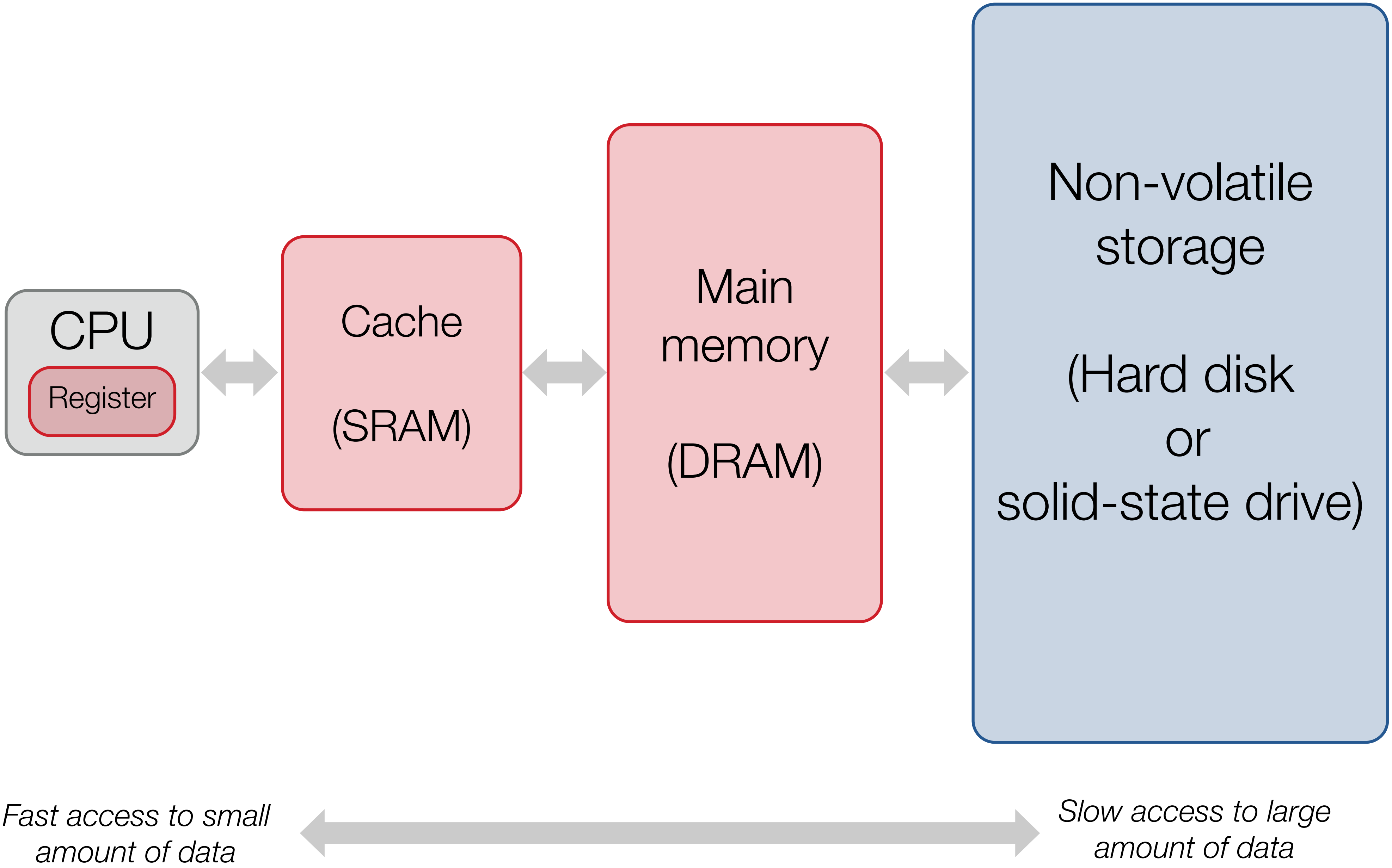
Lecture 10

Prof. Dan Rubenstein (danr@cs.columbia.edu)

Agenda (M&K 7.1-7.4)

- Memory

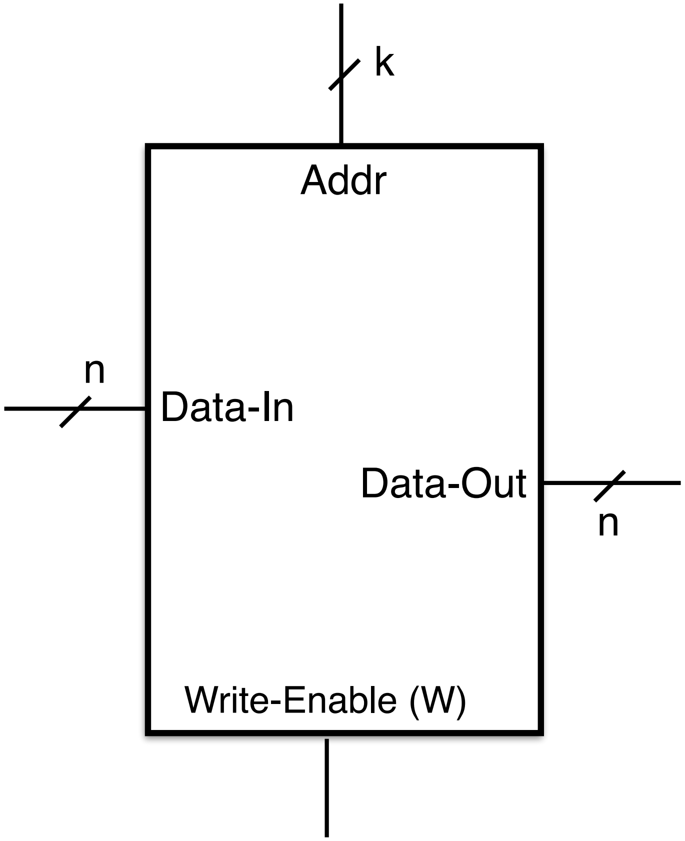
Storage hierarchy



Using Memory

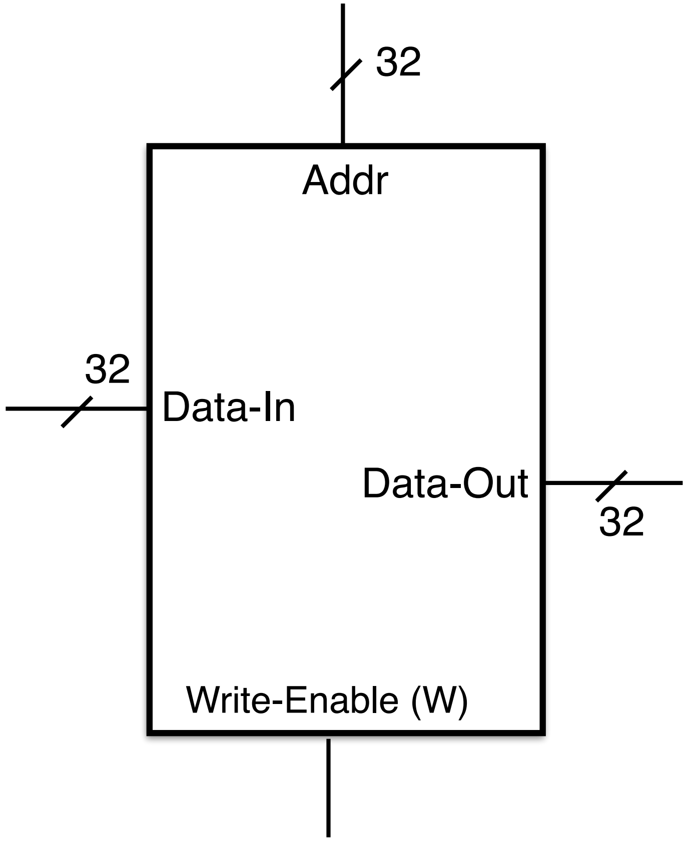
General Memory interface

- k-bit addresses (2^k addresses), n-bit data
- Read from memory:
  - Specify a (k-bit) address, A
  - Data-out outputs the n-bit contents in memory at address, A
- Write to memory:
  - Specify a (k-bit) address, A
  - Through Data-In, supply n-bit data to write at address A
  - set Write-Enable to 1
- Careful, if wanting to read and Write-Enable set to 1, will also be writing.



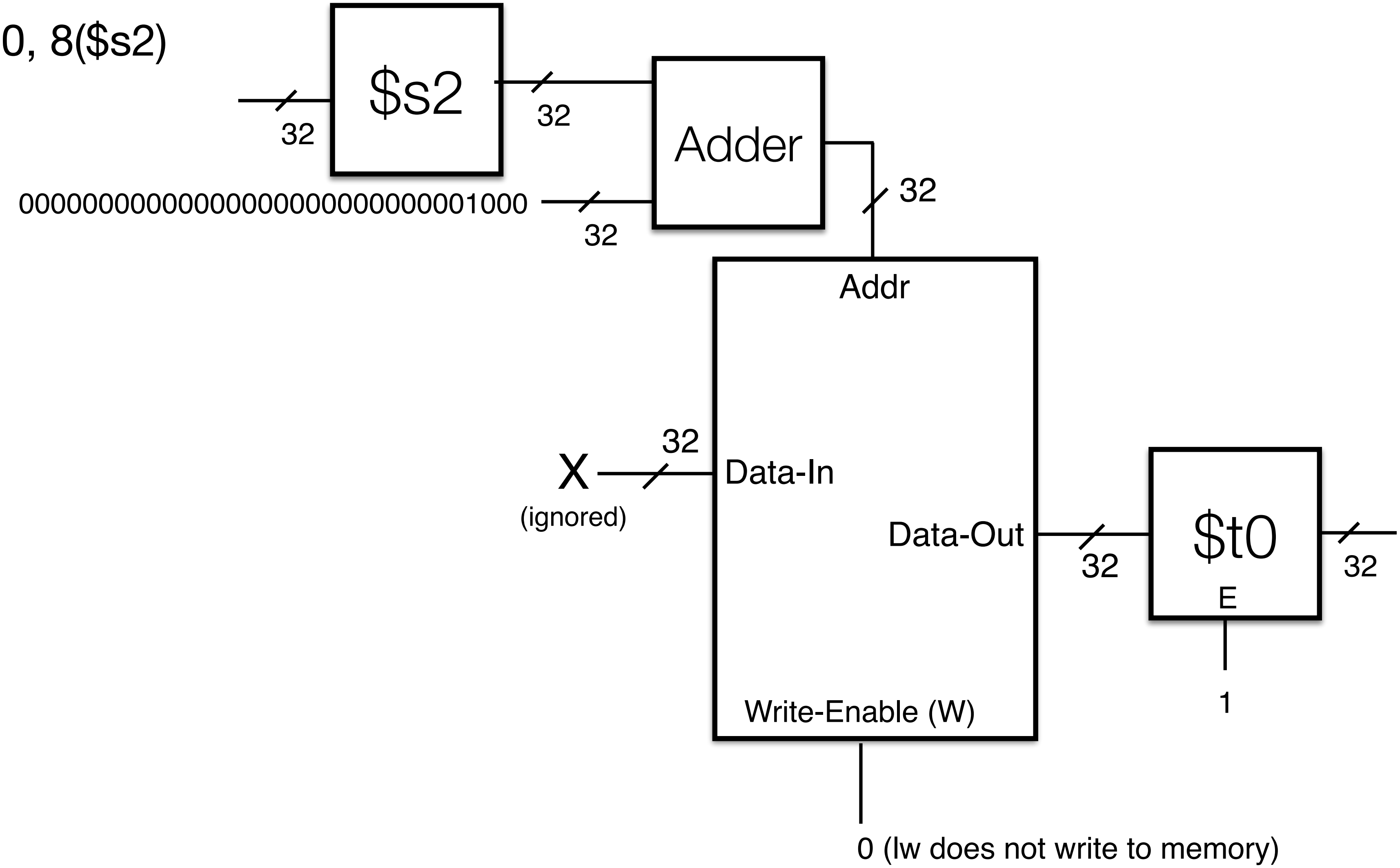
MIPS-specific

- k=32: 32-bit addresses (2^{32} byte-size addressable slots)
- n=32: we access memory 32-bits at a time
- 32-bit address always multiple of 4 (2 least significant bits always “00”)



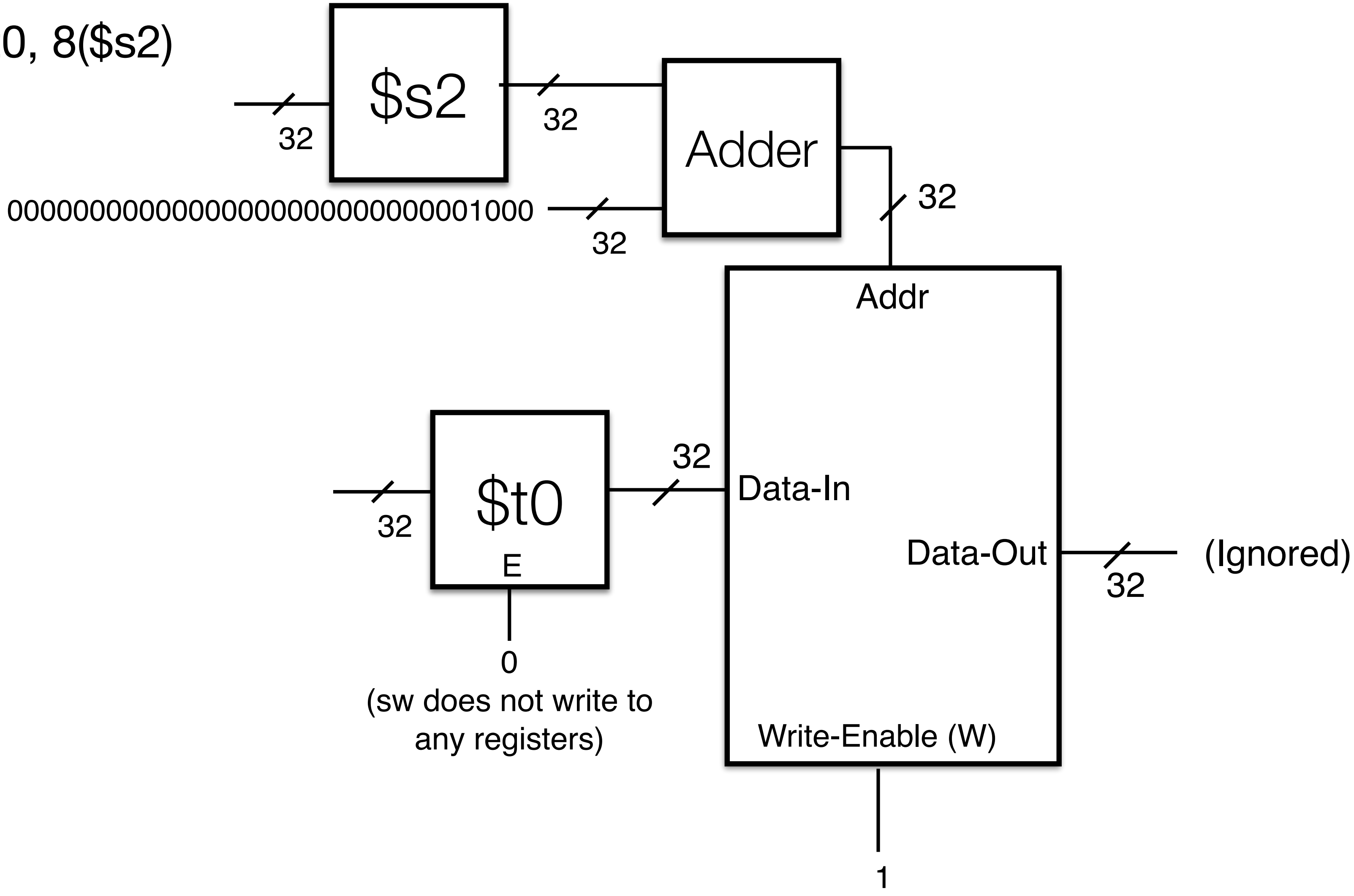
MIPS-specific read example

- e.g., lw $t0, 8($s2)



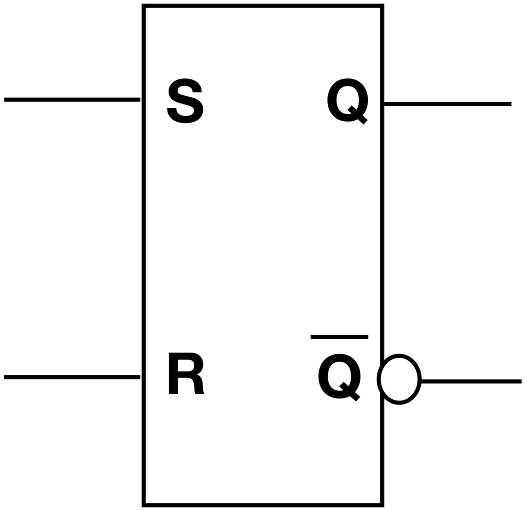
MIPS-specific read example

- e.g., sw $t0, 8($s2)



Building Memory

Recall: SR Latch

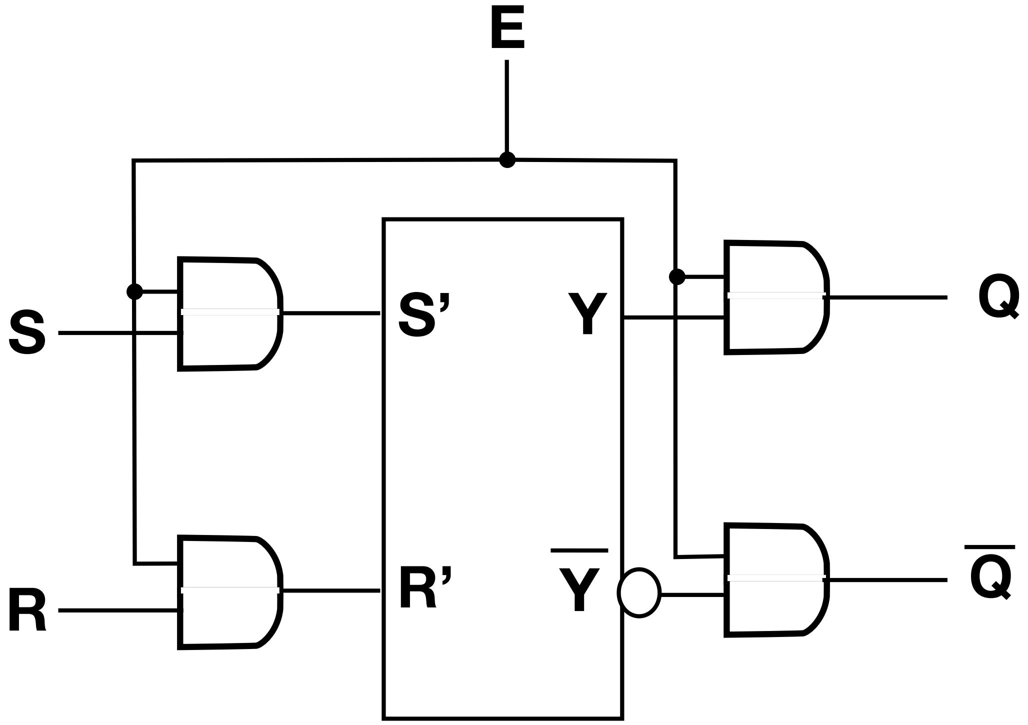


S	R	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	Hold value
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	

Single (1-bit) Memory Cell

Memory “Cell”

- memory storage of a single bit
- built from SR-latch and some enablers (AND gates)

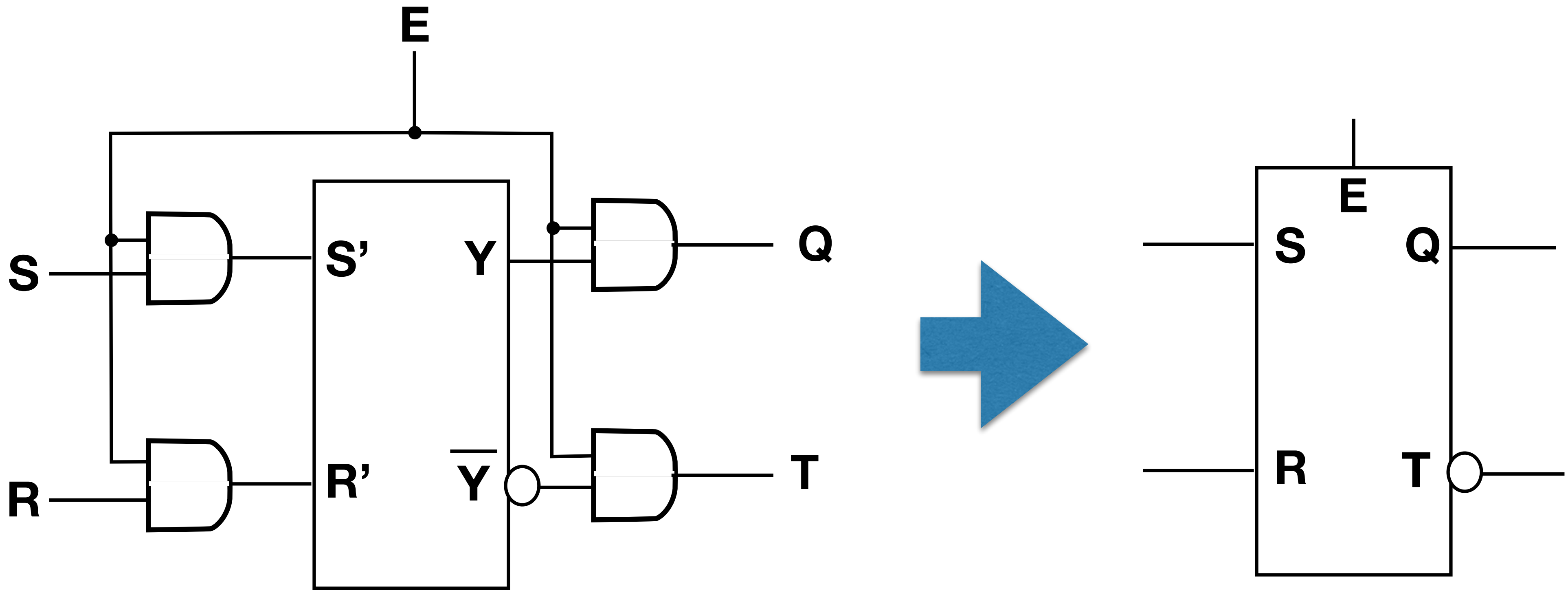


E	S	R	Y	$\overline{Y}$	Q	$\overline{Q}$	
0	X	X	Y	$\overline{Y}$	0	0	Disable Inputs and Zero Outputs
1	0	0	Y	$\overline{Y}$	Y	$\overline{Y}$	Hold Value
1	0	1	0	1	0	1	Reset (to 0)
1	1	0	1	0	1	0	Set (to 1)
1	1	1	0	0	0	0	

Note: Setting E=0 does not erase value in latch, just “zeros” the value being output

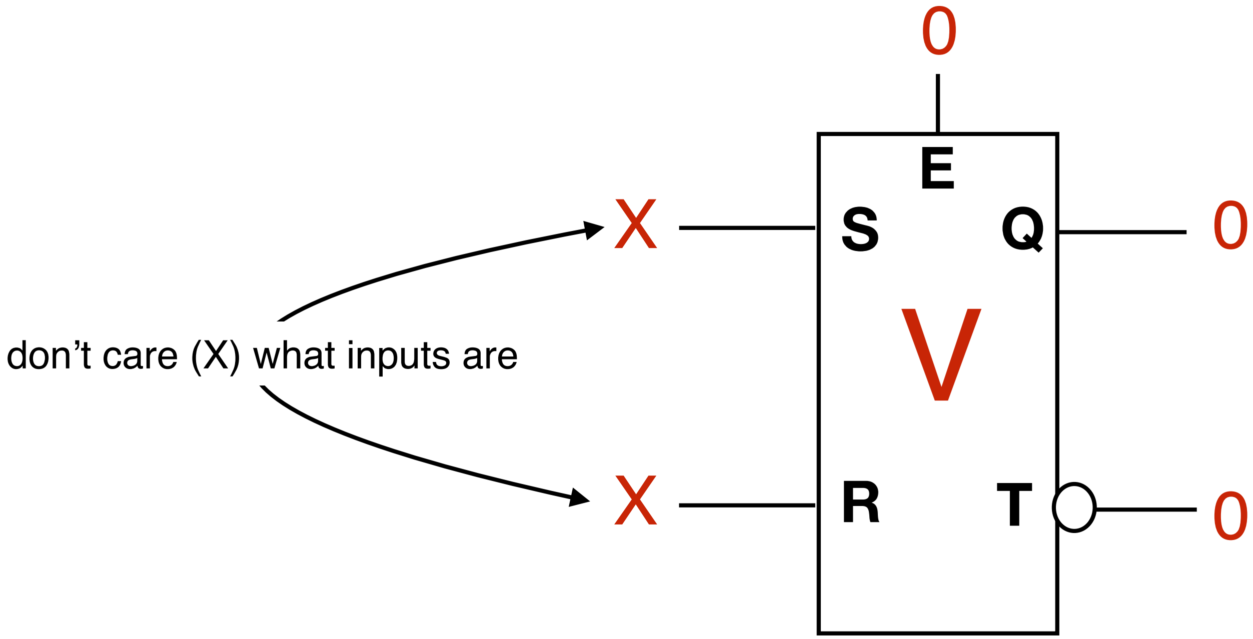
Memory "Cell"

- memory storage of a single bit
- built from SR-latch and some enablers (AND gates)



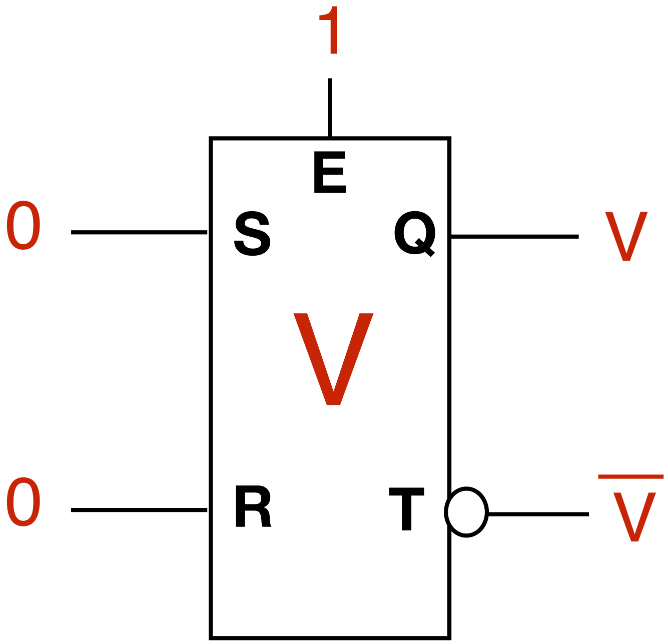
Disable a Cell

- Set E=0, S=X,R=X
- Output not 0’d, cell (1-bit) storage unchanged



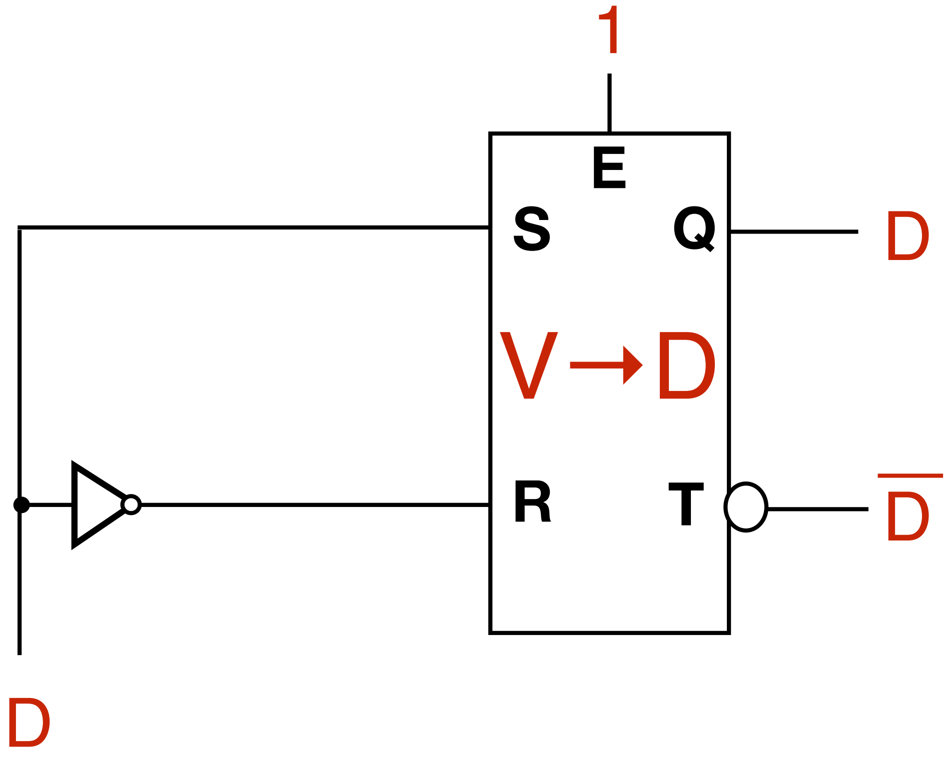
Read from a Cell

- Set E=1, S=R=0
- Output not 0’d, cell (1-bit) storage unchanged



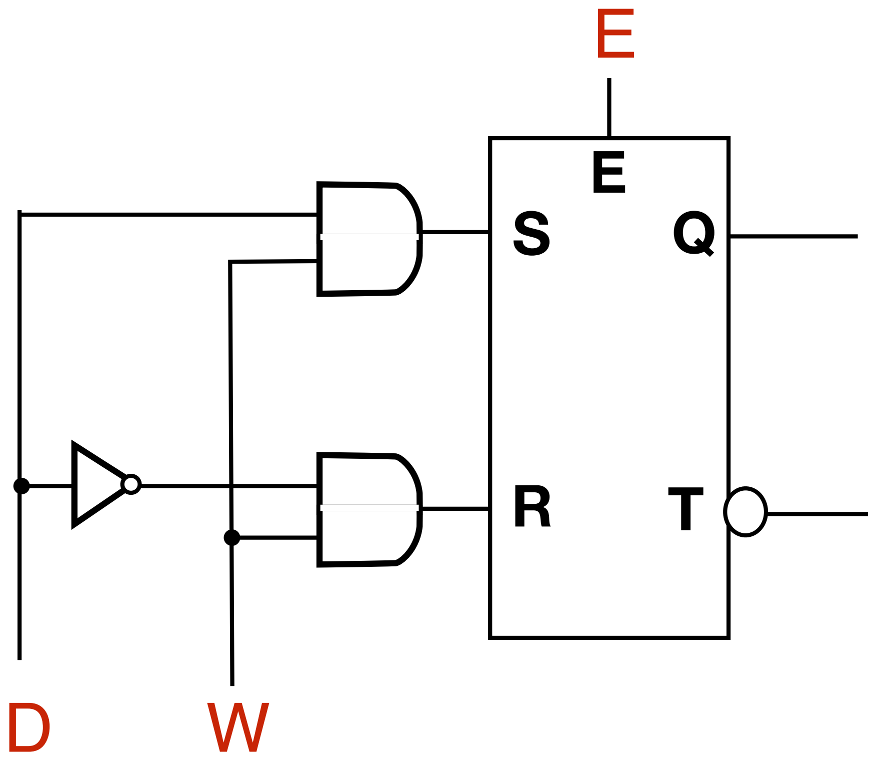
Write to a Cell

- To write a data bit D into the cell
  - Enable the cell (E=1)
  - Set S=D, R=$\overline{D}$



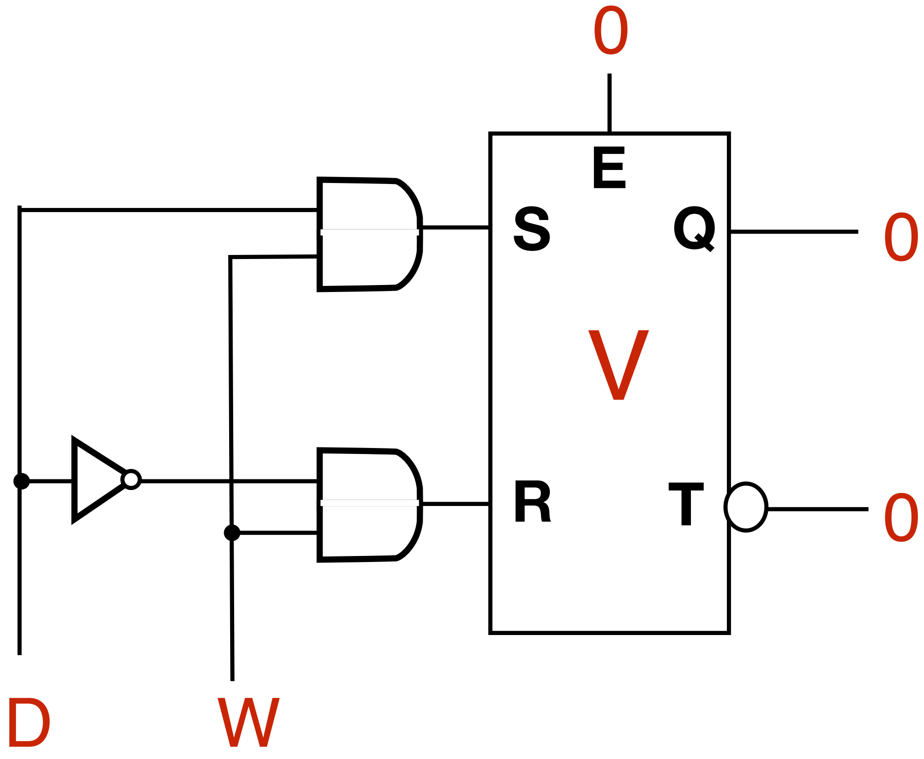
Reading or Writing

- E=0, ignore all inputs (outputs both 0)
- E=1
    - W=0: output bit stored in cell
    - W=1, write D into cell (also the output of the cell)



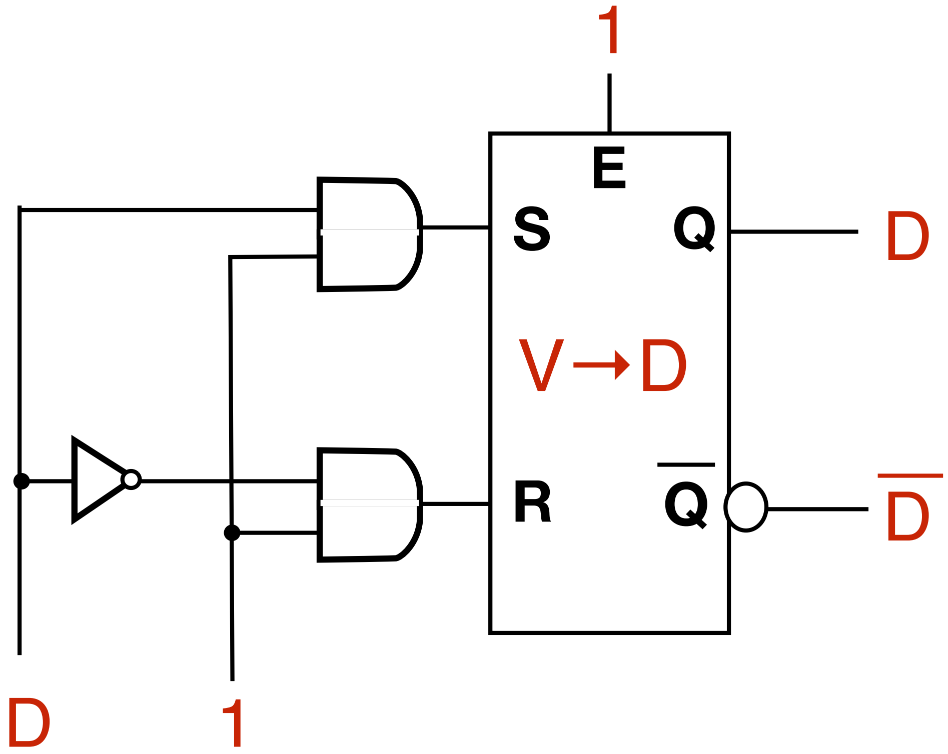
Reading or Writing

- E=0, ignore all inputs (outputs both 0)
- E=1
  - W=0: output bit stored in cell
  - W=1, write D into cell (also the output of the cell)



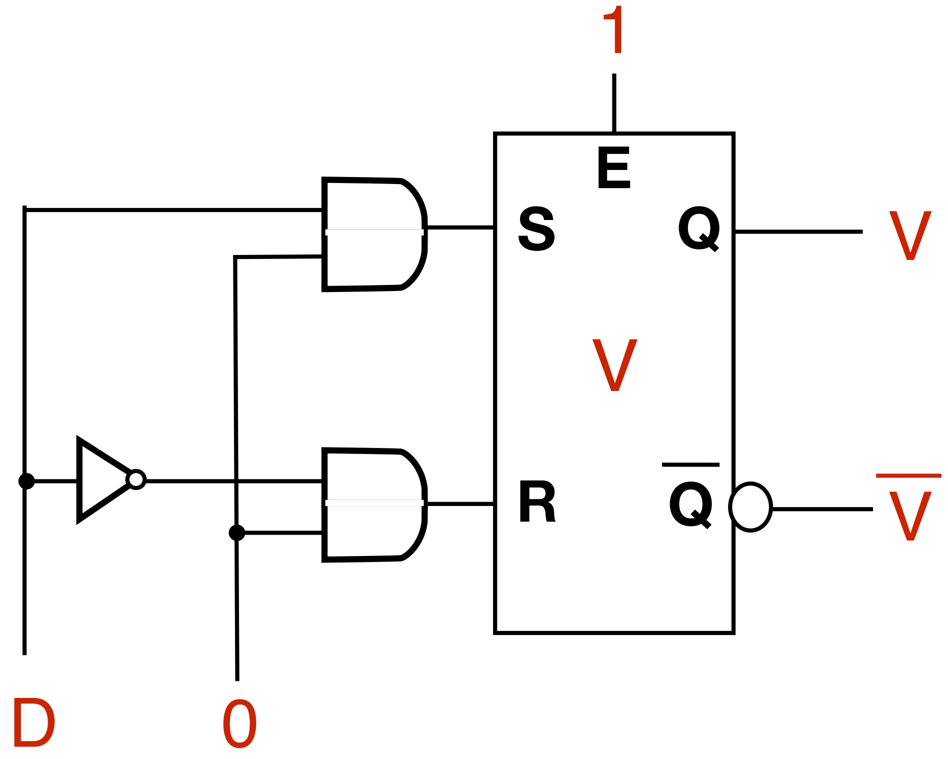
Reading or Writing

- E=0, ignore all inputs (outputs both 0)
- E=1
  - W=0: output bit stored in cell
  - W=1, write D into cell (also the output of the cell)



Reading or Writing

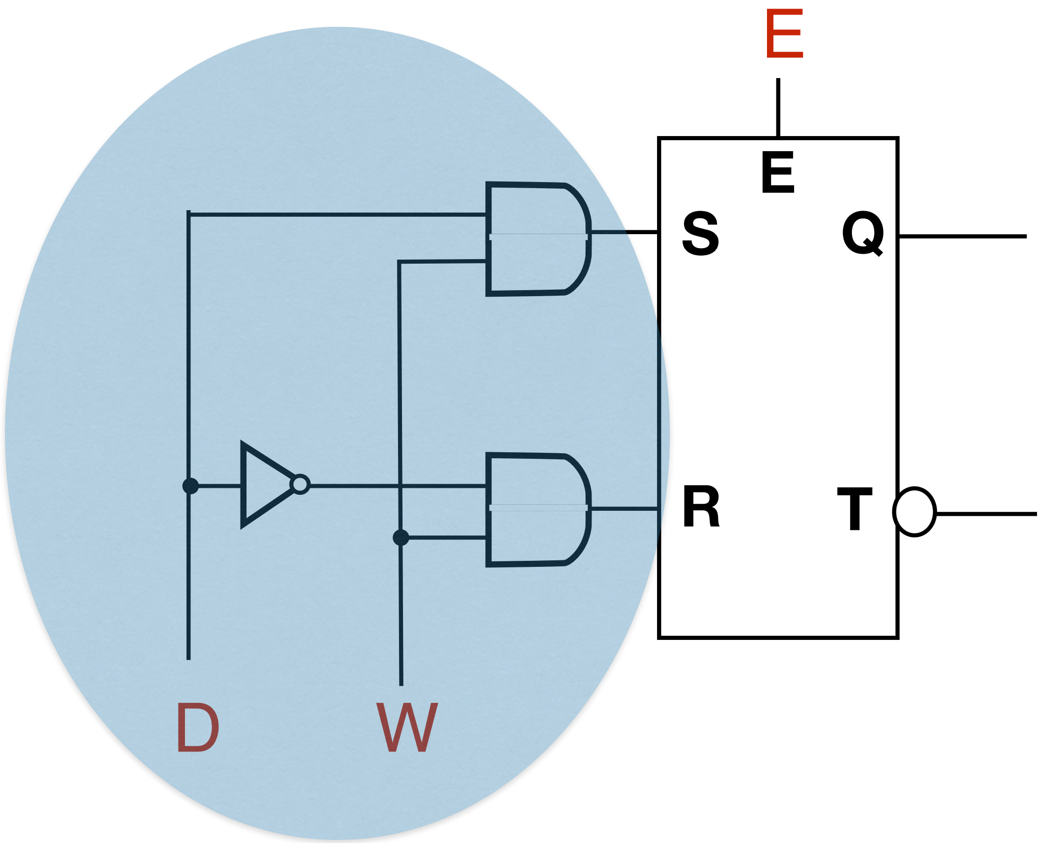
- E=0, ignore all inputs (outputs both 0)
- E=1
    - W=0: output bit stored in cell
    - W=1, write D into cell (also the output of the cell)



Reading or Writing

- E=0, ignore all inputs (outputs both 0)
- E=1
  - W=0: output bit stored in cell
  - W=1, write D into cell (also the output of the cell)

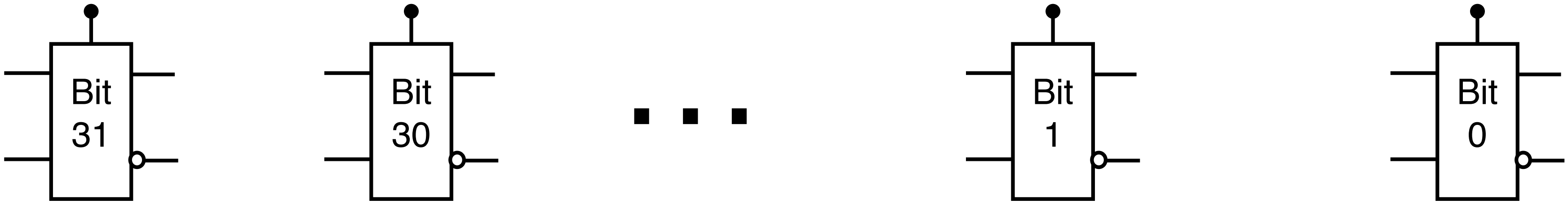
Write Logic for Memory Cell



Memory Word
(MIPS: 32 bits)

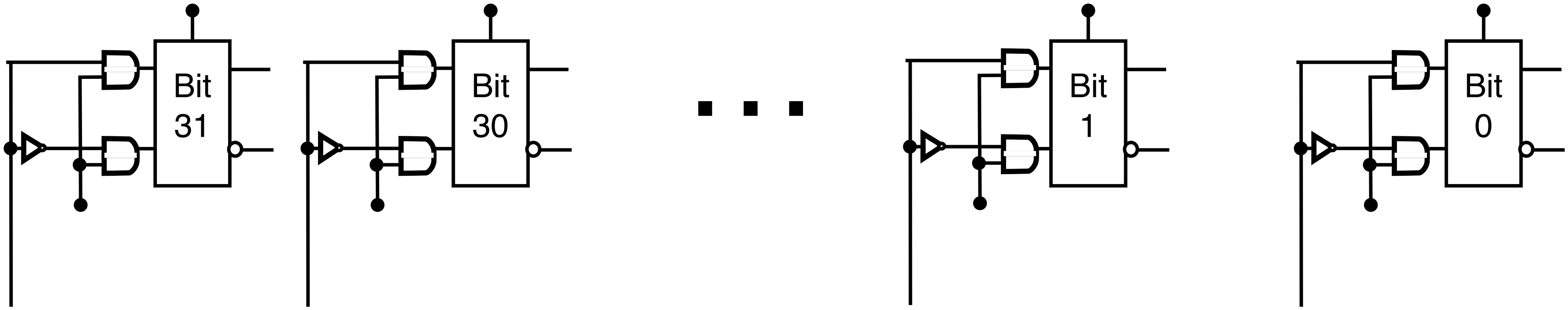
Word of MIPS Memory

- memory storage of a single bit
- built from SR-latch and some enablers (AND gates)



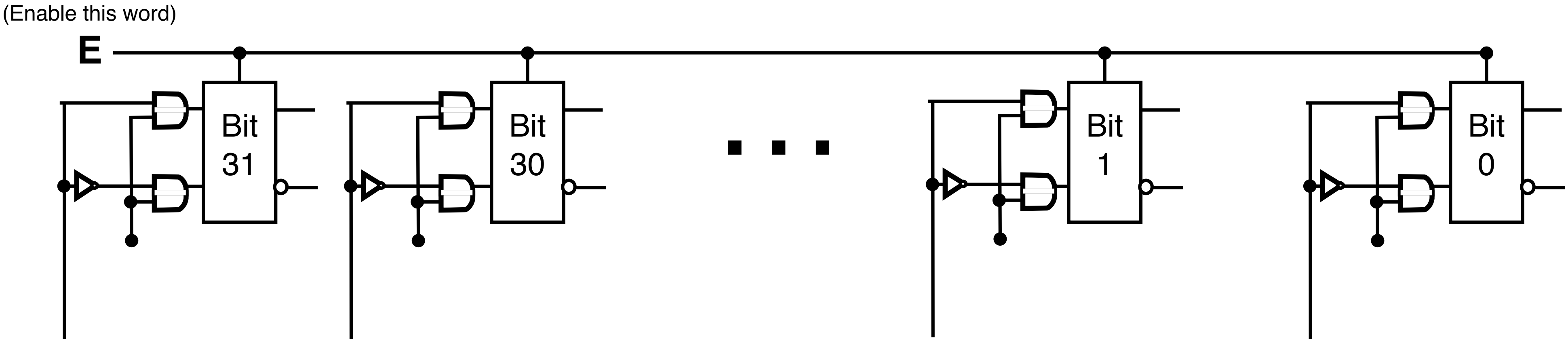
32 latches w/ Enable

Word of MIPS Memory



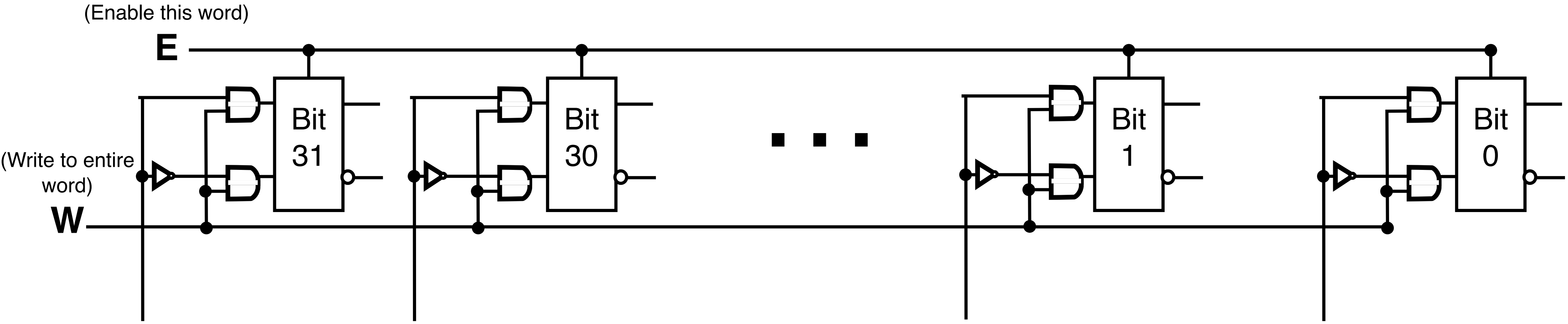
Attach Write Logic to each

Word of MIPS Memory



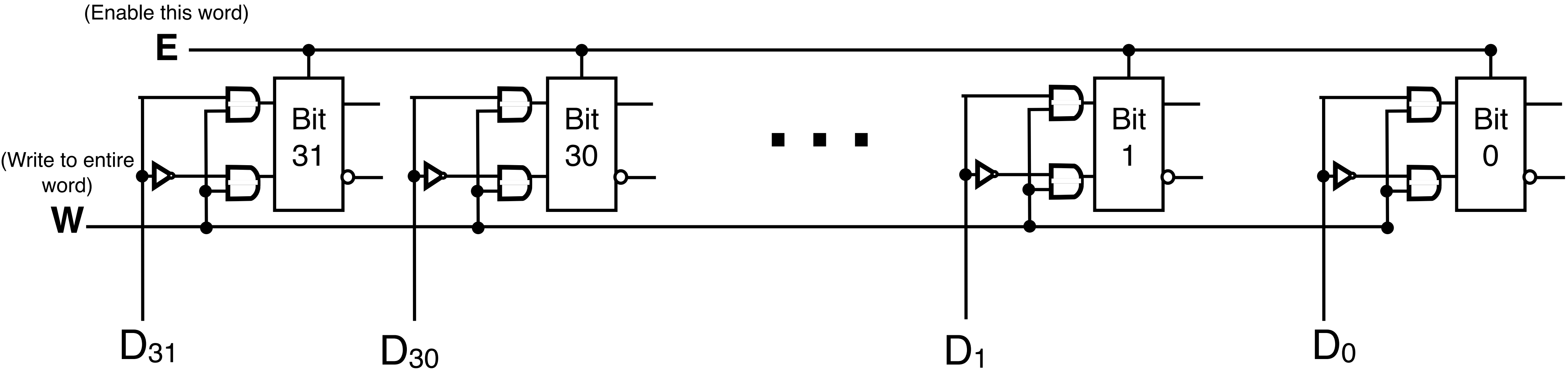
Attach to same enable signal
(Enable all 32 or none)

Word of MIPS Memory



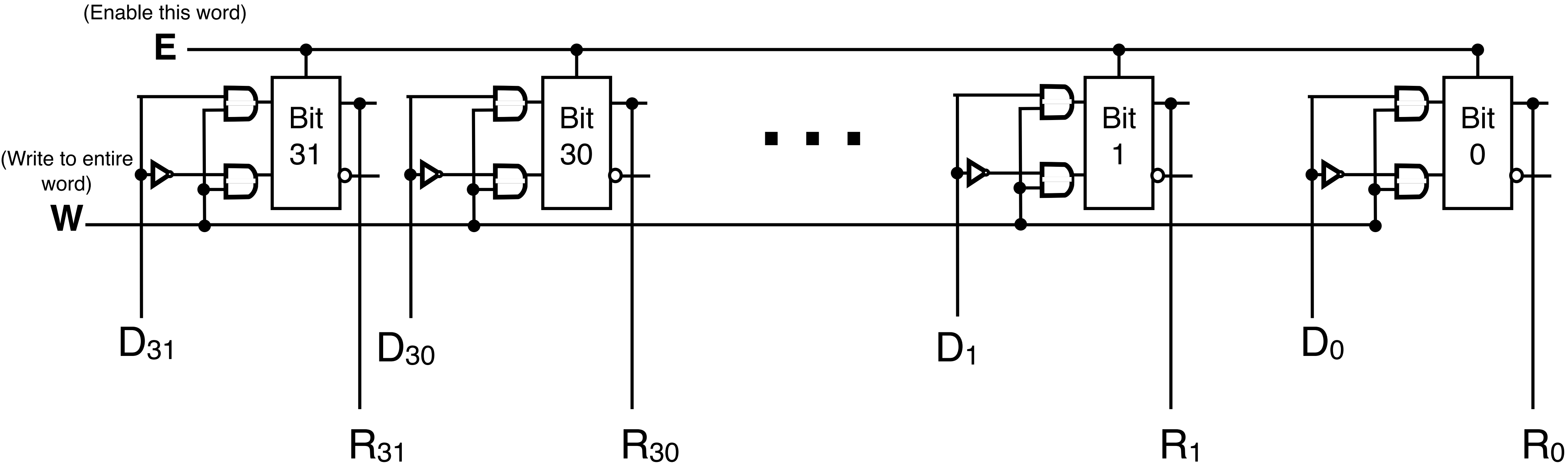
Attach to same write signal
(write to all 32 (when enabled) or none)

Word of MIPS Memory



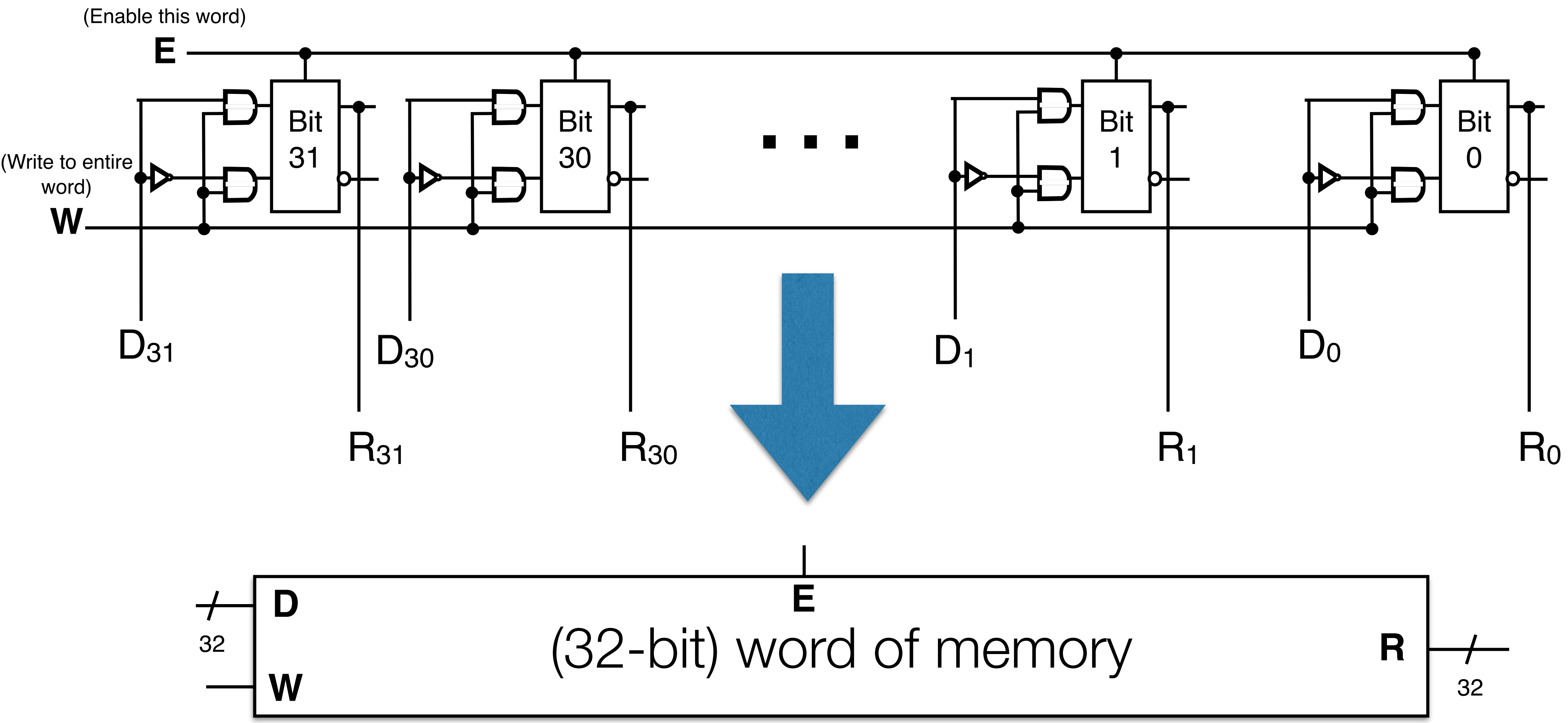
Attach 32-bit Data-In signal, D, bit-by-bit to corresponding latches

Word of MIPS Memory



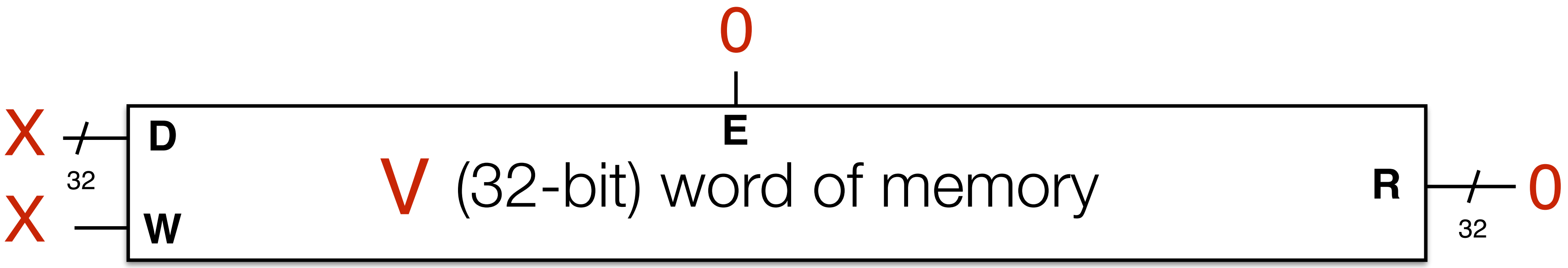
Attach 32-bit Data-Out signal, R, bit-by-bit to corresponding latches

Word of MIPS Memory



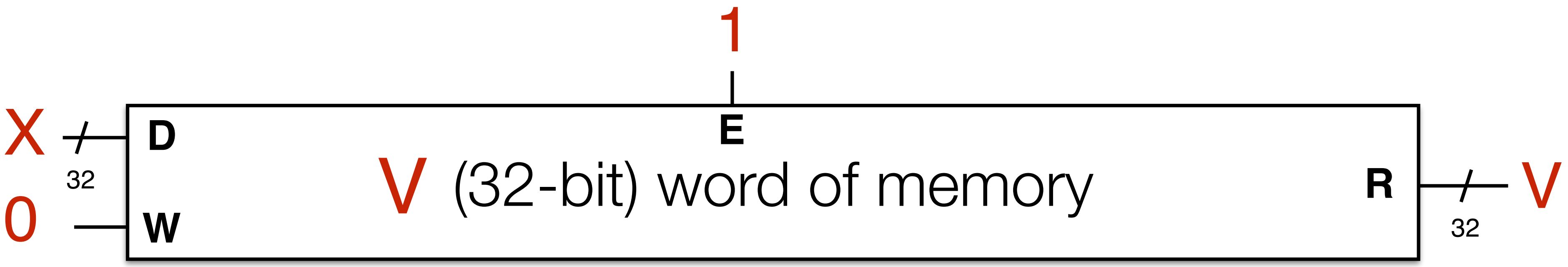
Word of MIPS Memory

- E=0: don’t write to this word of memory, output all 0’s



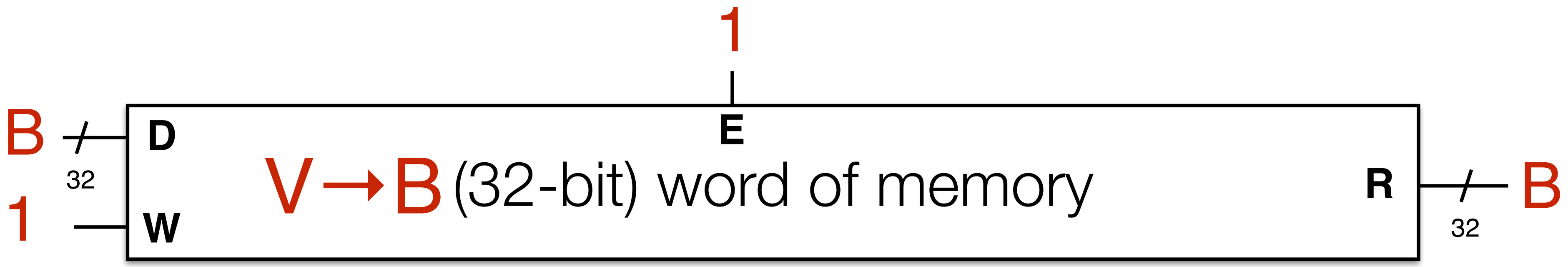
Word of MIPS Memory

- E=1, W=0: read from this word of memory, but don't write

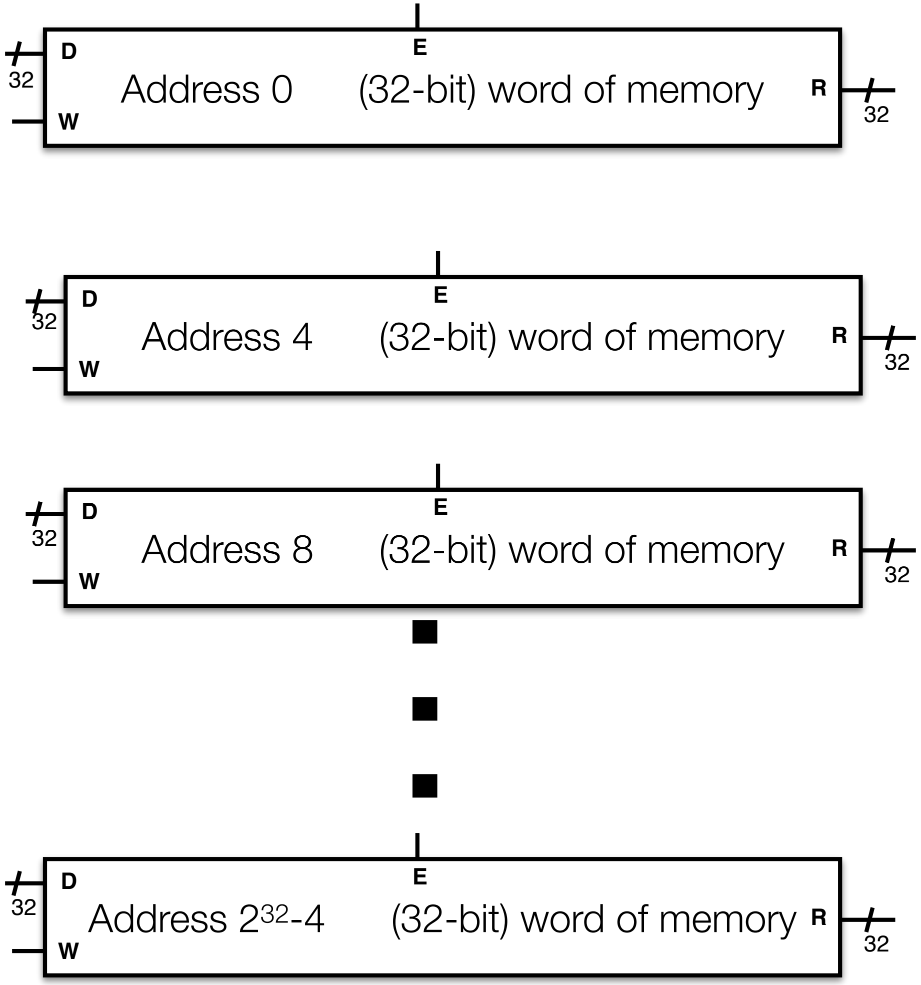


Word of MIPS Memory

- E=1, W=1: write input at D into memory (read as well)

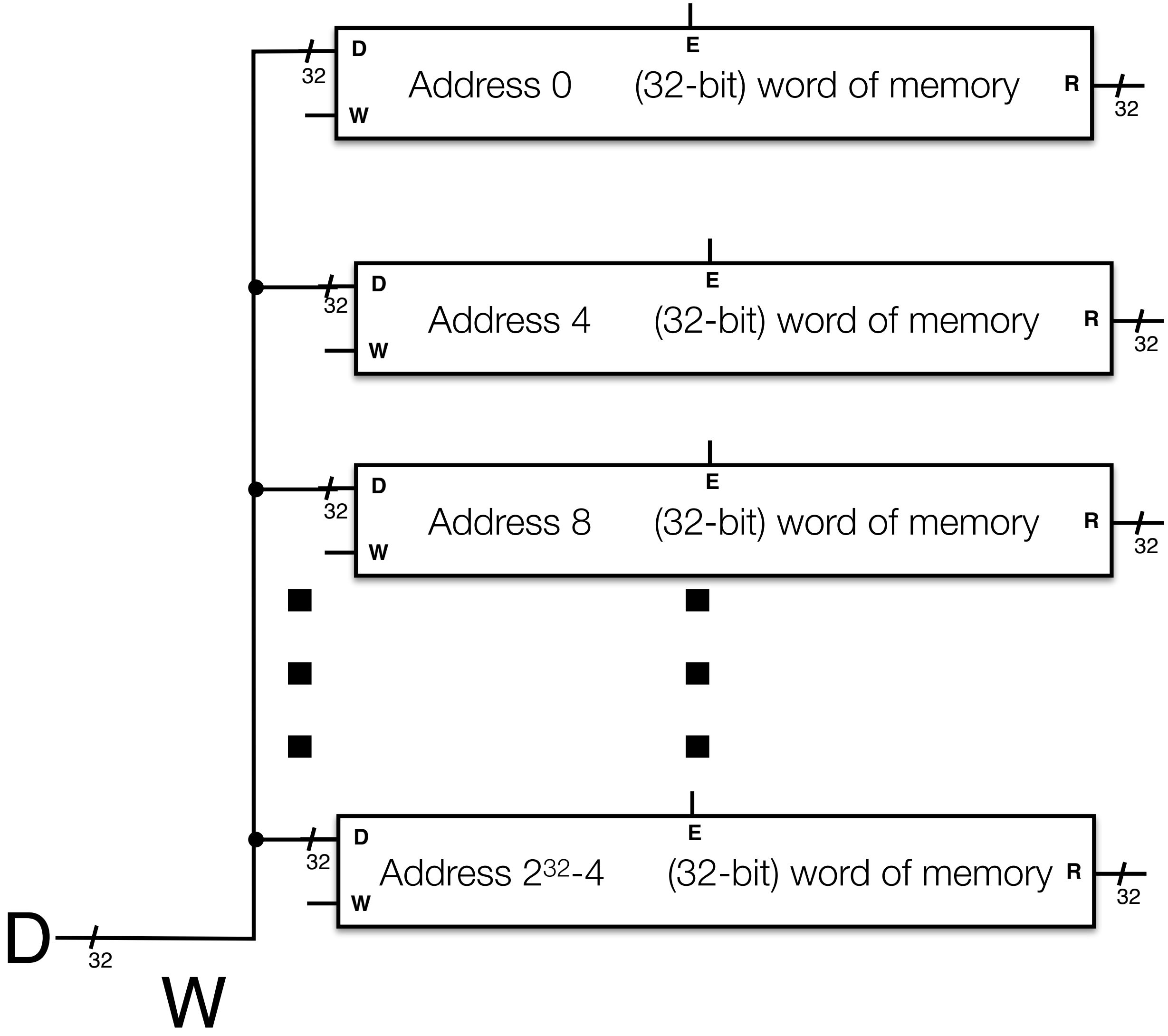


2^{32} bytes = 2^{30} words of Addressable Memory



2^{32} bytes = 2^{30} words of Addressable Memory

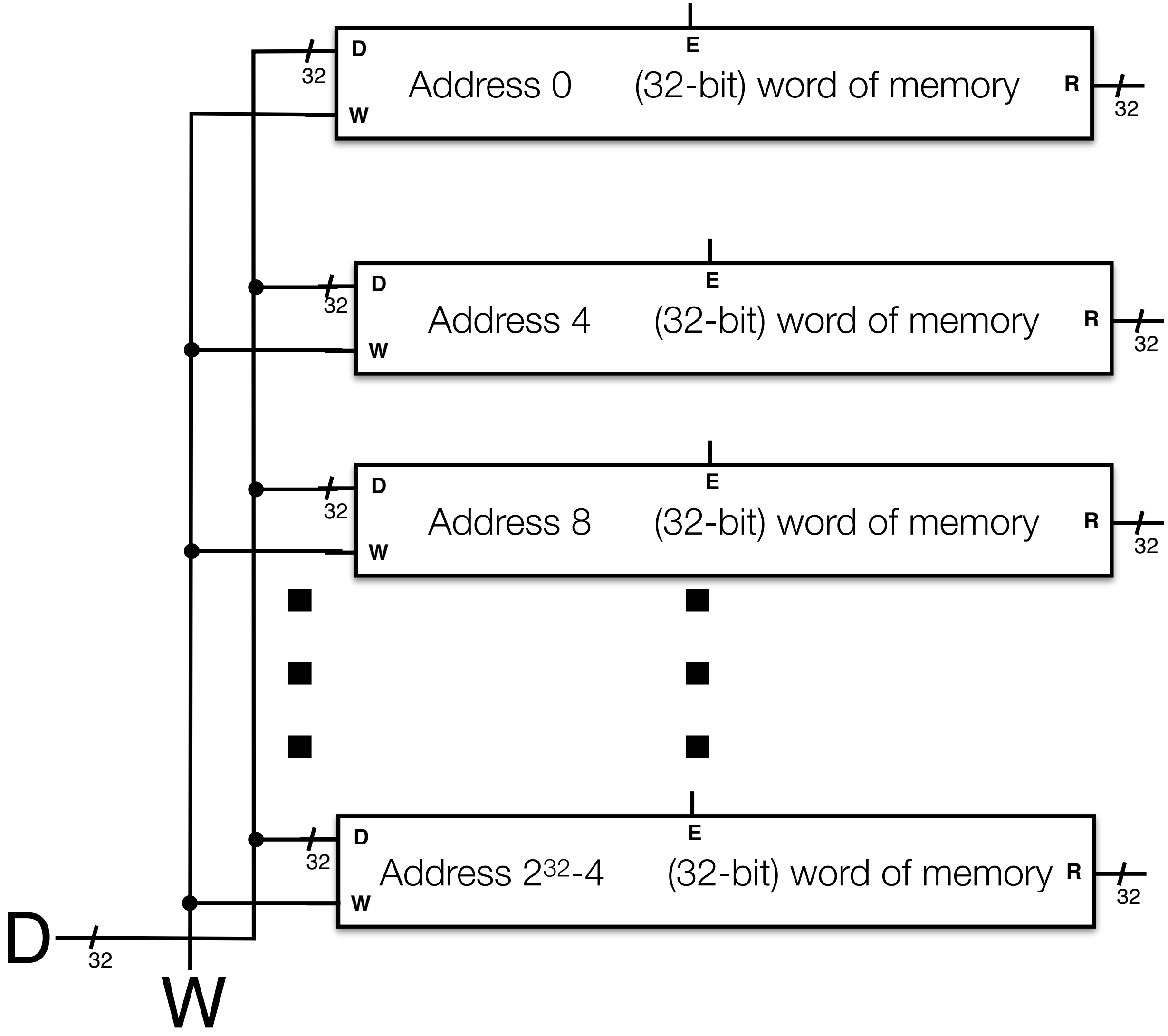
[30 bits describing address] 00



- Each address is really described by 30 bits (since the 2 lowest-order bits of the 32-bit address are 00)
- Data Input to memory, D, fed into every word address

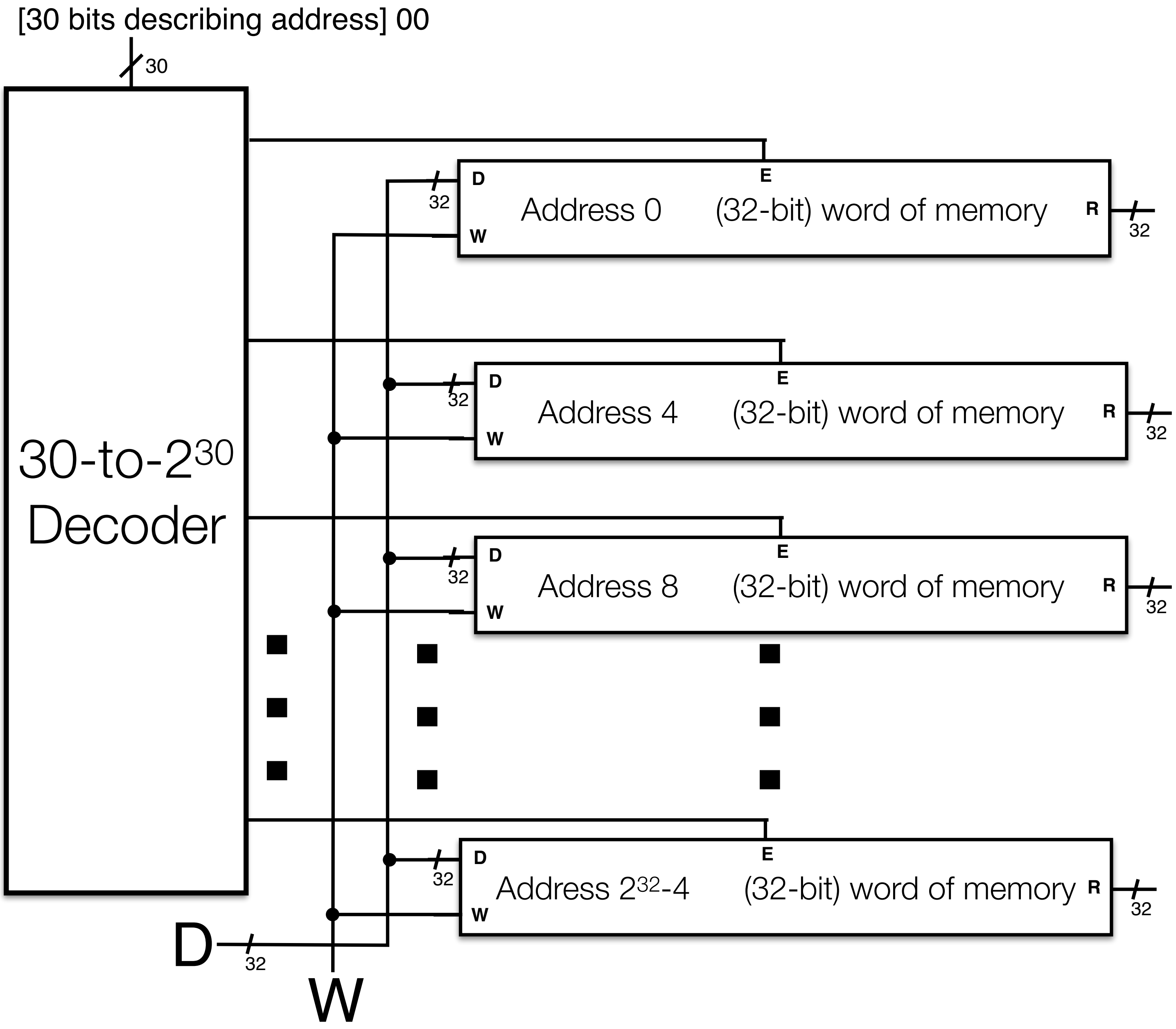
2^{32} bytes = 2^{30} words of Addressable Memory

[30 bits describing address] 00



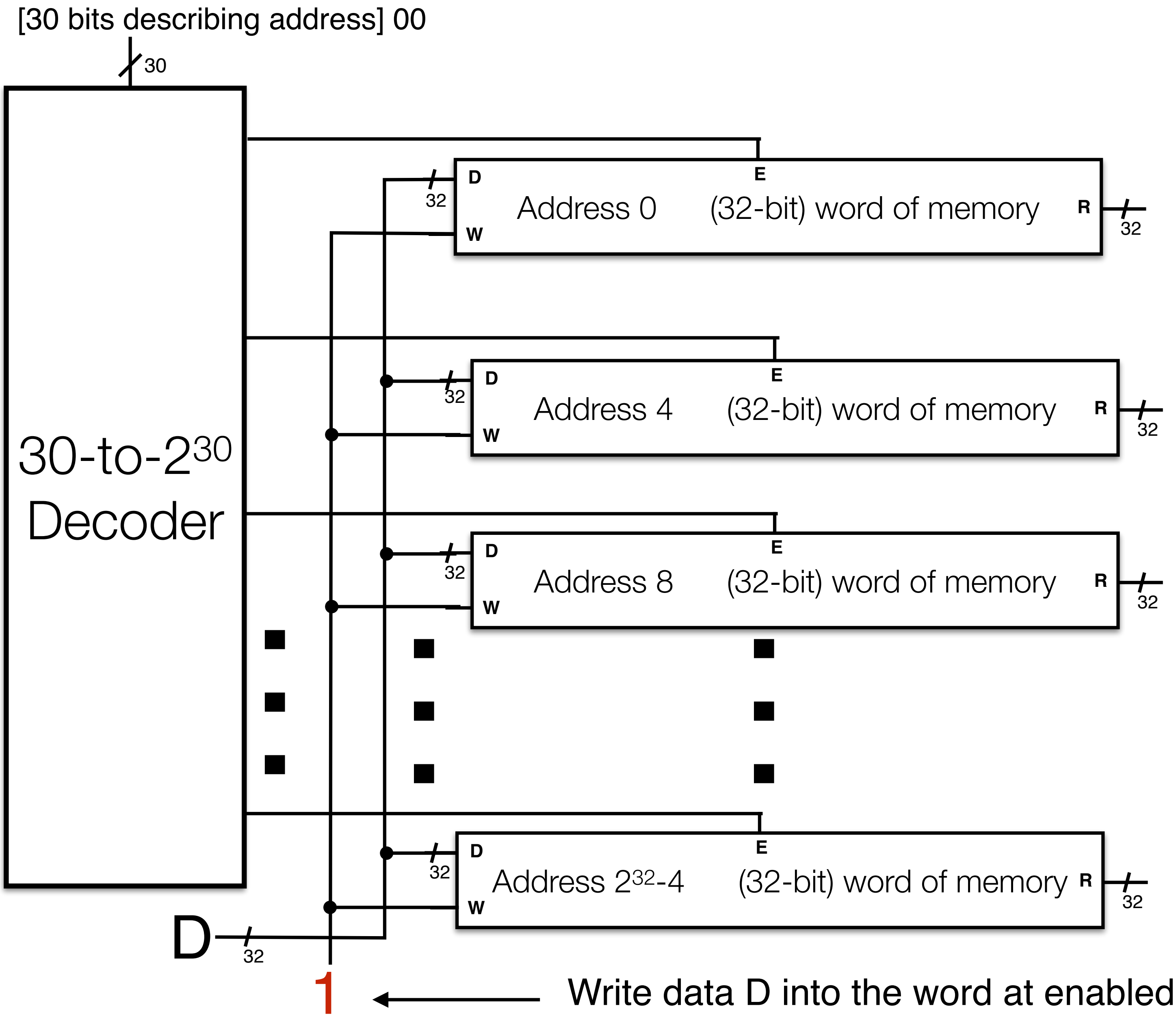
- Each address is really described by 30 bits (since the 2 lowest-order bits of the 32-bit address are 00)
- Data Input to memory, D, fed into word at each address
- Write Enable, W, fed into word at each addresss

2^{32} bytes = 2^{30} words of Addressable Memory



- Each address is really described by 30 bits (since the 2 lowest-order bits of the 32-bit address are 00)
- Data Input to memory, D, fed into word at each address
- Write Enable, W, fed into word at each addresss
- Decoder used to enable only 1 word

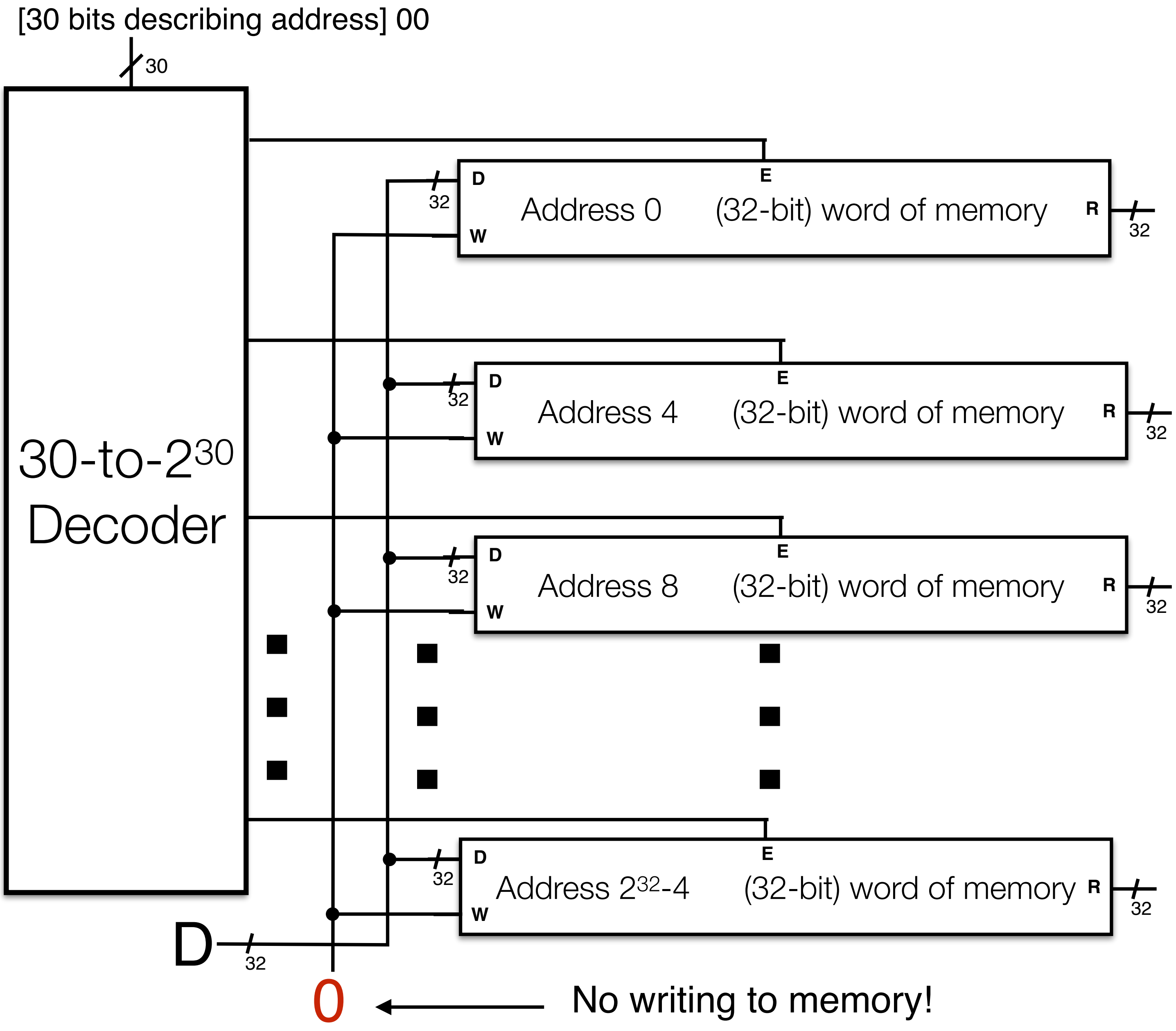
2^{32} bytes = 2^{30} words of Addressable Memory



Write data D into the word at enabled address

- Each address is really described by 30 bits (since the 2 lowest-order bits of the 32-bit address are 00)
- Data Input to memory, D, fed into word at each address
- Write Enable, W, fed into word at each addresss
- Decoder used to enable only 1 word

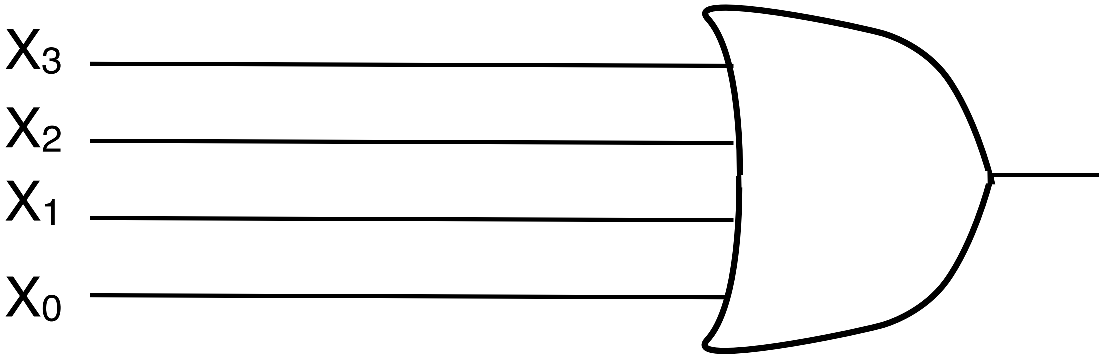
2^{32} bytes = 2^{30} words of Addressable Memory



- Each address is really described by 30 bits (since the 2 lowest-order bits of the 32-bit address are 00)
- Data Input to memory, D, fed into word at each address
- Write Enable, W, fed into word at each addresss
- Decoder used to enable only 1 word

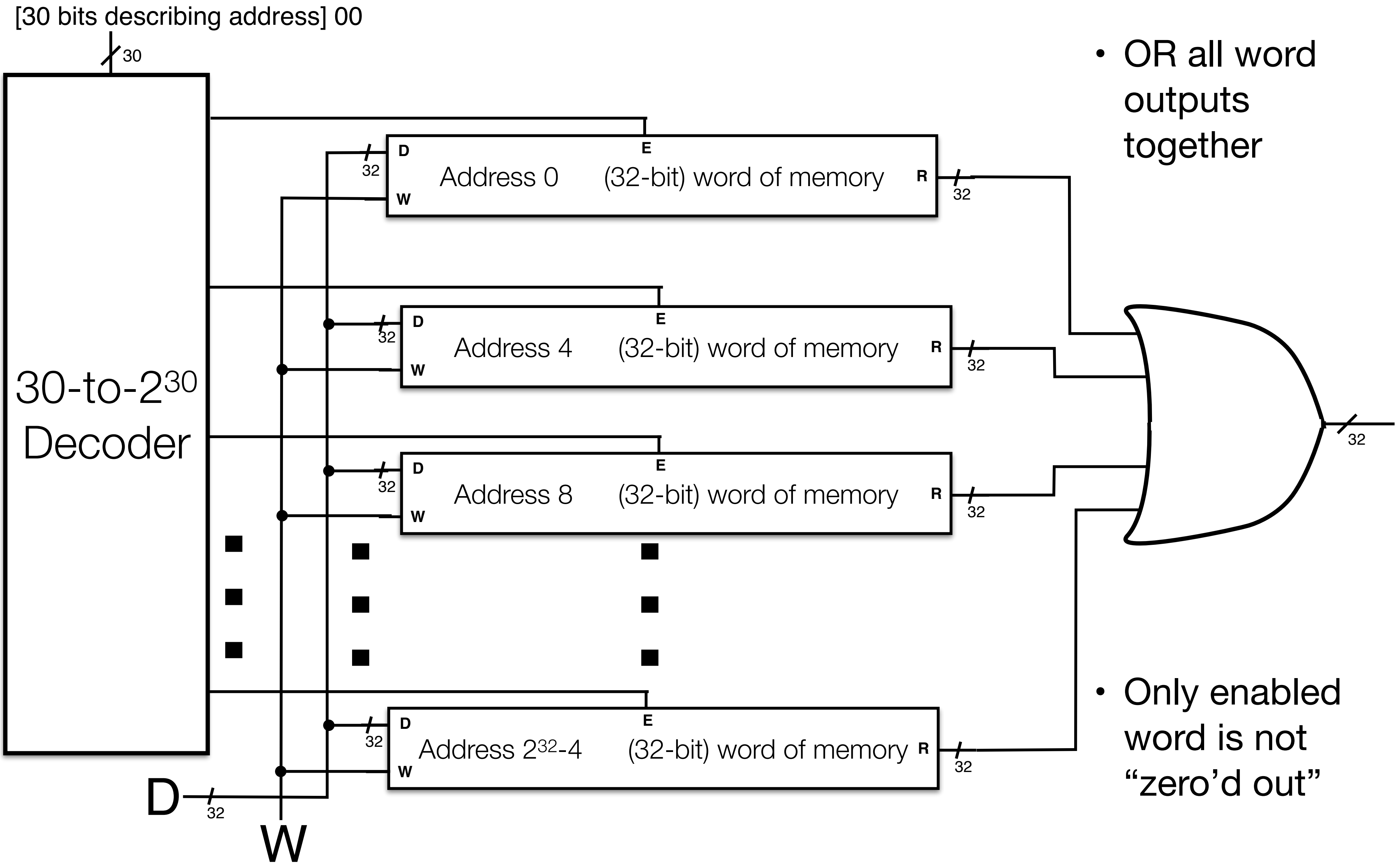
RECALL: Merge-with-known-0’s

- Suppose have m input signals
- All of them = 0 except maybe one X_j.
- Circuit to retrieve the value of the one unknown



- If all X_i=0 for i≠j, then the OR over all X_i (including X_j) = X_j

2^{32} bytes = 2^{30} words of Addressable Memory



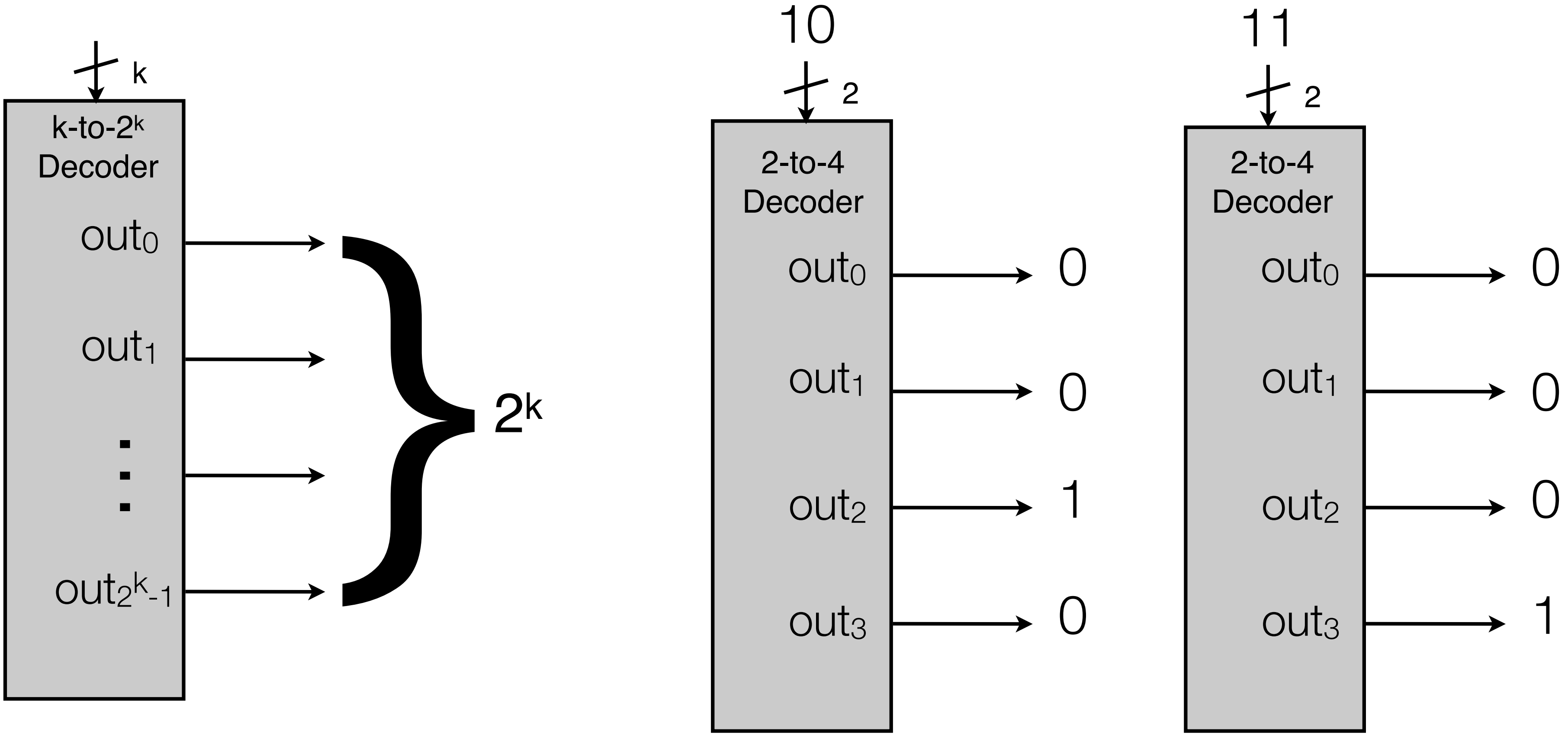
- OR all word outputs together

- Only enabled word is not “zero’d out”

Coincident Selection

Recall: Decoder

- No “DATA" inputs, just a k-bit “selector” input
- 2^k 1-bit outputs
- Selector input (i) chooses which output = 1, all other outputs = 0



Pick along 1 Dimension: Use a decoder

Suppose have 2^k “items” and want to choose 1
(here k=3)

Pick along 1 Dimension: Use a decoder

000

001

010

011

100

101

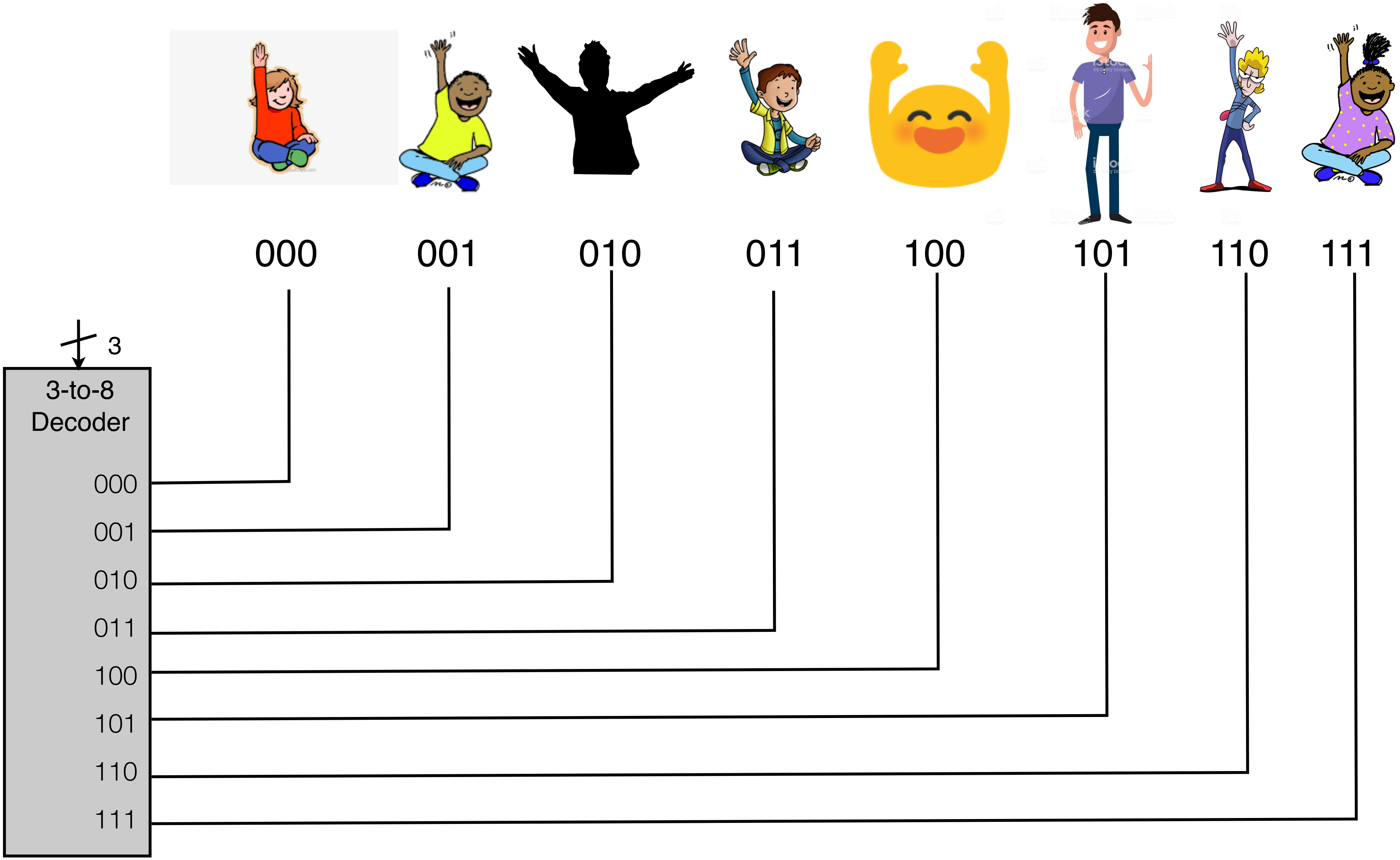
110

111

Suppose have 2^k “items” and want to choose 1
(here k=3)

Assign a number to each “item”

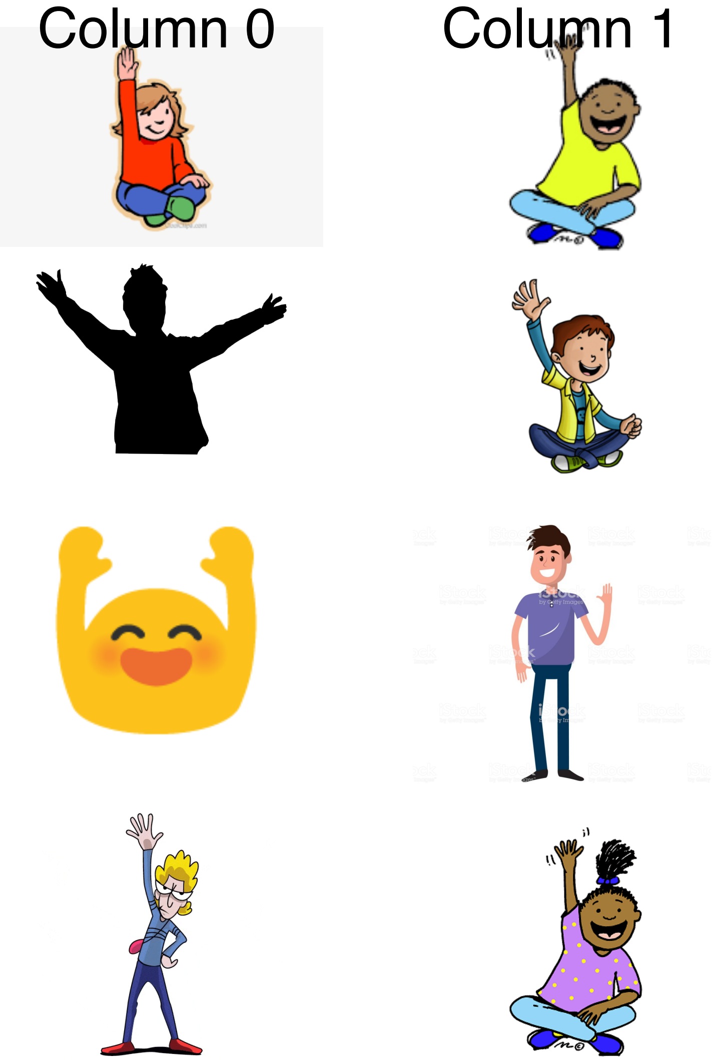
Pick along 1 Dimension: Use a decoder



1-Dimensional Decoding Summary

- To enable 1 of 2^k addresses:
  - 1 Decoder
  - k selector “pins” on decoder
  - 2^k output “pins”

Pick along 2 Dimensions: Use 2 decoders



Column 0

Column 1

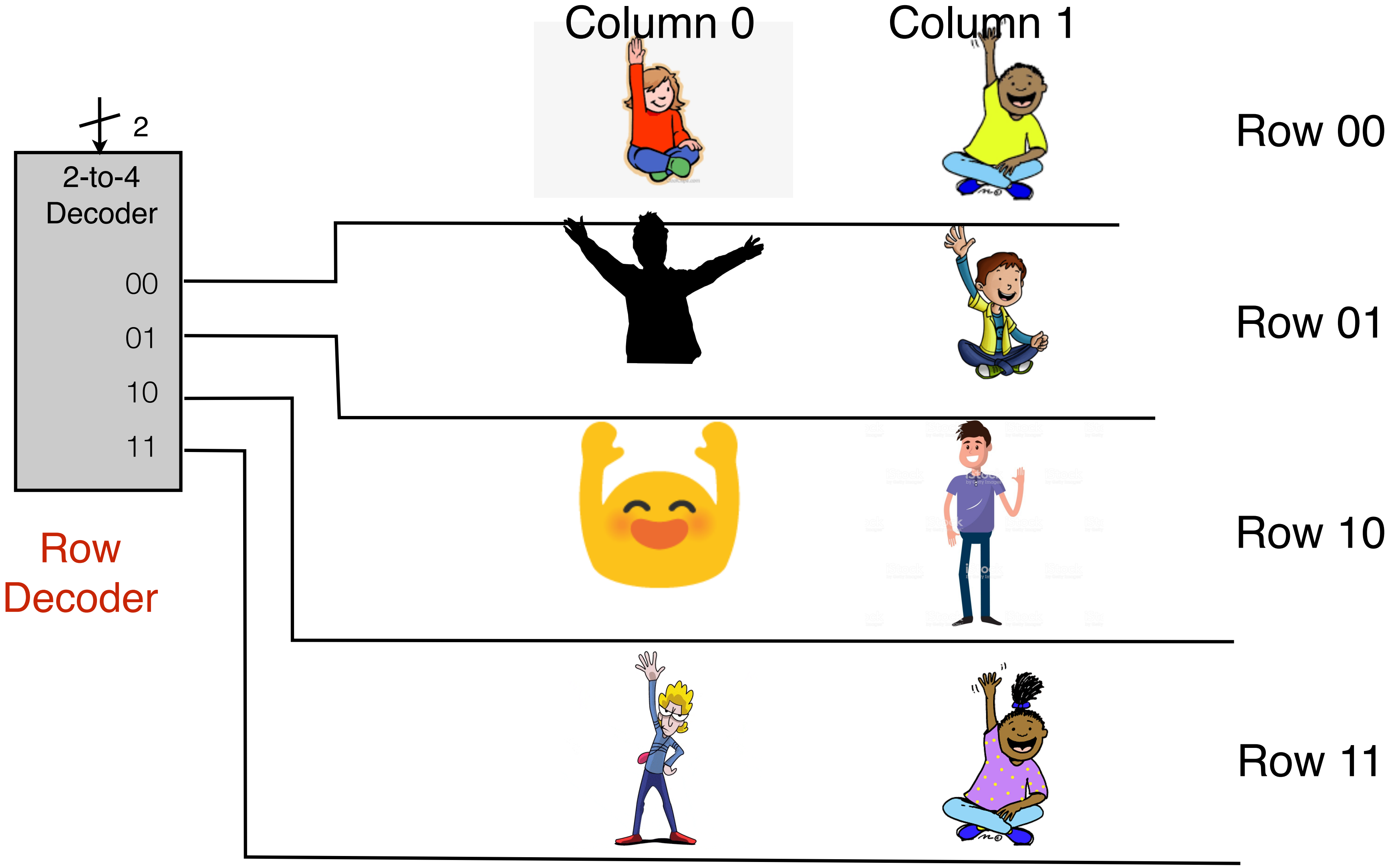
Row 00

Row 01

Row 10

Row 11

Pick along 2 Dimensions: Use 2 decoders



Column 0

Column 1

2

2-to-4 Decoder

00

01

10

11

Row 00

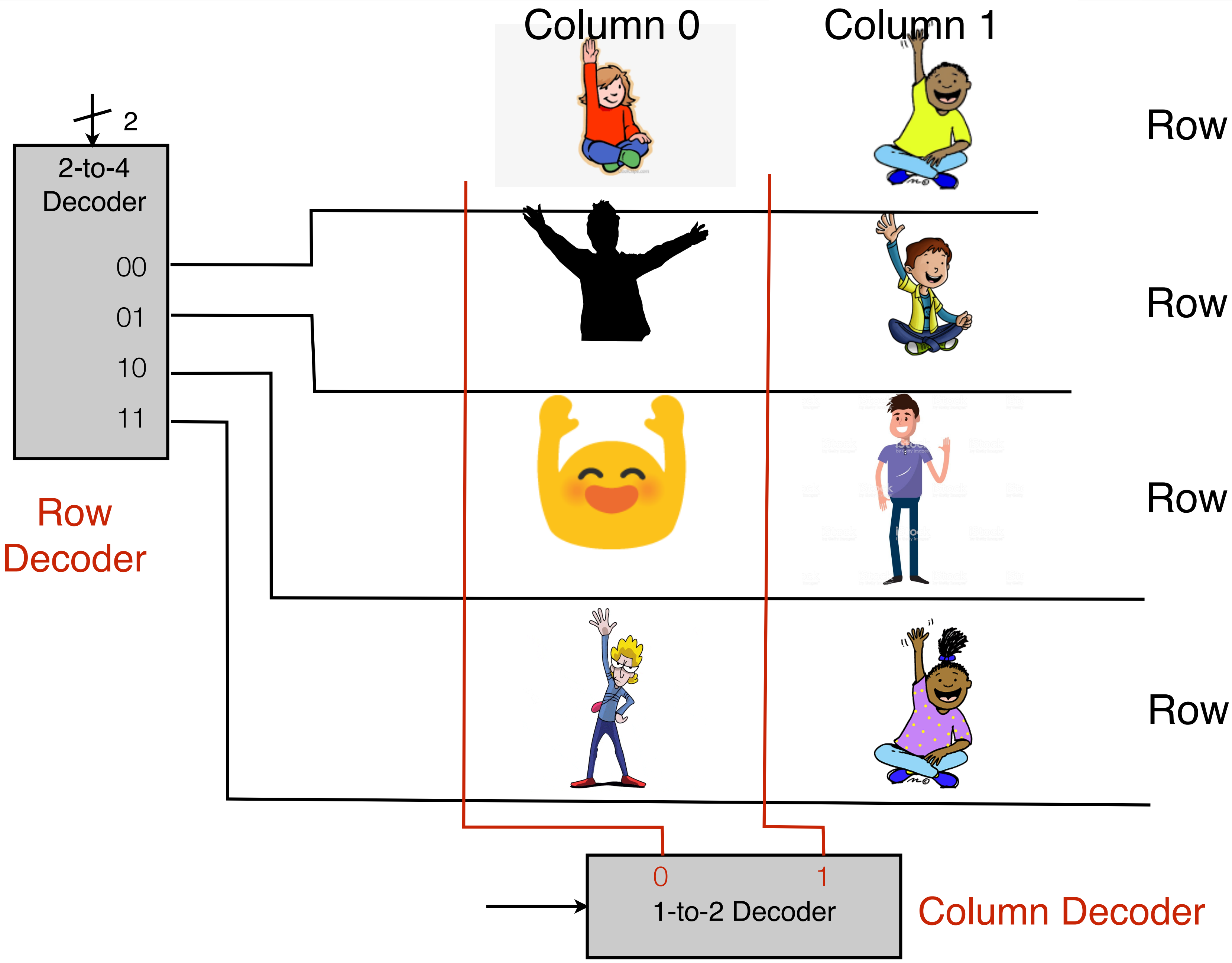
Row 01

Row 10

Row 11

Row Decoder

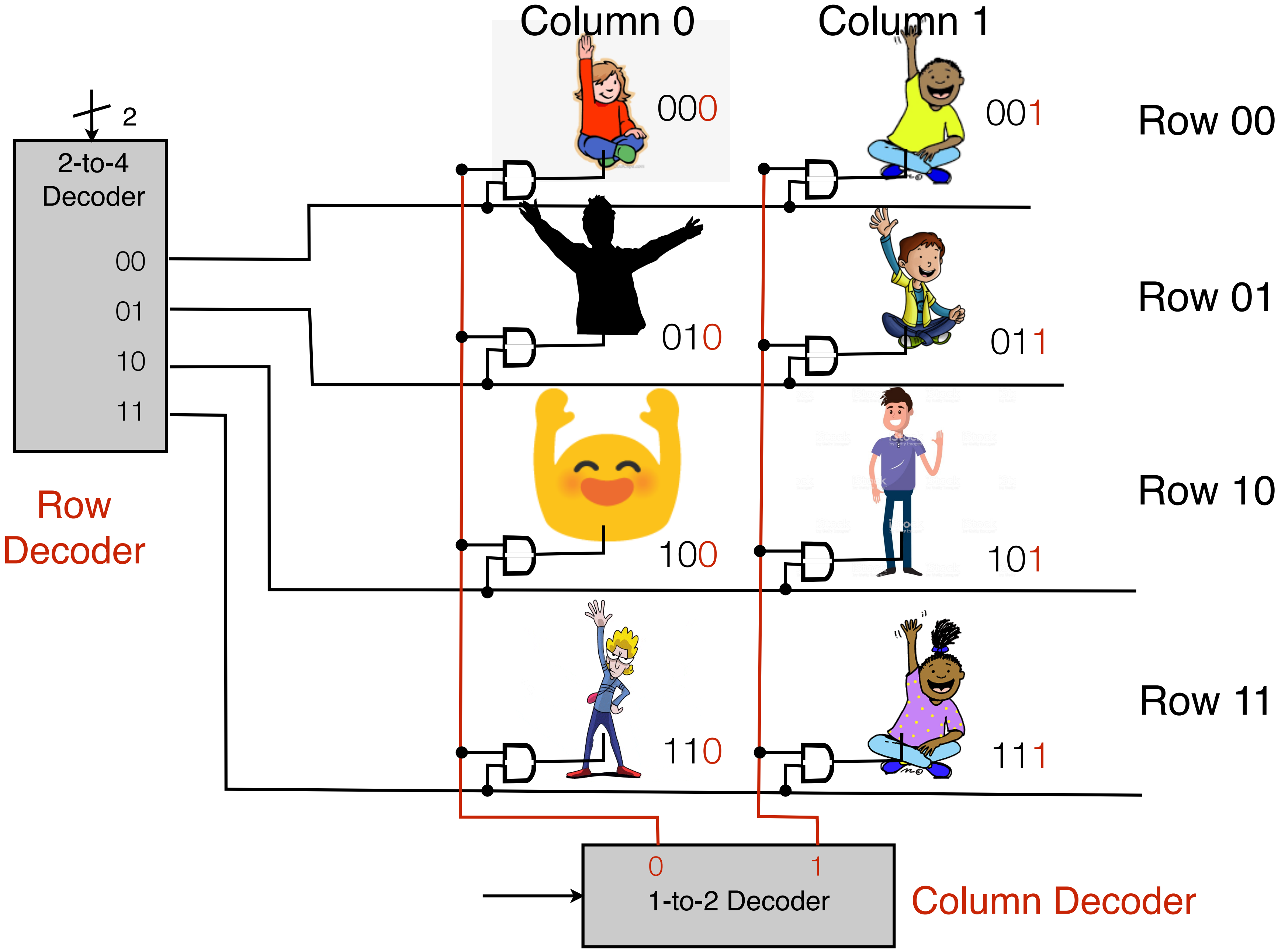
Pick along 2 Dimensions: Use 2 decoders



Row Decoder

Column Decoder

Pick along 2 Dimensions: Use 2 decoders



1-D v. 2-D Decoding Comparison Summary

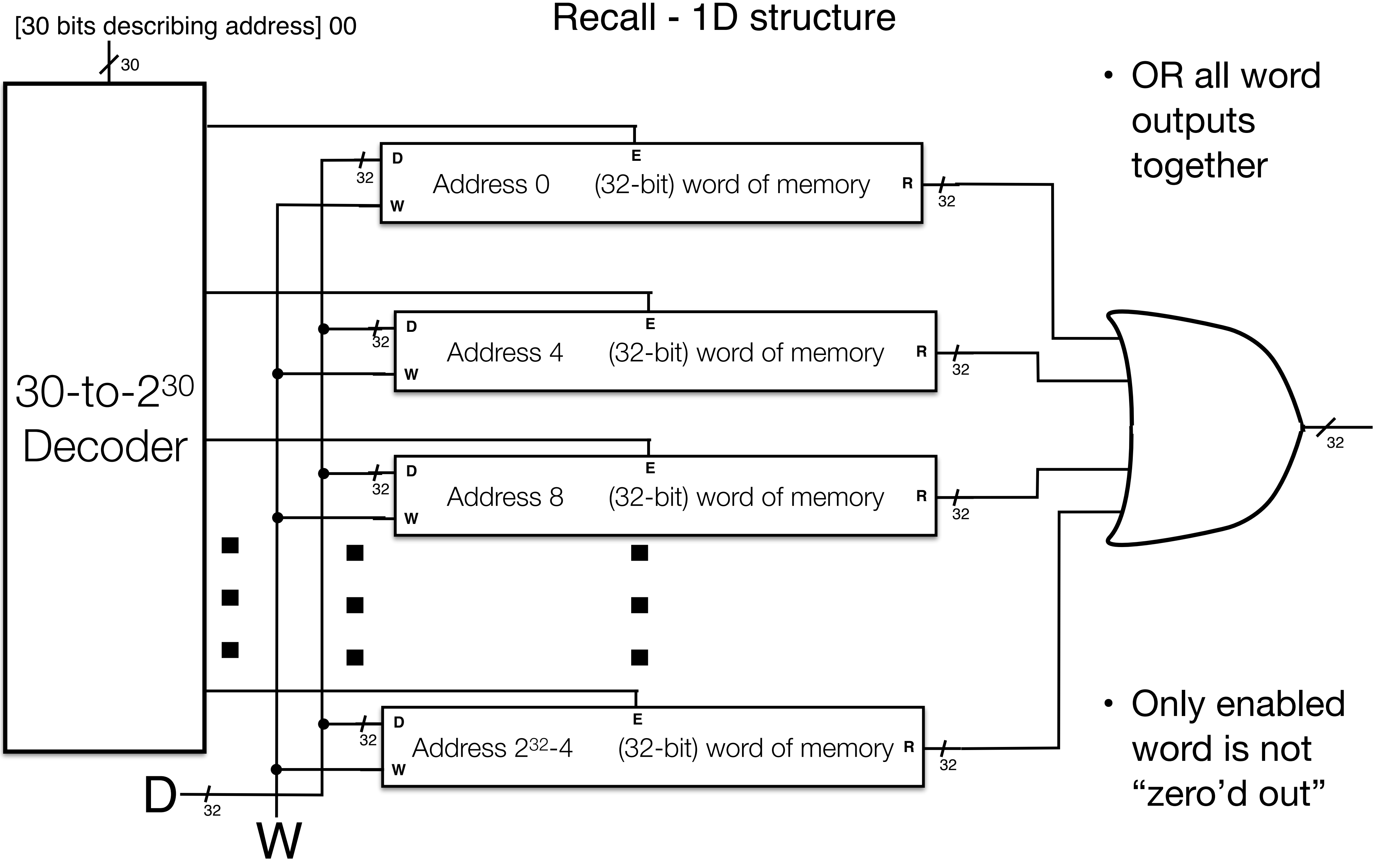
- 1D
- To enable 1 of 2^k addresses:
  - 1 Decoder
  - k selector “pins” on decoder
  - 2^k output “pins”
  - e.g., k=10, 1024 output pins

- 2D
- To enable 1 of 2^k addresses:
  - 2 Decoders
  - k_1 and k_2 selector “pins” on decoder with $k_1+k_2=k$
  - $2^{k_1} + 2^{k_2}$ output “pins”
  - e.g., $k_1=5, k_2=5$, 64 output pins

2D has Significantly Less Decoder Logic

Modifying memory for coincident selection…

Recall - 1D structure

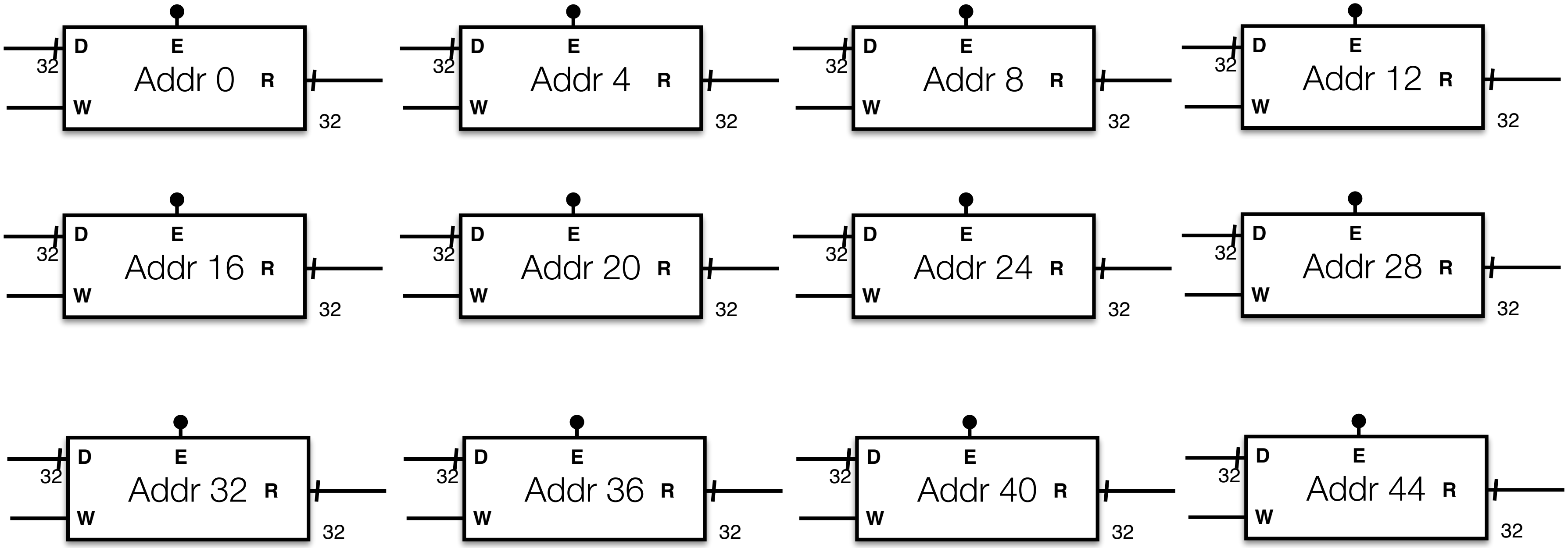


- OR all word outputs together

- Only enabled word is not “zero’d out”

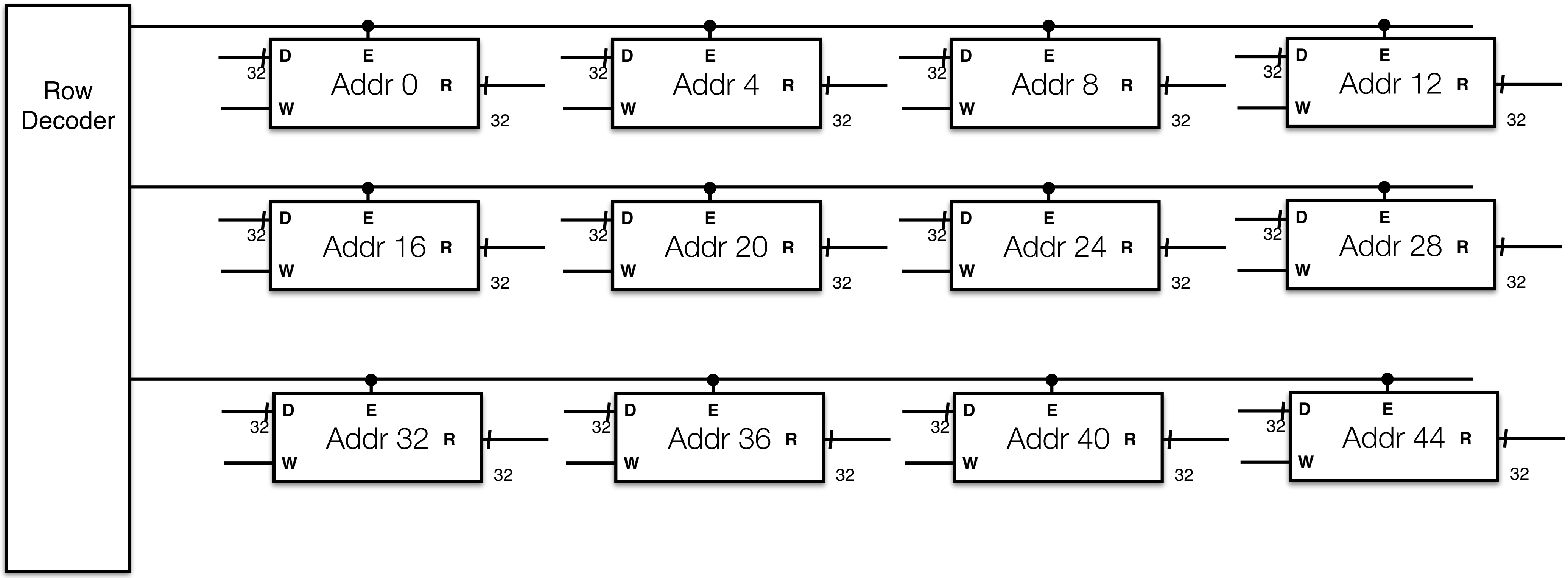
Layout words of memory into rows and columns

- Arrange memory words into 2-dimensional grid



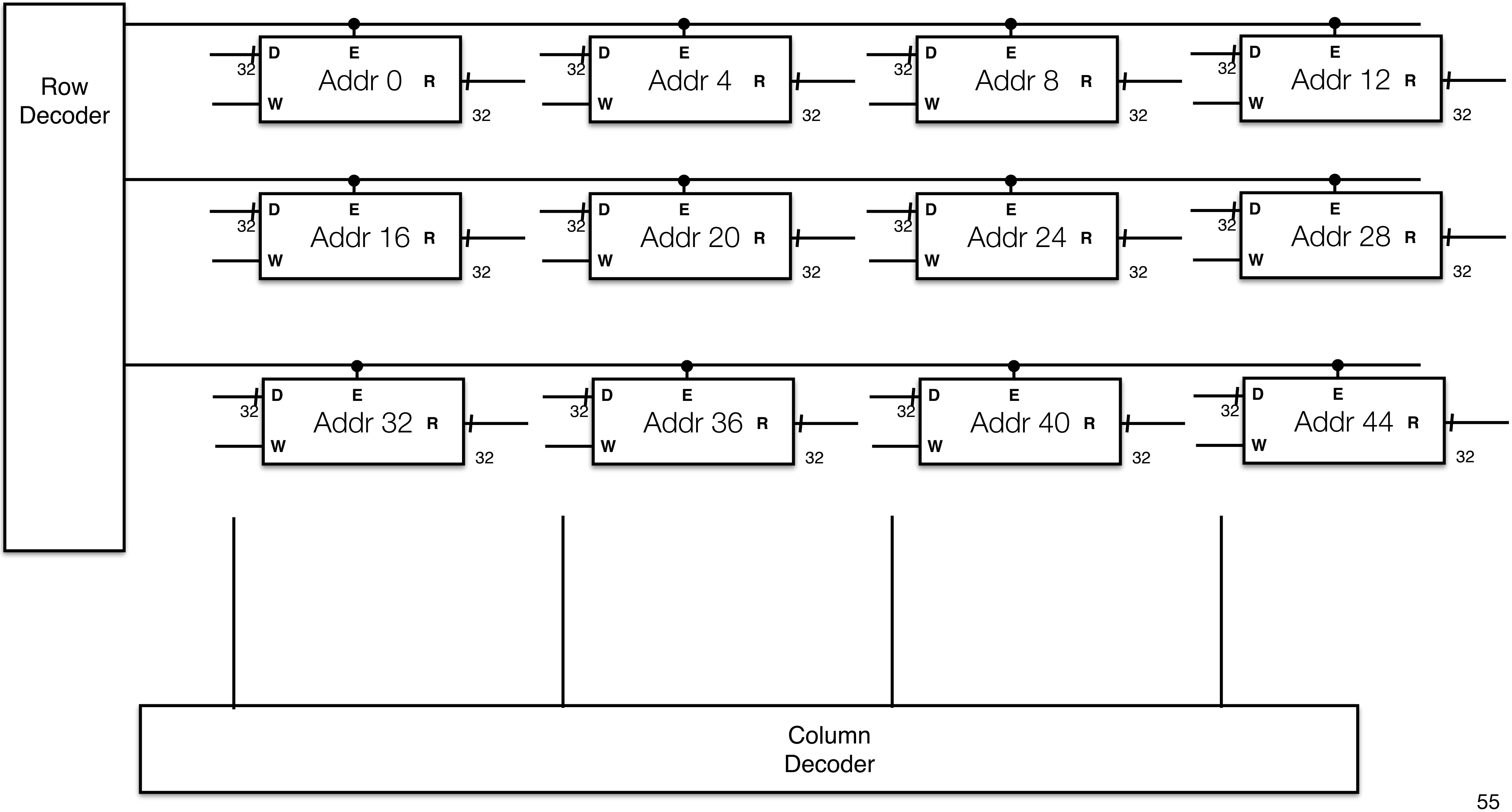
Layout words of memory into rows and columns

- Row decoder enables 1 row of addresses for use



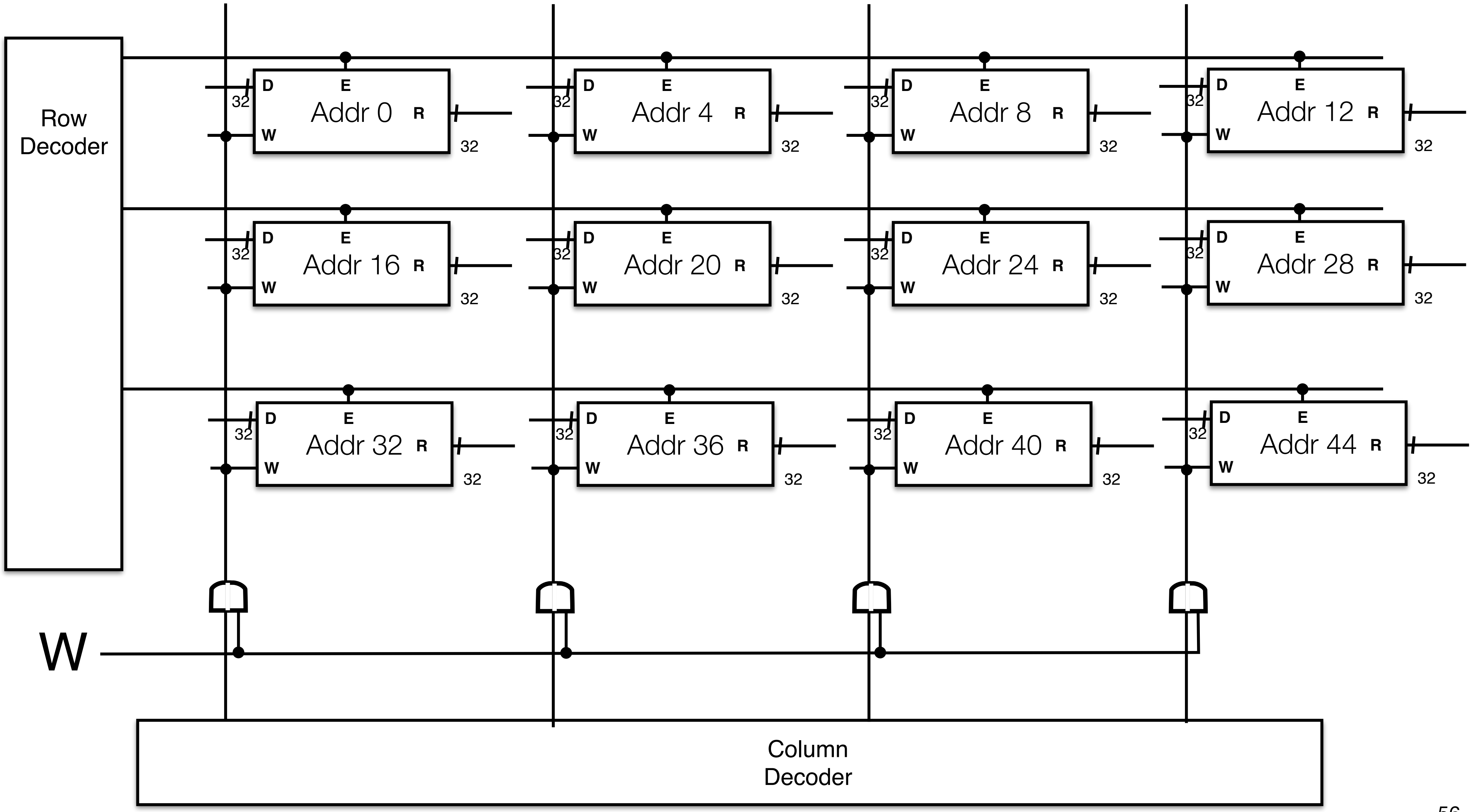
Layout words of memory into rows and columns

- Column decoder will select a column, but how?



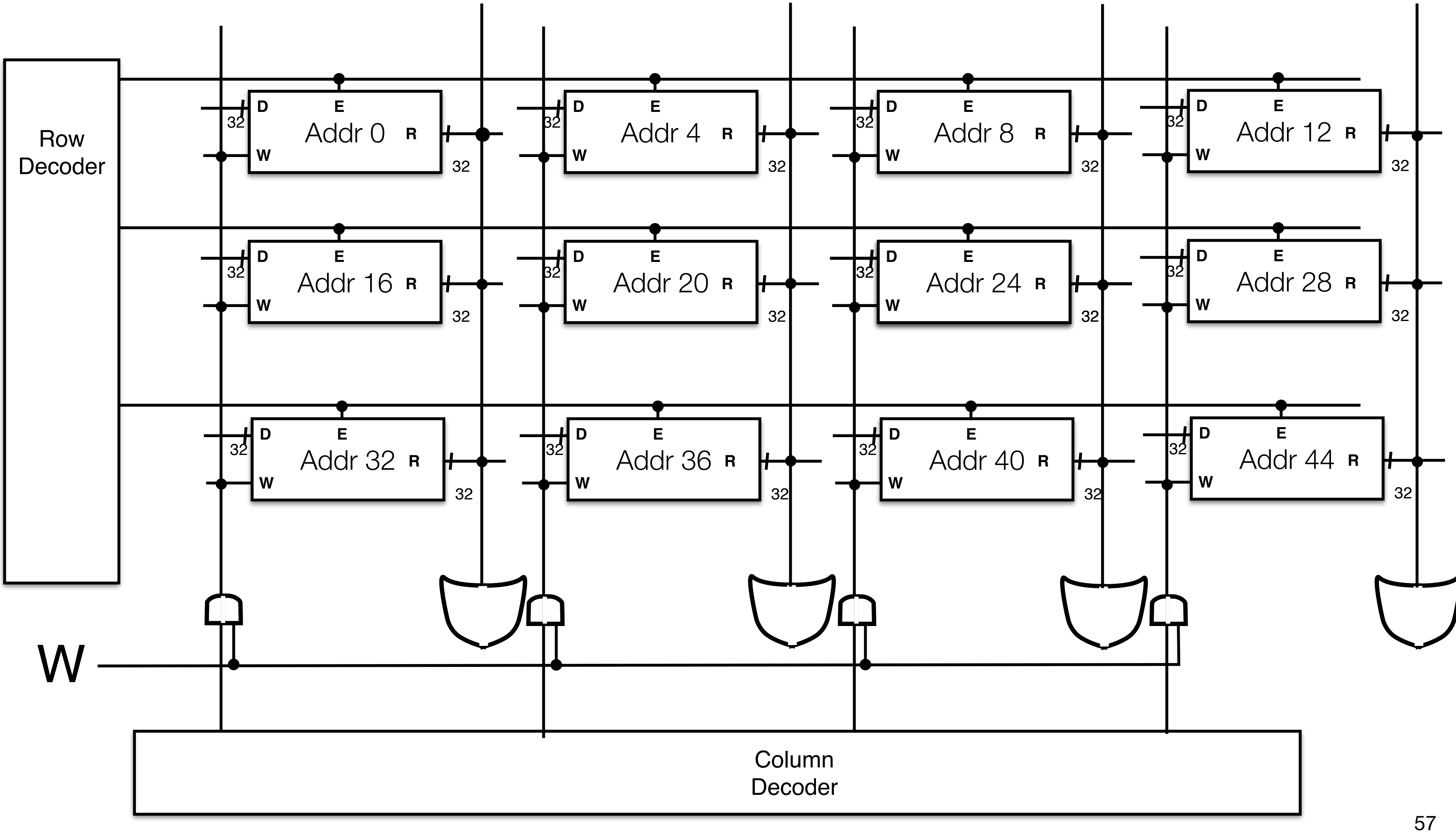
Layout words of memory into rows and columns

- For write: AND decoder output with W input



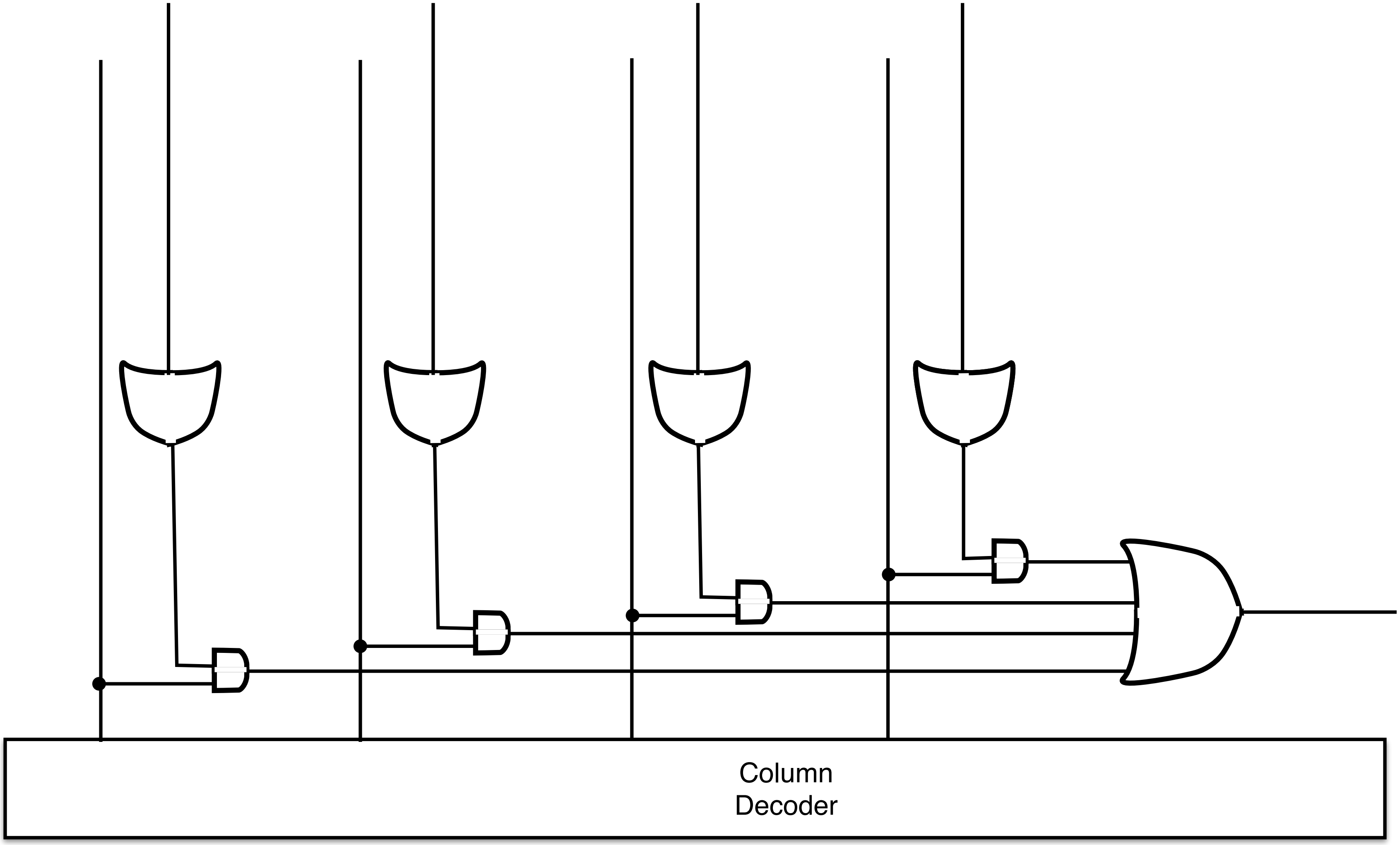
Layout words of memory into rows and columns

- For read, take output of each column (only 1 row enabled)…



Layout words of memory into rows and columns

- Choose the column to enable also



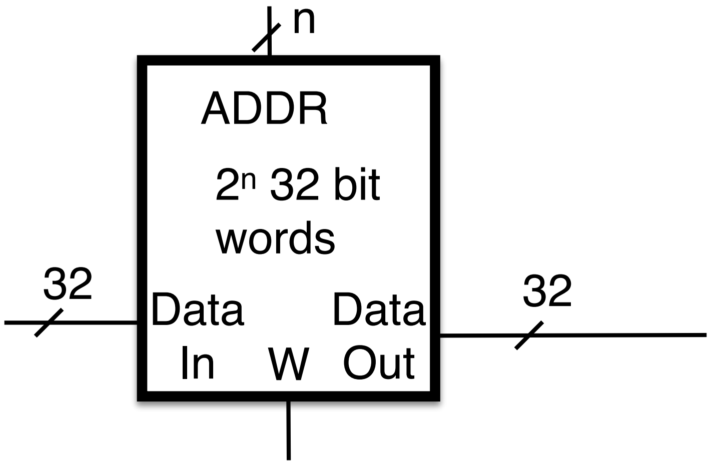
Using Multiple Memory Chips to Scale Memory

Two ways to Scale Memory

- Given memory chips with 2^n addresses and k-bit word size
    - Suppose want more addresses (e.g., 2^{n+i})
    - Suppose want larger word size (e.g., mk for some integer m>1)

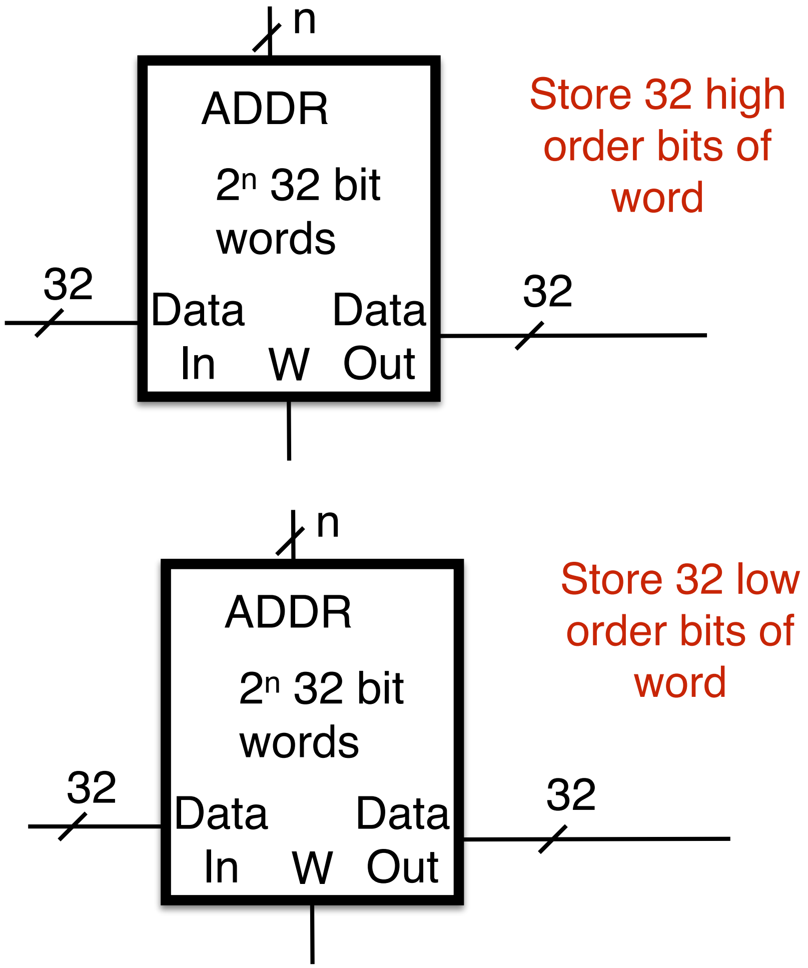
Multi-chip memories: extend wordsize

- Extend wordsize by storing full word in parallel across multiple chips
- e.g., build 64-bit memory storing 2^n words from two 32-bit 2^n-word chips



Multi-chip memories: extend wordsize

- Extend wordsize by storing full word in parallel across multiple chips
- e.g., build 64-bit memory storing 2^n words from two 32-bit 2^n-word chips

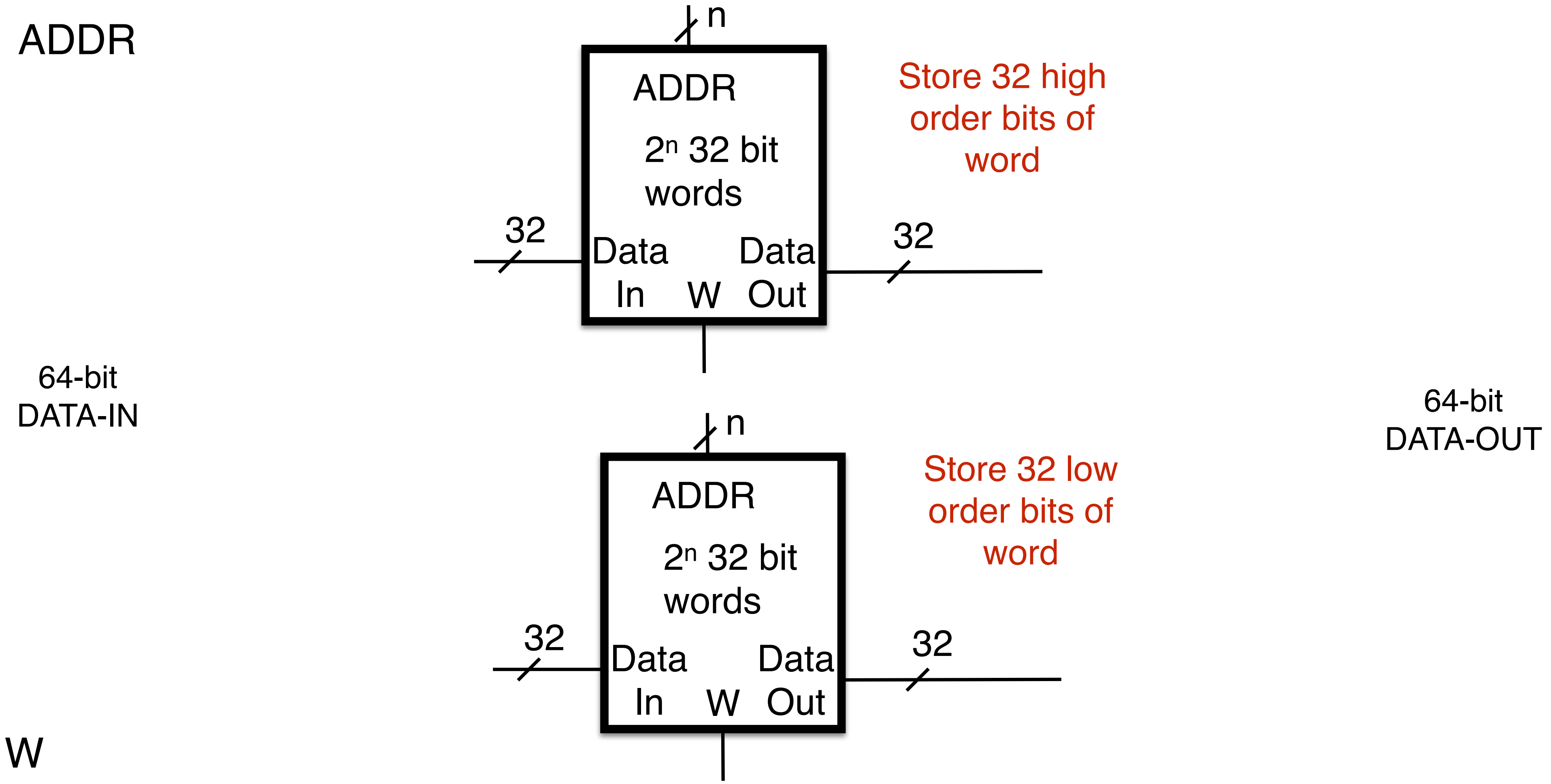


Store 32 high order bits of word

Store 32 low order bits of word

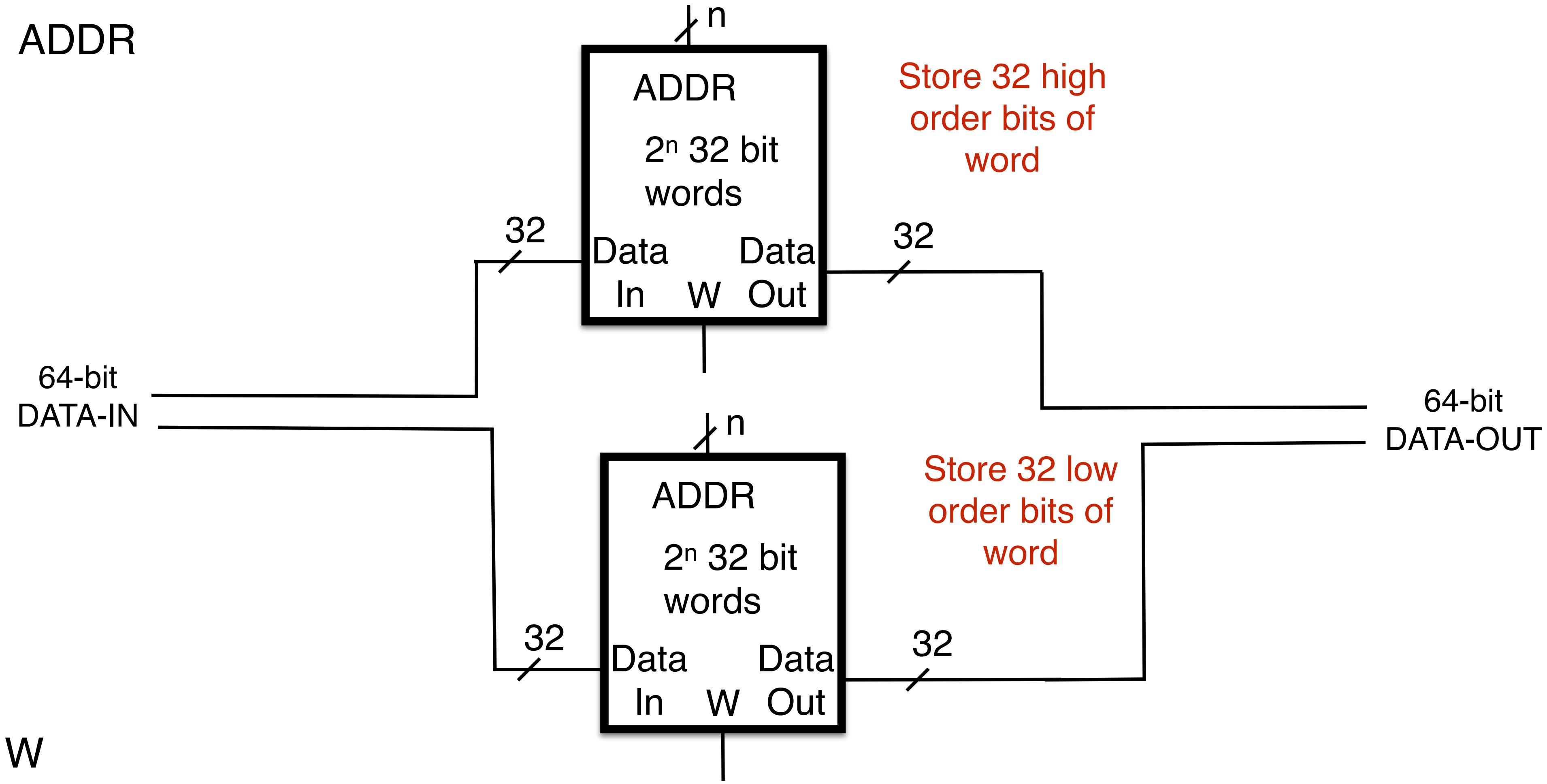
Multi-chip memories: extend wordsize

- Extend wordsize by storing full word in parallel across multiple chips
- e.g., build 64-bit memory storing 2^n words from two 32-bit 2^n-word chips



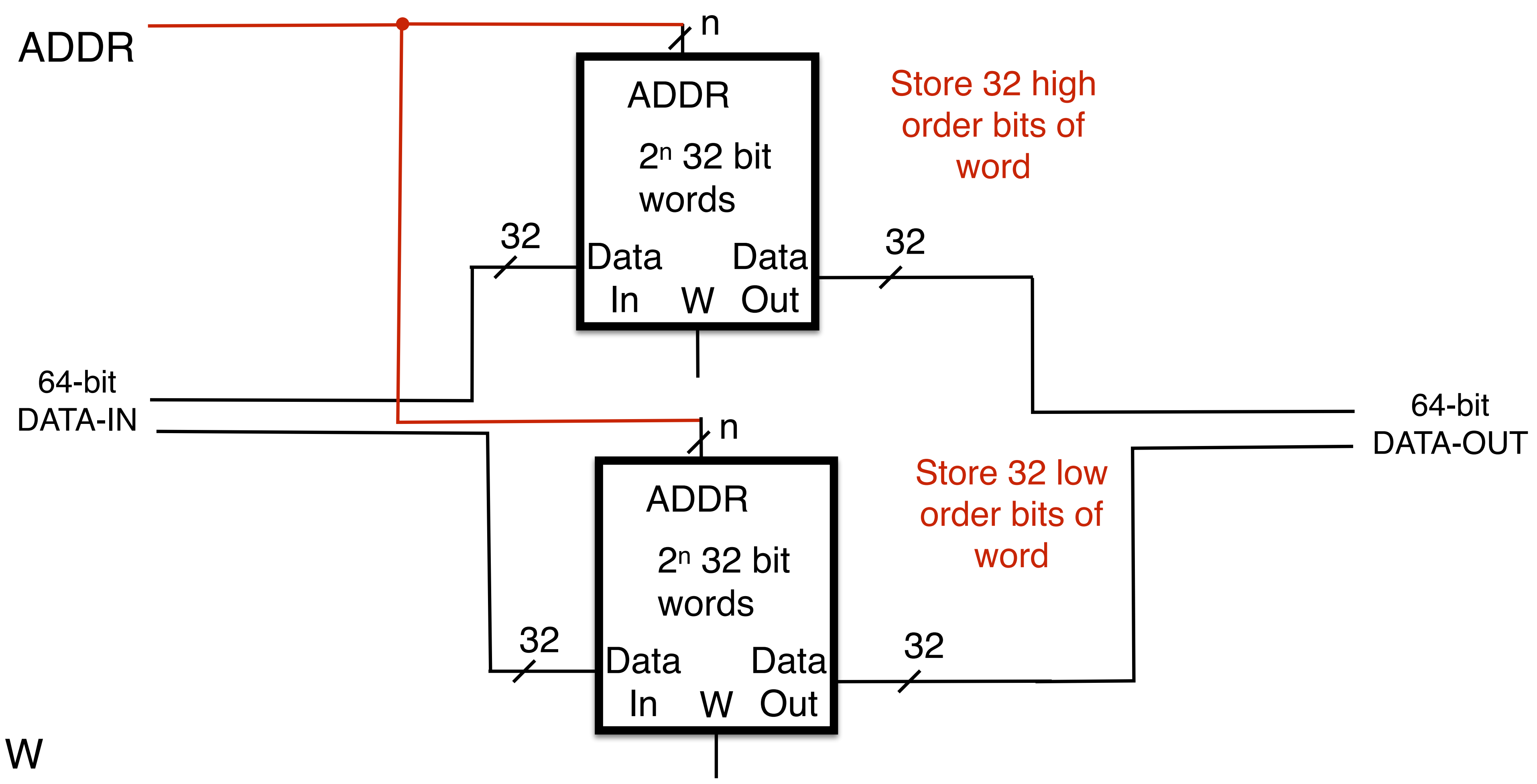
Multi-chip memories: extend wordsize

- Extend wordsize by storing full word in parallel across multiple chips
- e.g., build 64-bit memory storing 2^n words from two 32-bit 2^n-word chips



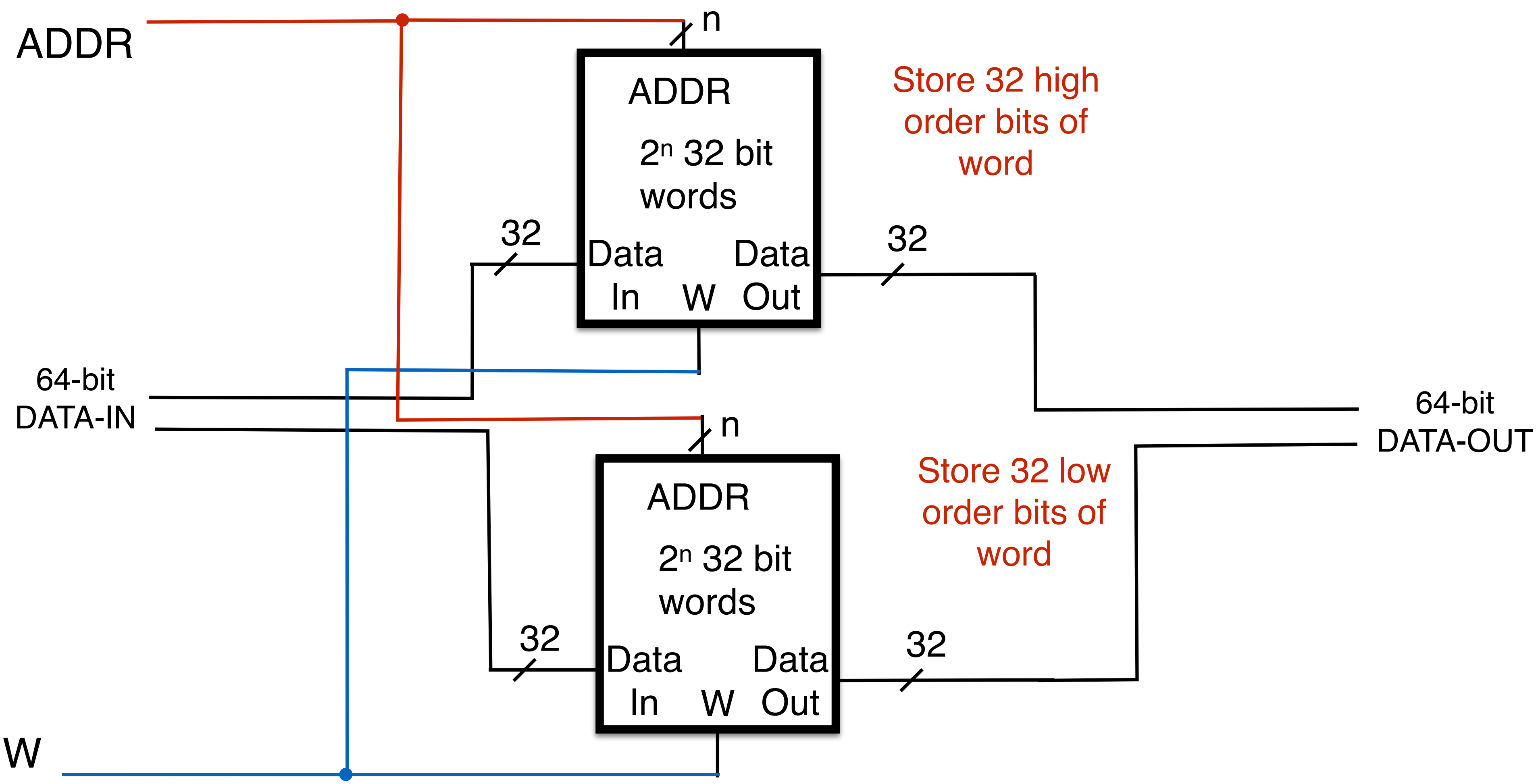
Multi-chip memories: extend wordsize

- Extend wordsize by storing full word in parallel across multiple chips
- e.g., build 64-bit memory storing 2^n words from two 32-bit 2^n-word chips



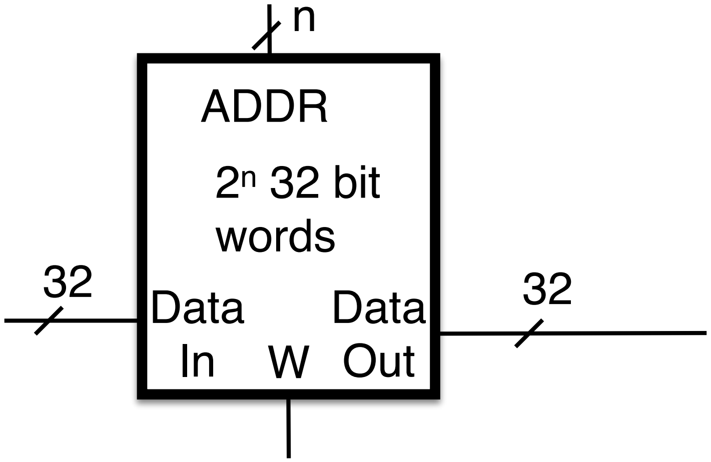
Multi-chip memories: extend wordsize

- Extend wordsize by storing full word in parallel across multiple chips
- e.g., build 64-bit memory storing 2^n words from two 32-bit 2^n-word chips



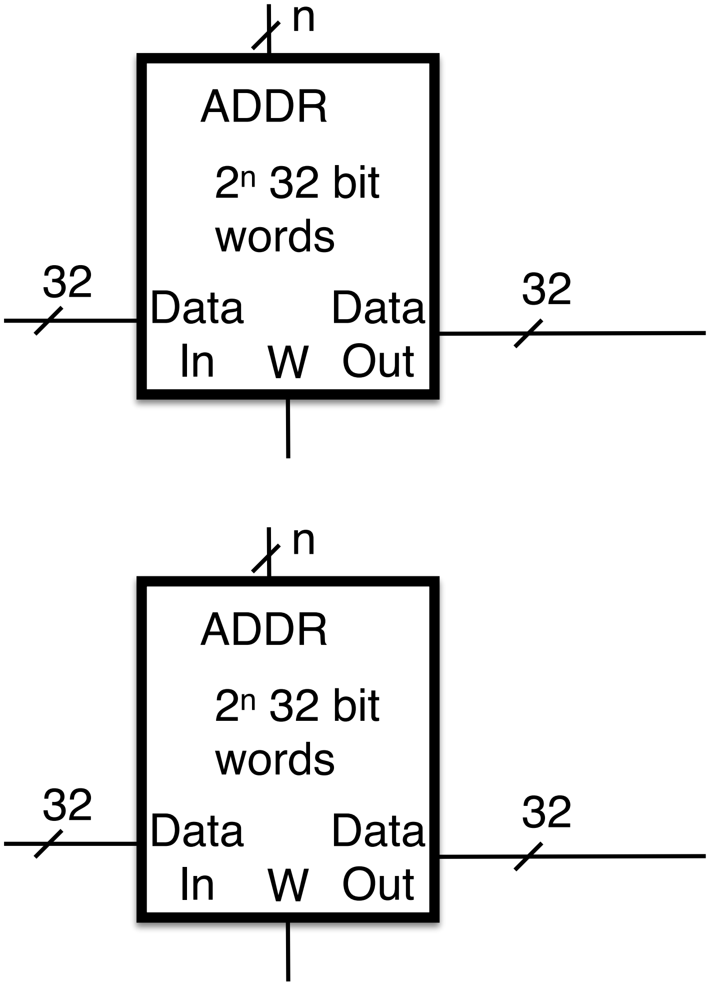
Multi-chip memories: extend address space

- Suppose each chip stores 2^n 32-bit words (has 2^n n-bit addresses)



Multi-chip memories: extend address space

- Suppose each chip stores 2^n 32-bit words (has 2^n n-bit addresses)
- 2 of these chips can store 2^{n+1} 32-bit worse (2^{n+1} (n+1)-bit addresses)

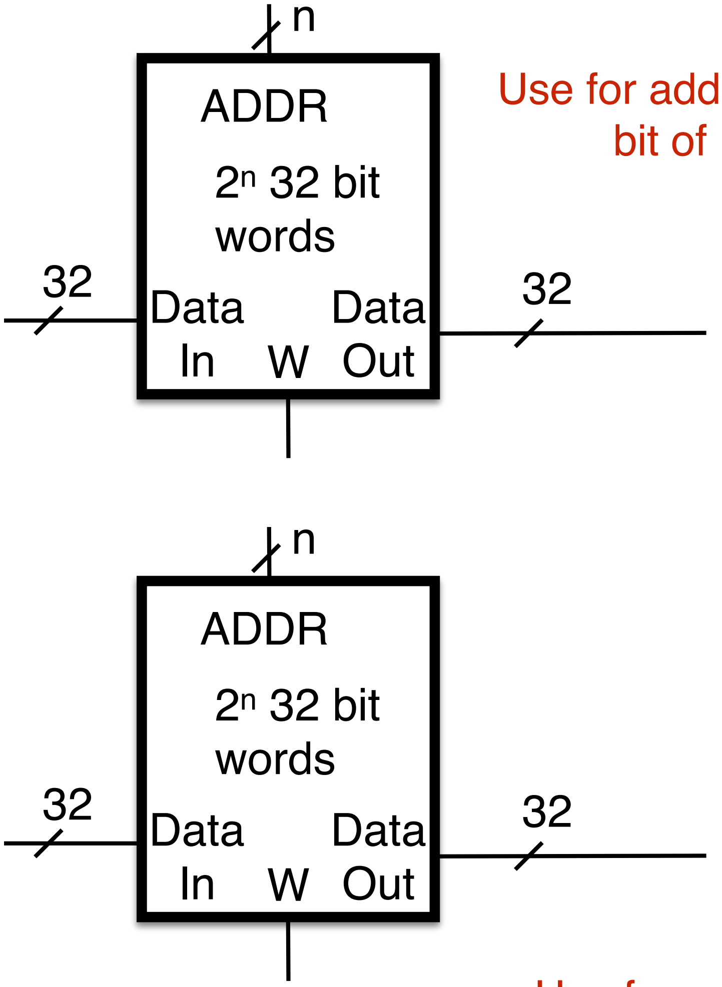


Multi-chip memories: extend address space

- Suppose each chip stores 2^n 32-bit words (has 2^n n-bit addresses)
- 2 of these chips can store 2^{n+1} 32-bit worse (2^{n+1} (n+1)-bit addresses)

n+1 bit address

$A_n \;\; A_{n-1} \ldots A_1 A_0$

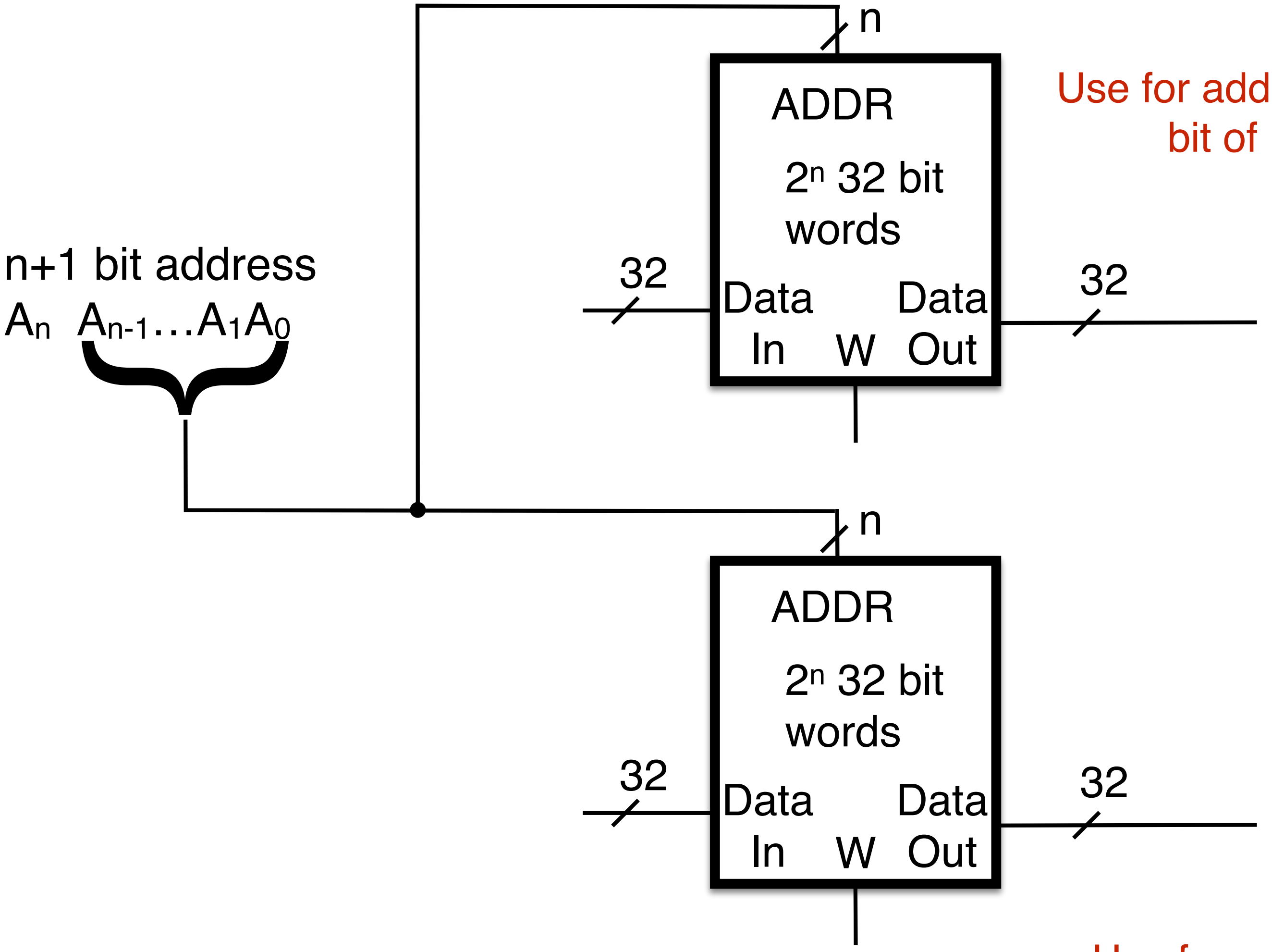


Use for addresses where highest order bit of (n+1-bit) address is 0

Use for addresses where highest order bit of (n+1-bit) address is 1

Multi-chip memories: extend address space

- Suppose each chip stores 2^n 32-bit words (has 2^n n-bit addresses)
- 2 of these chips can store 2^{n+1} 32-bit worse (2^{n+1} (n+1)-bit addresses)

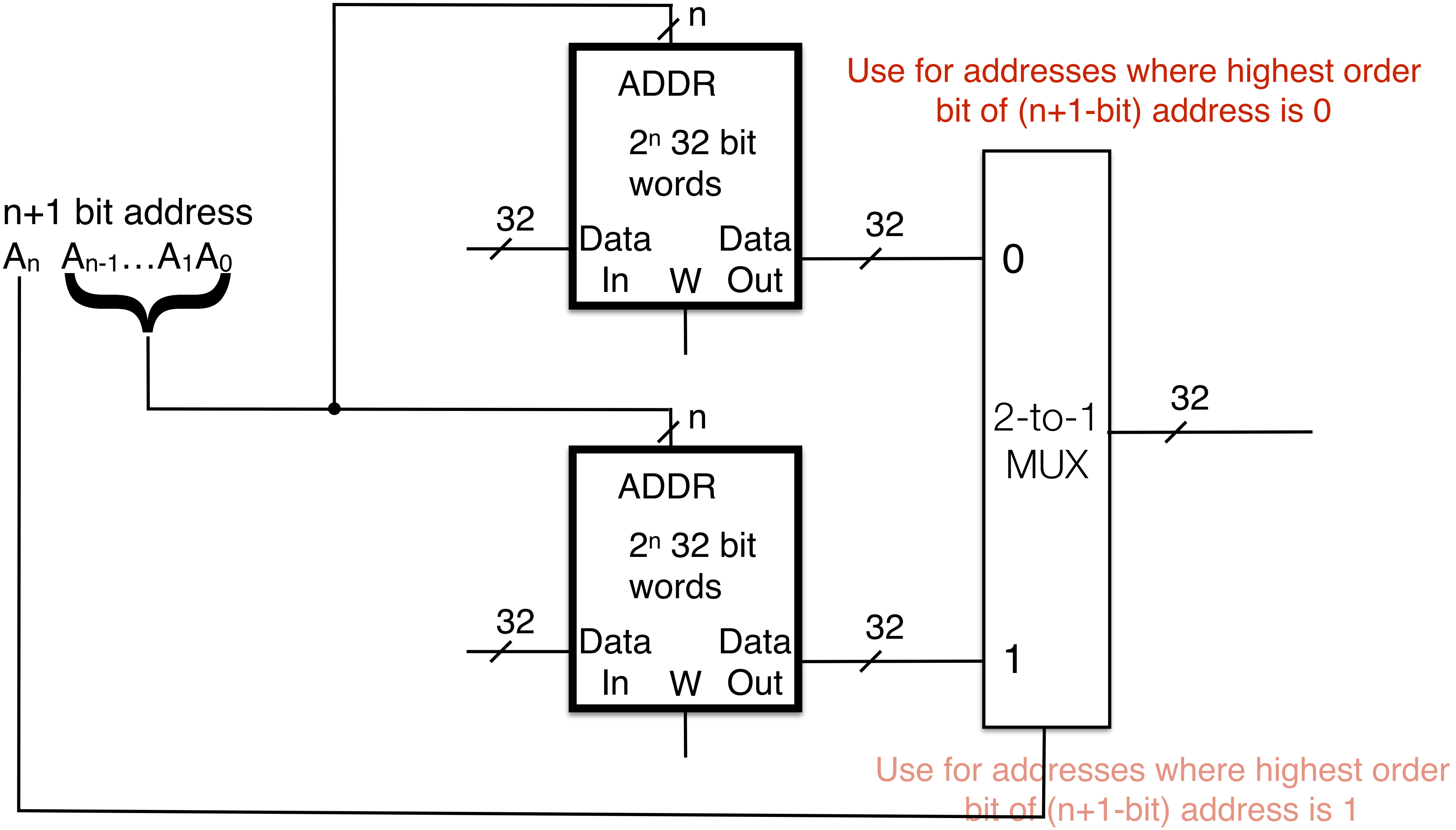


Use for addresses where highest order bit of (n+1-bit) address is 0

Use for addresses where highest order bit of (n+1-bit) address is 1

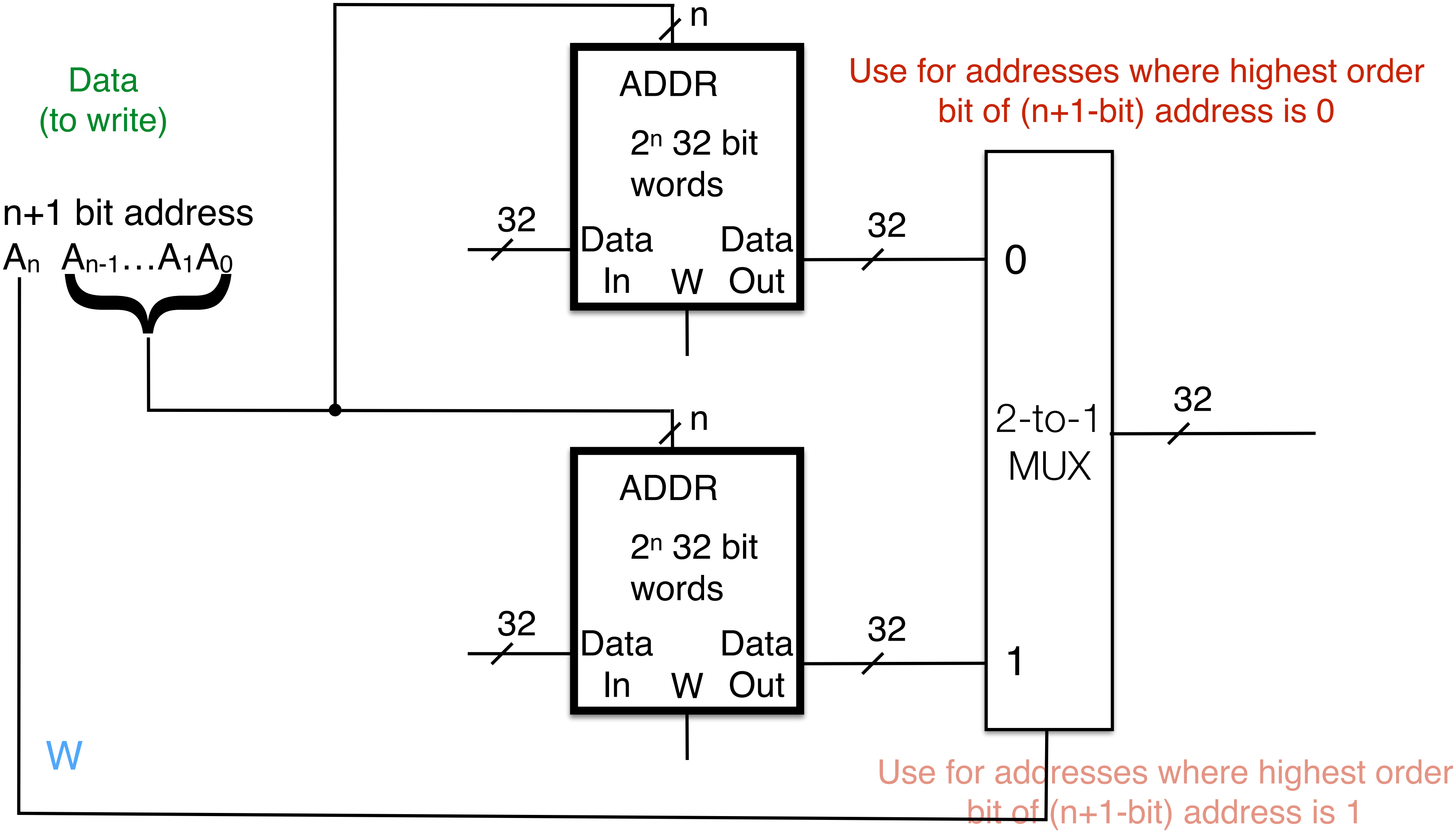
Multi-chip memories: extend address space

- Suppose each chip stores 2^n 32-bit words (has 2^n n-bit addresses)
- 2 of these chips can store 2^{n+1} 32-bit worse (2^{n+1} (n+1)-bit addresses)


n+1 bit address
A_n $A_{n-1}\ldots A_1A_0$
n
ADDR
2^n 32 bit words
32
Data In
W
Data Out
32
0
Use for addresses where highest order bit of (n+1-bit) address is 0
n
ADDR
2^n 32 bit words
32
Data In
W
Data Out
32
1
2-to-1 MUX
32
Use for addresses where highest order bit of (n+1-bit) address is 1

Multi-chip memories: extend address space

- Suppose each chip stores 2^n 32-bit words (has 2^n n-bit addresses)
- 2 of these chips can store 2^{n+1} 32-bit worse (2^{n+1} (n+1)-bit addresses)

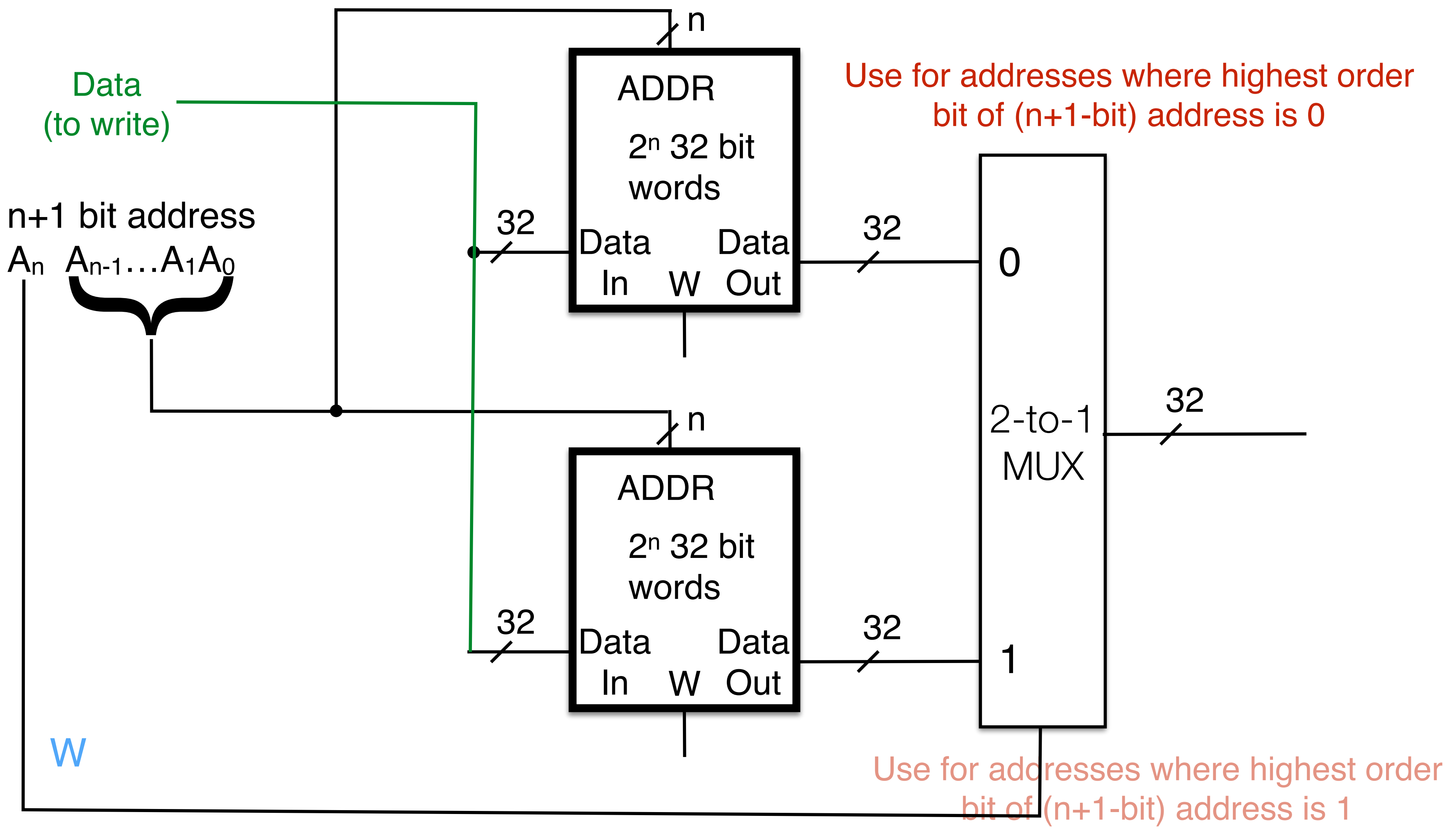


Use for addresses where highest order bit of (n+1-bit) address is 0

Use for addresses where highest order bit of (n+1-bit) address is 1

Multi-chip memories: extend address space

- Suppose each chip stores 2^n 32-bit words (has 2^n n-bit addresses)
- 2 of these chips can store 2^{n+1} 32-bit worse (2^{n+1} (n+1)-bit addresses)

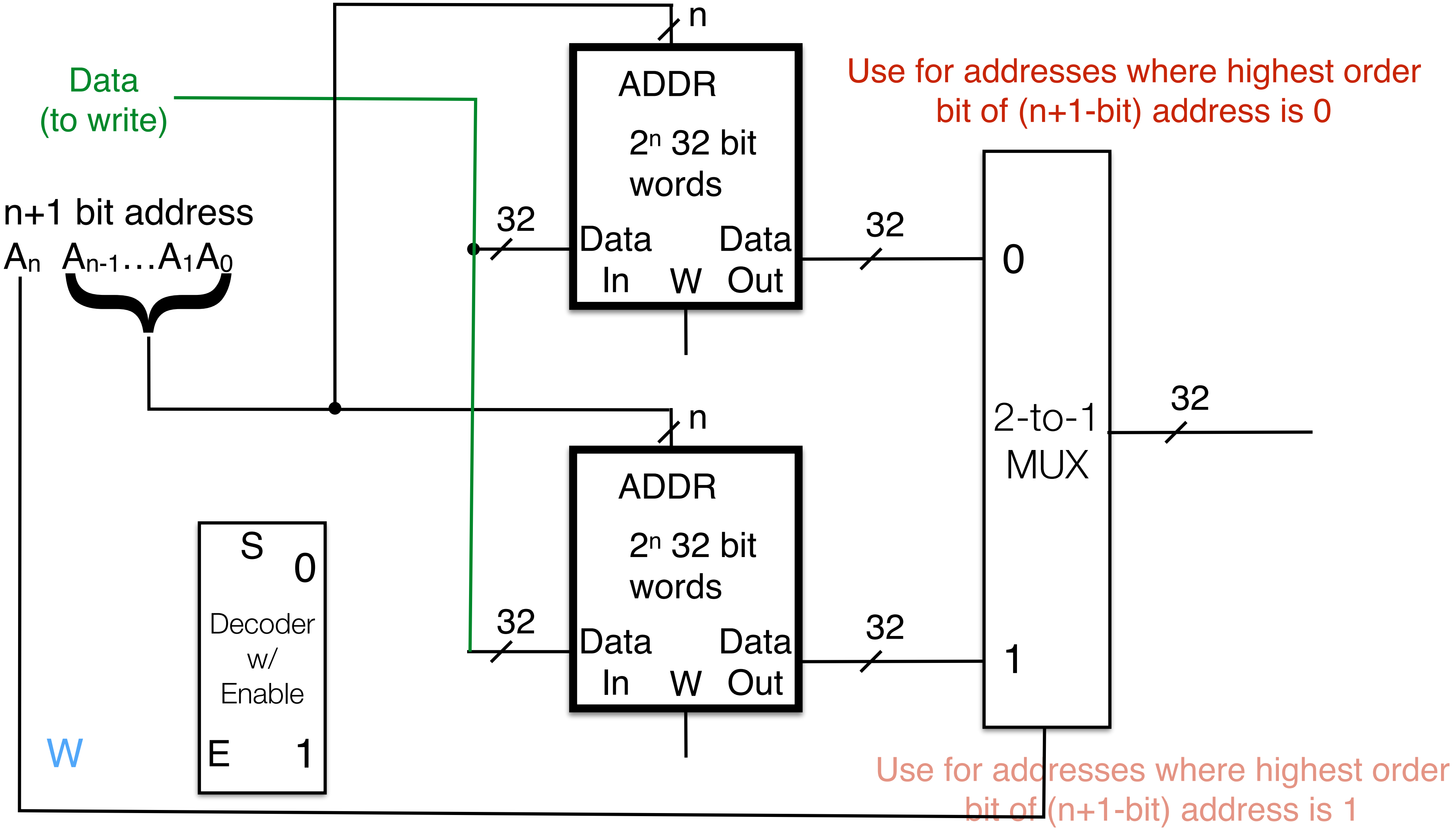


Use for addresses where highest order bit of (n+1-bit) address is 0

Use for addresses where highest order bit of (n+1-bit) address is 1

Multi-chip memories: extend address space

- Suppose each chip stores 2^n 32-bit words (has 2^n n-bit addresses)
- 2 of these chips can store 2^{n+1} 32-bit worse (2^{n+1} (n+1)-bit addresses)

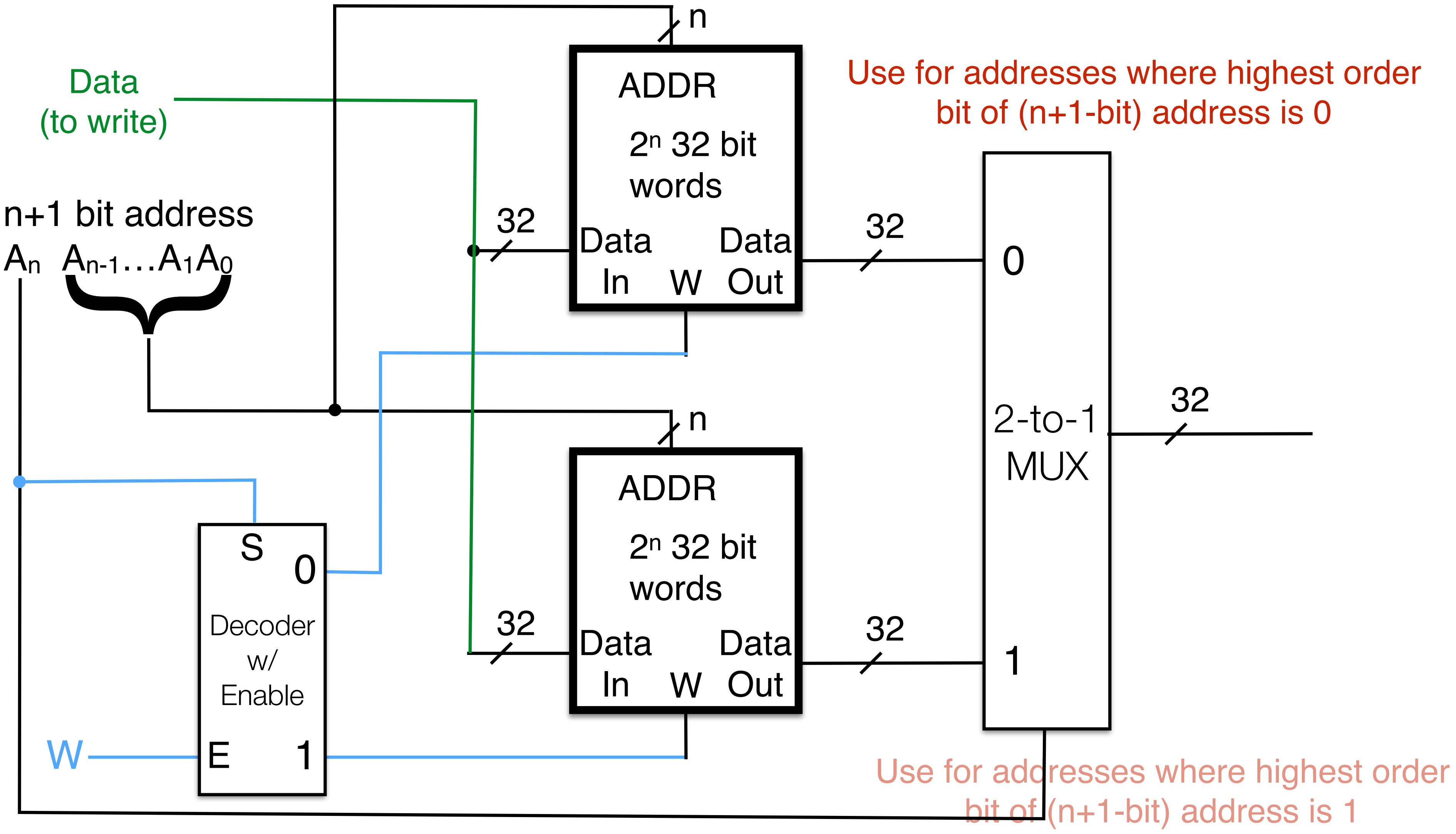


Use for addresses where highest order bit of (n+1-bit) address is 0

Use for addresses where highest order bit of (n+1-bit) address is 1

Multi-chip memories: extend address space

- Suppose each chip stores 2^n 32-bit words (has 2^n n-bit addresses)
- 2 of these chips can store 2^{n+1} 32-bit worse (2^{n+1} (n+1)-bit addresses)



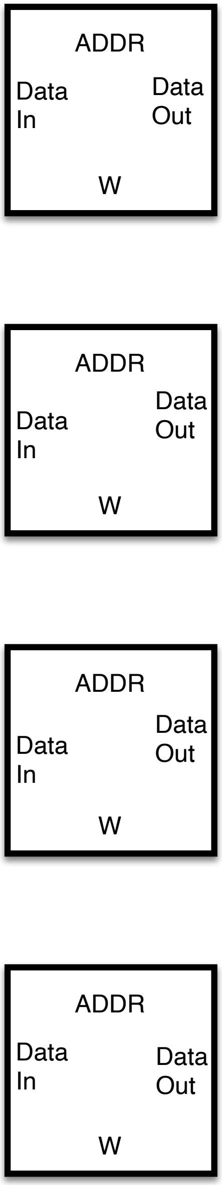
Use for addresses where highest order bit of (n+1-bit) address is 0

Use for addresses where highest order bit of (n+1-bit) address is 1

One more example: quadruple address space

Extending address beyond 2 chips

$A_{n+1}\ A_n\ \ A_{n-1} \ldots A_1 A_0$ (n+2-bit address)

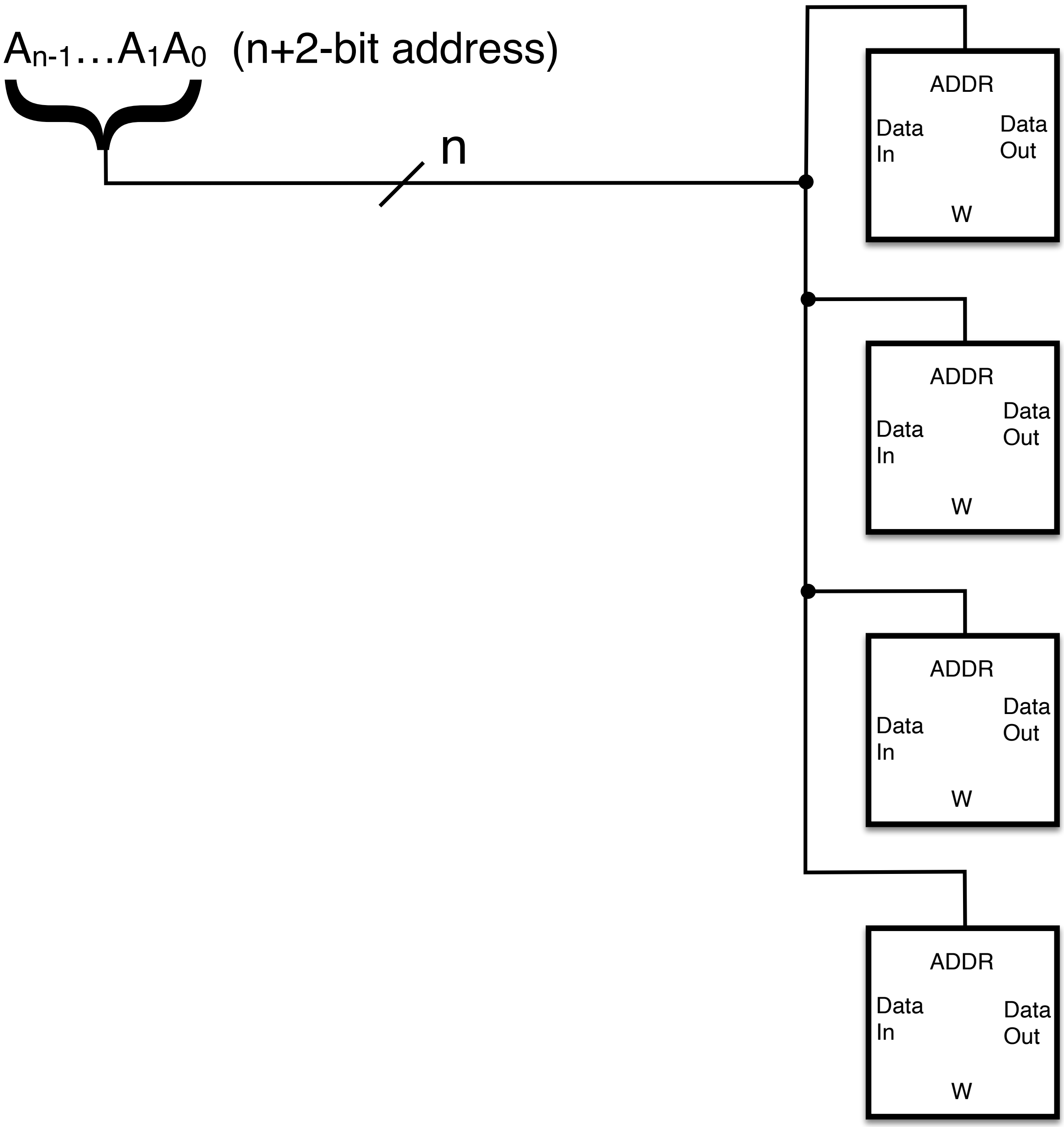


W

Data
(to write)

Extending address beyond 2 chips

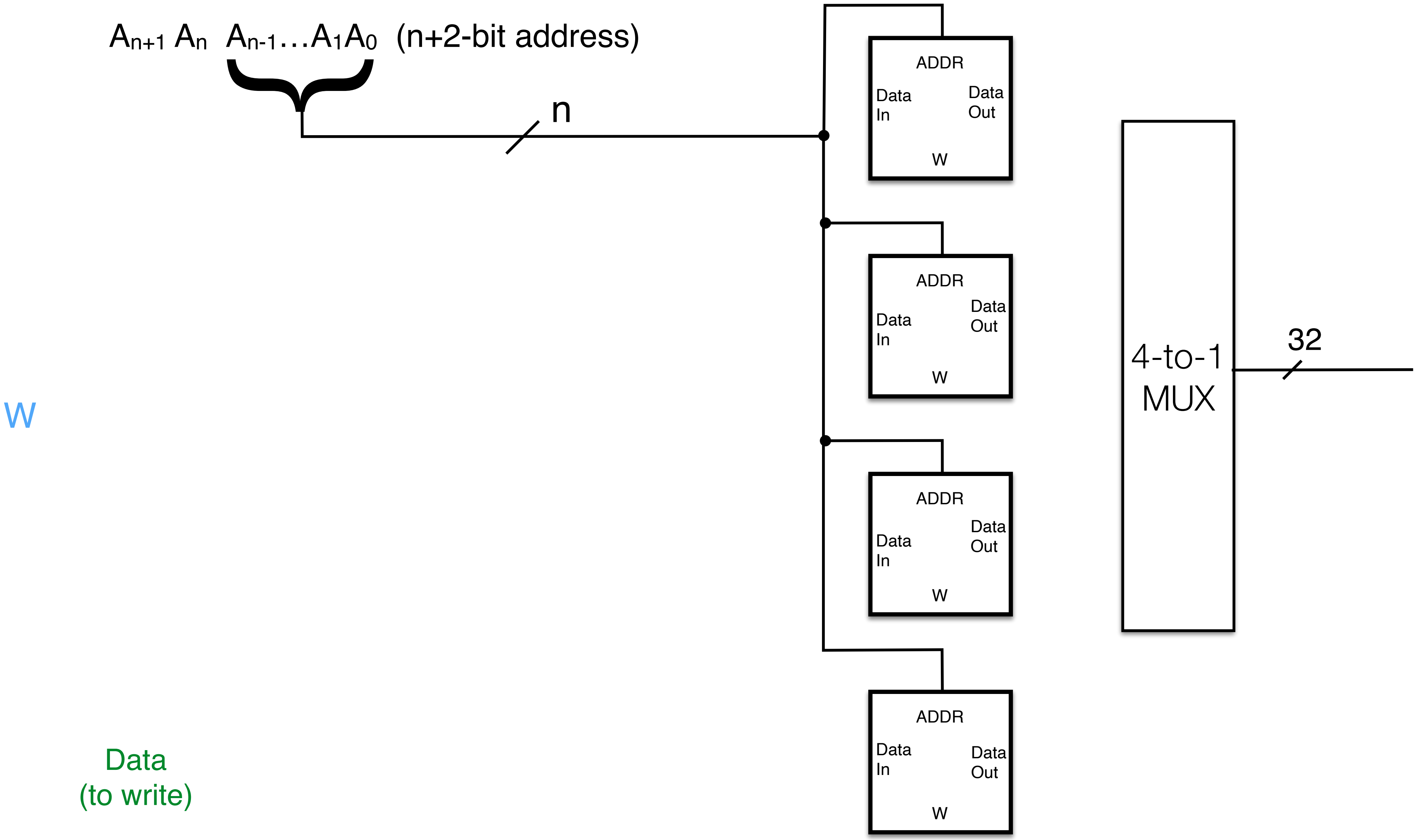
$A_{n+1}\ A_n\ \underbrace{A_{n-1}\ldots A_1A_0}_{n}$ (n+2-bit address)



W

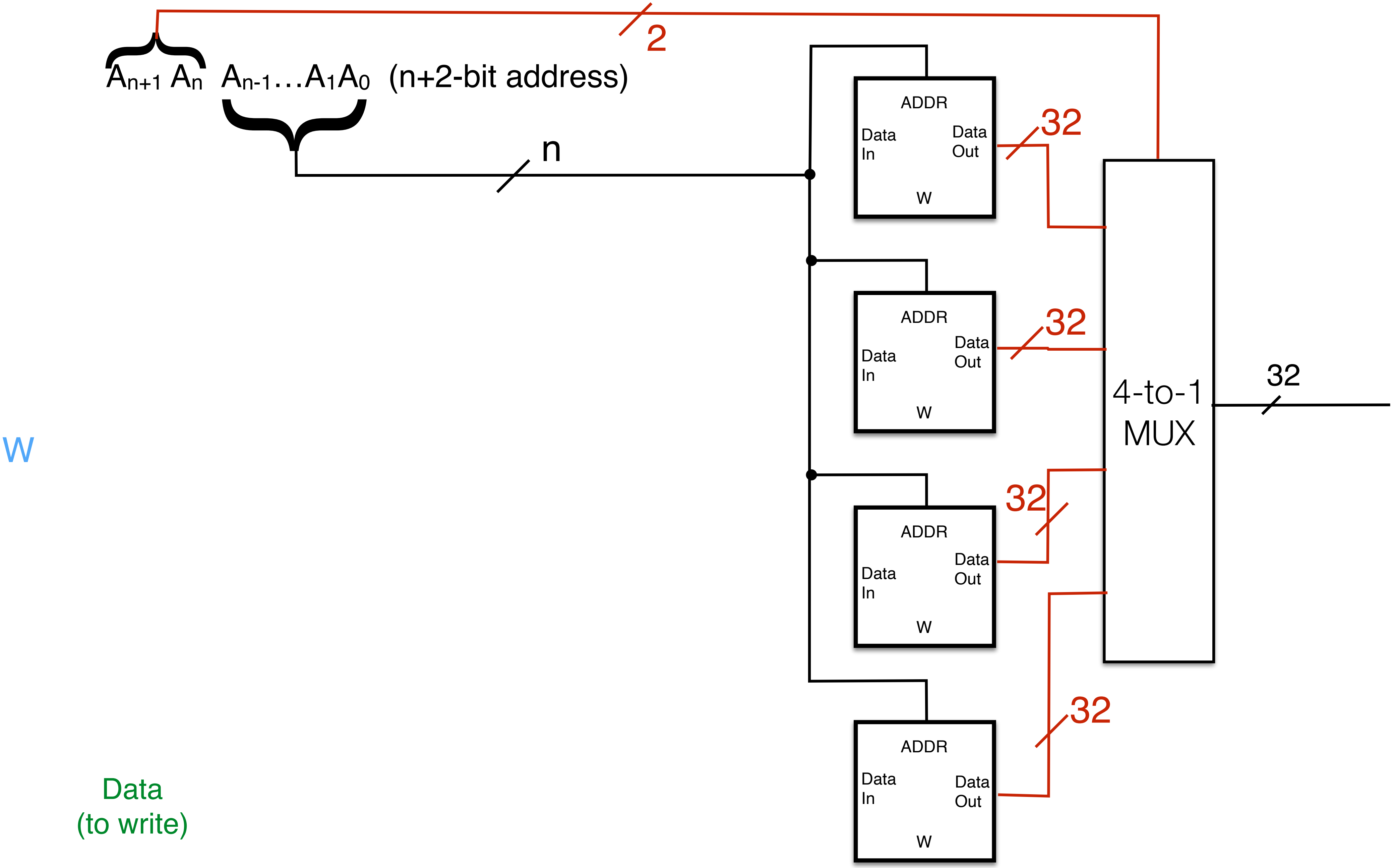
Data
(to write)

Extending address beyond 2 chips

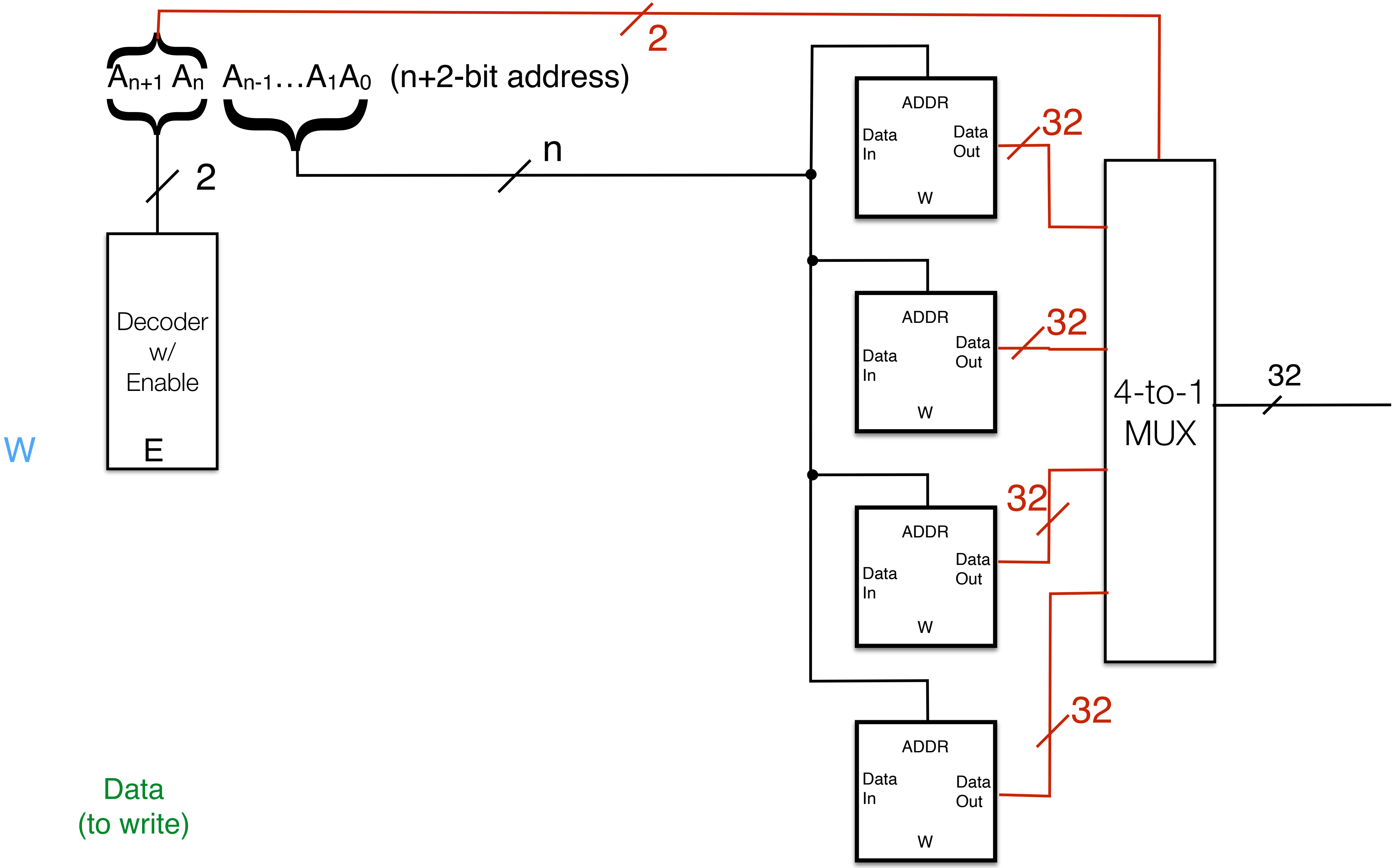


$A_{n+1}\ A_n\ \underbrace{A_{n-1}\ldots A_1A_0}_{n}$ (n+2-bit address)

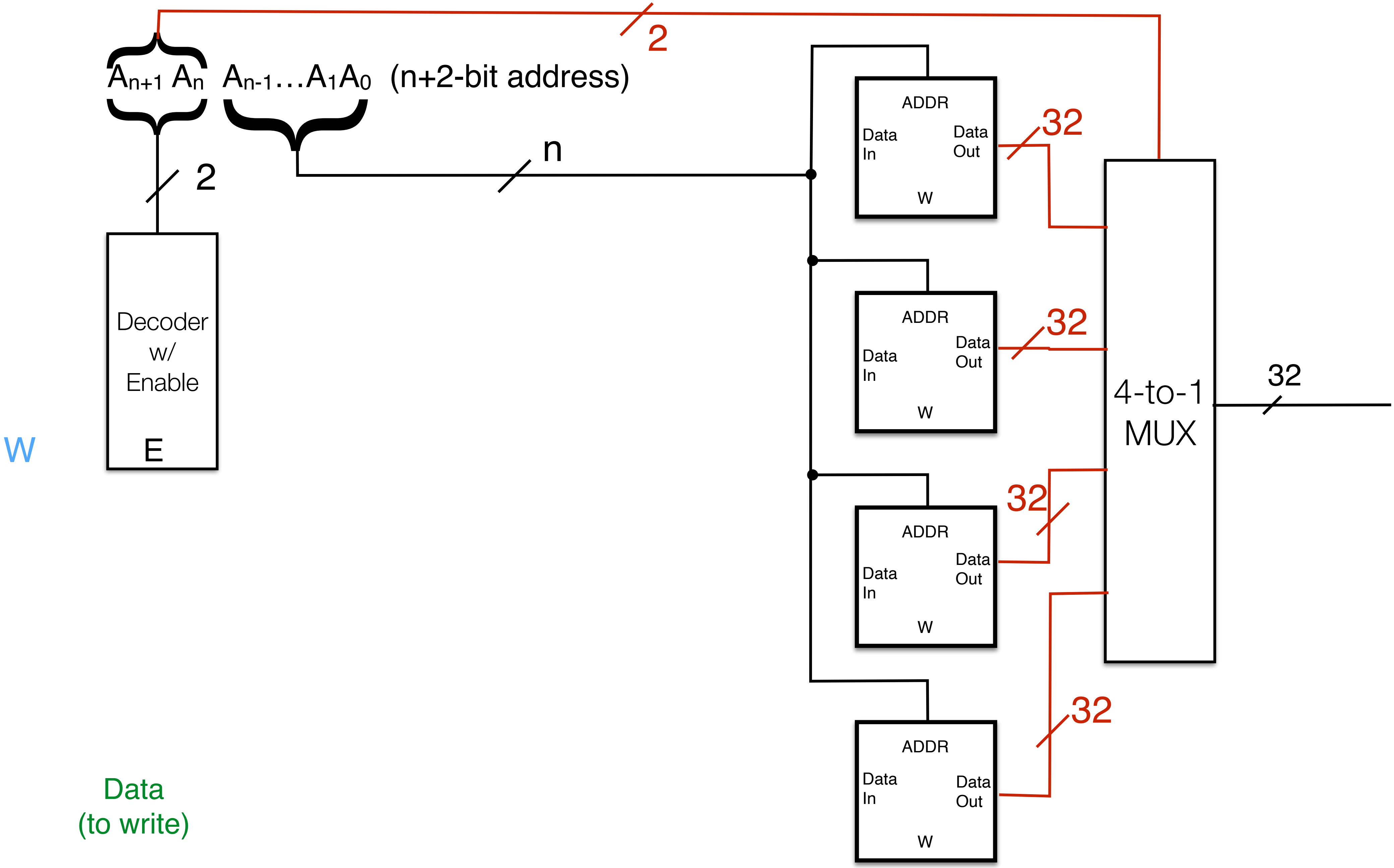
Extending address beyond 2 chips



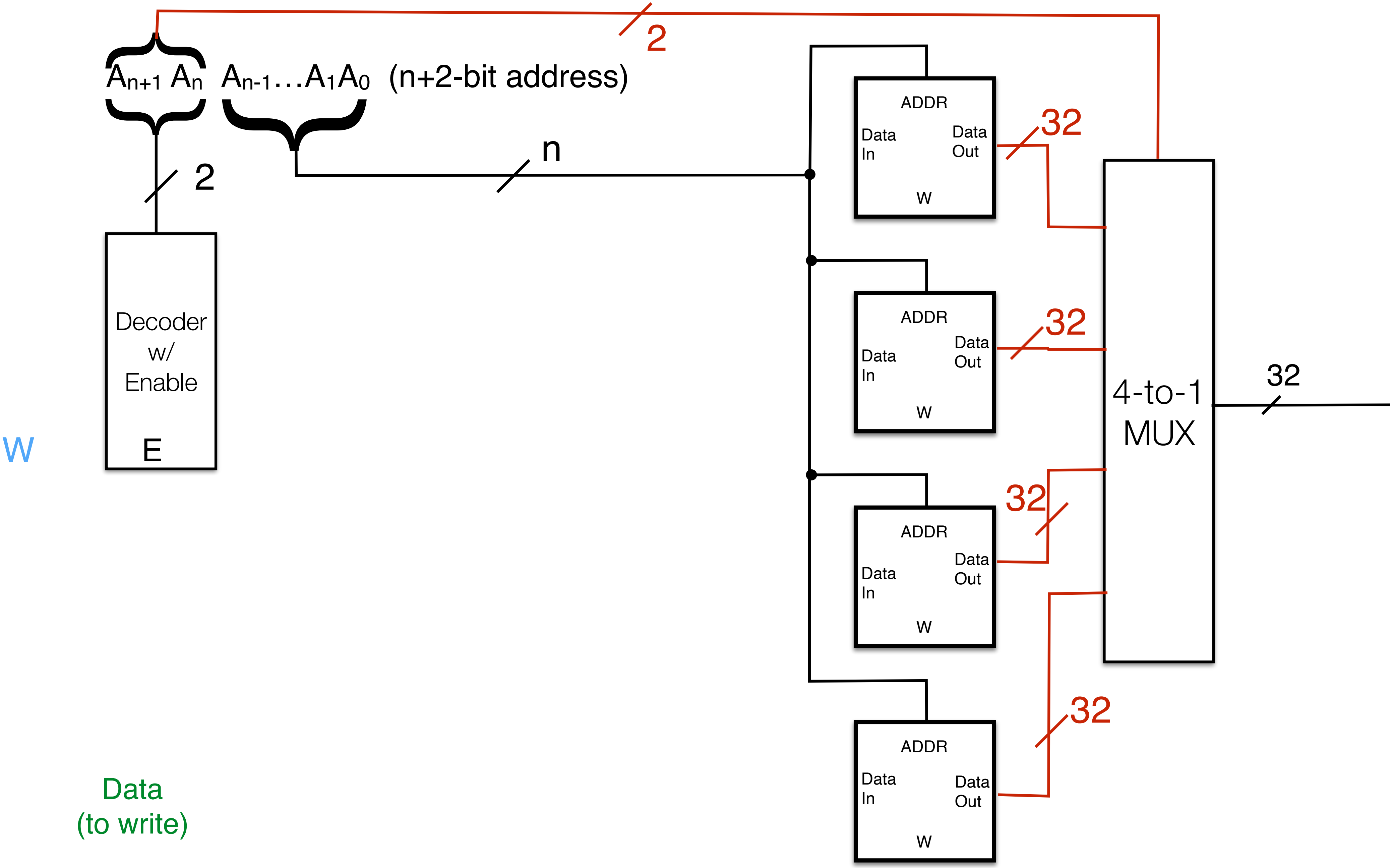
Extending address beyond 2 chips



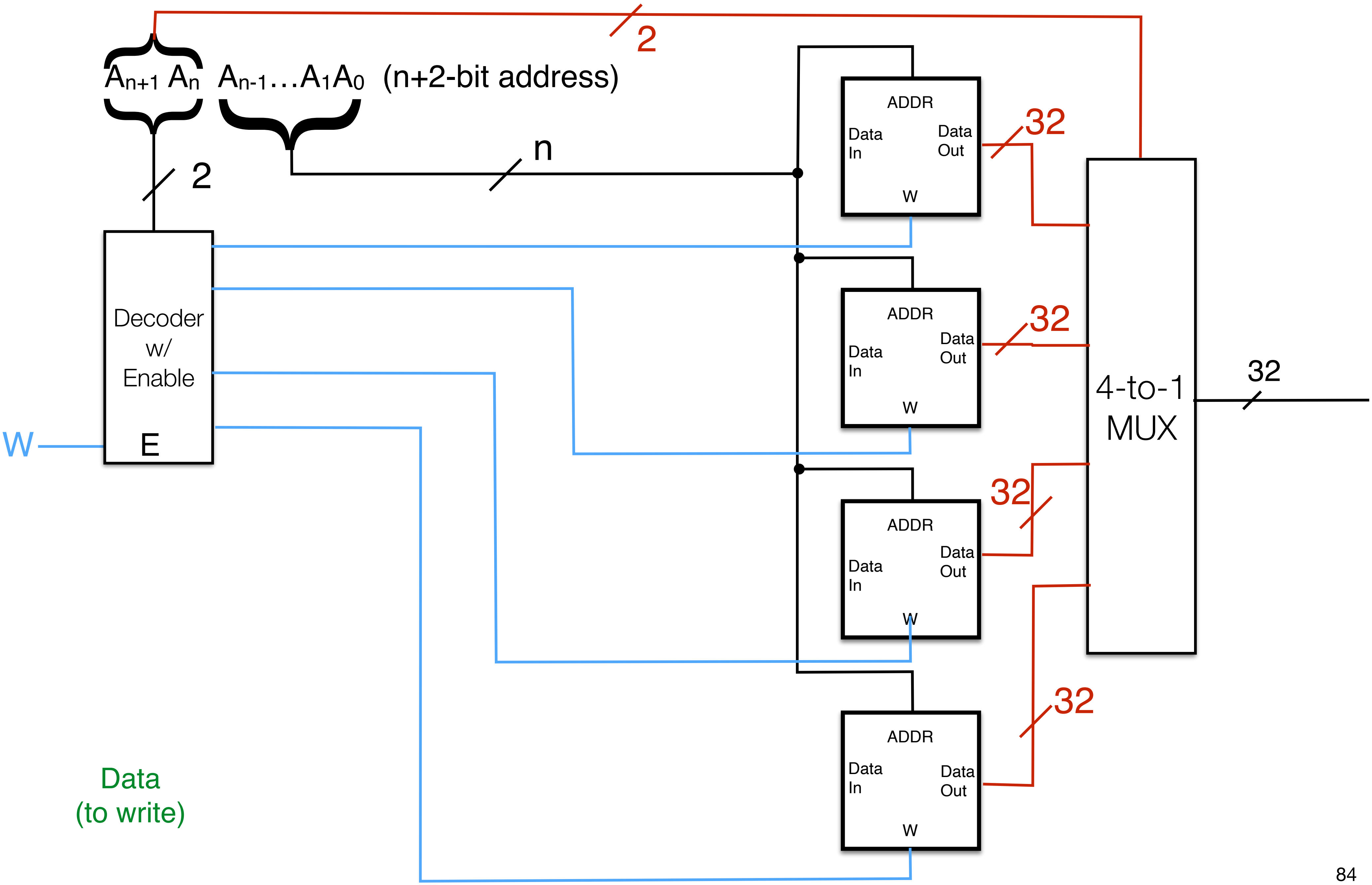
Extending address beyond 2 chips



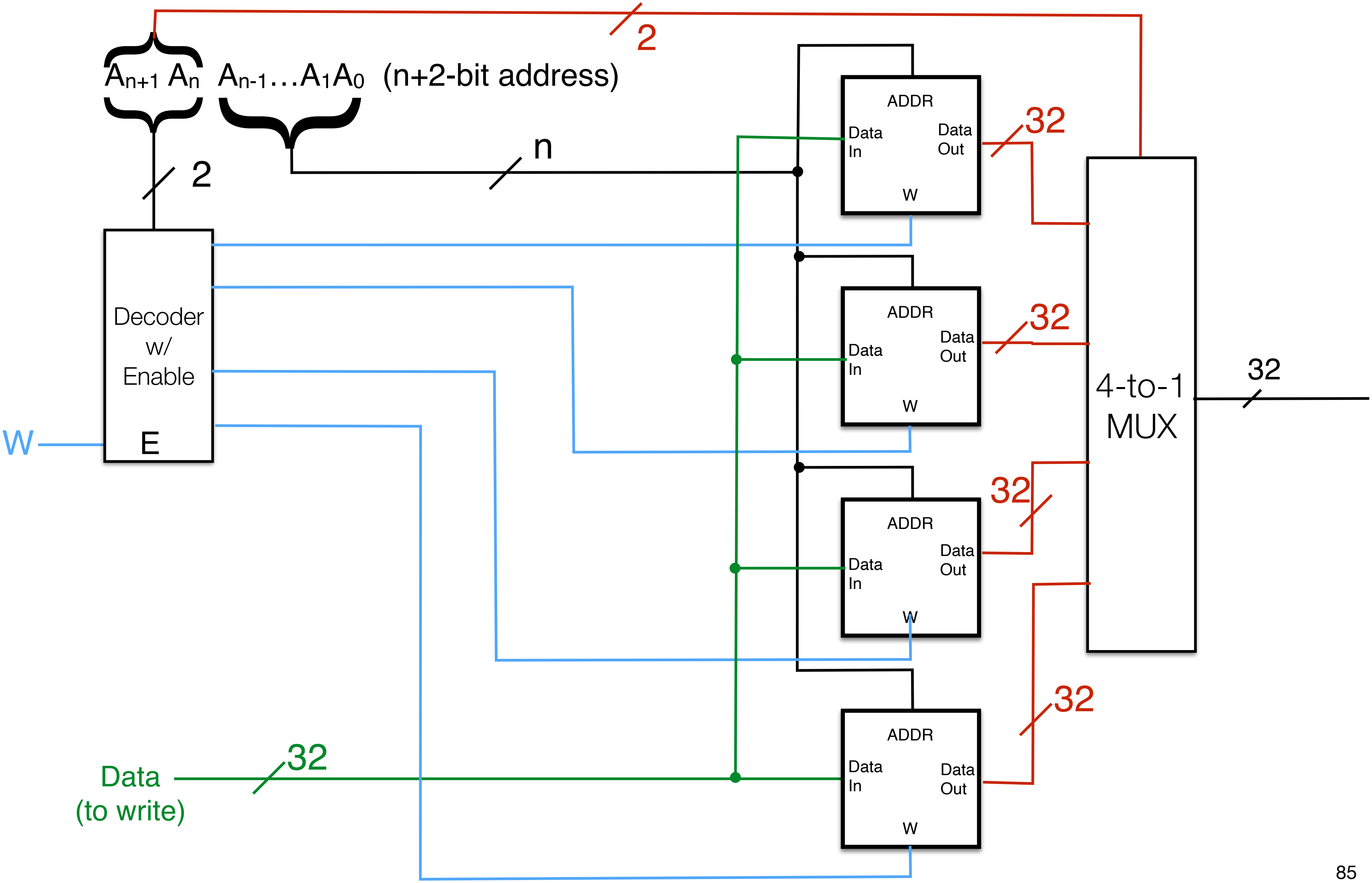
Extending address beyond 2 chips



Extending address beyond 2 chips



Extending address beyond 2 chips



Some Timing Issues with Memory

Memory Timing: Write example

- Even though memory not “on the clock”, timing still an issue:
    - inputs to memory are “on the clock”
    - Real memory takes several clock cycles to access (will deal with this later via cache)
    - must first be properly enabled for reading or writing before data is transferred
        - Write:
            - Appropriate address chosen & Data to write fed in
            - Wait for address & Data to stabilize
            - Activate write
        - Read:
            - Appropriate address chosen
            - Wait for address & corresponding data out to stabilize

Memory Timing: Write example

- Suppose:
  - 2 clock cycles to set address
  - 1 clock cycle to prepare data for writing
  - Write needs 1.5 clock cycles to ensure writing complete

a. Write request (address + data to write) flows to chip

b. Data to write stabilized

c. Address stabilized

d. Memory with new data stabilized

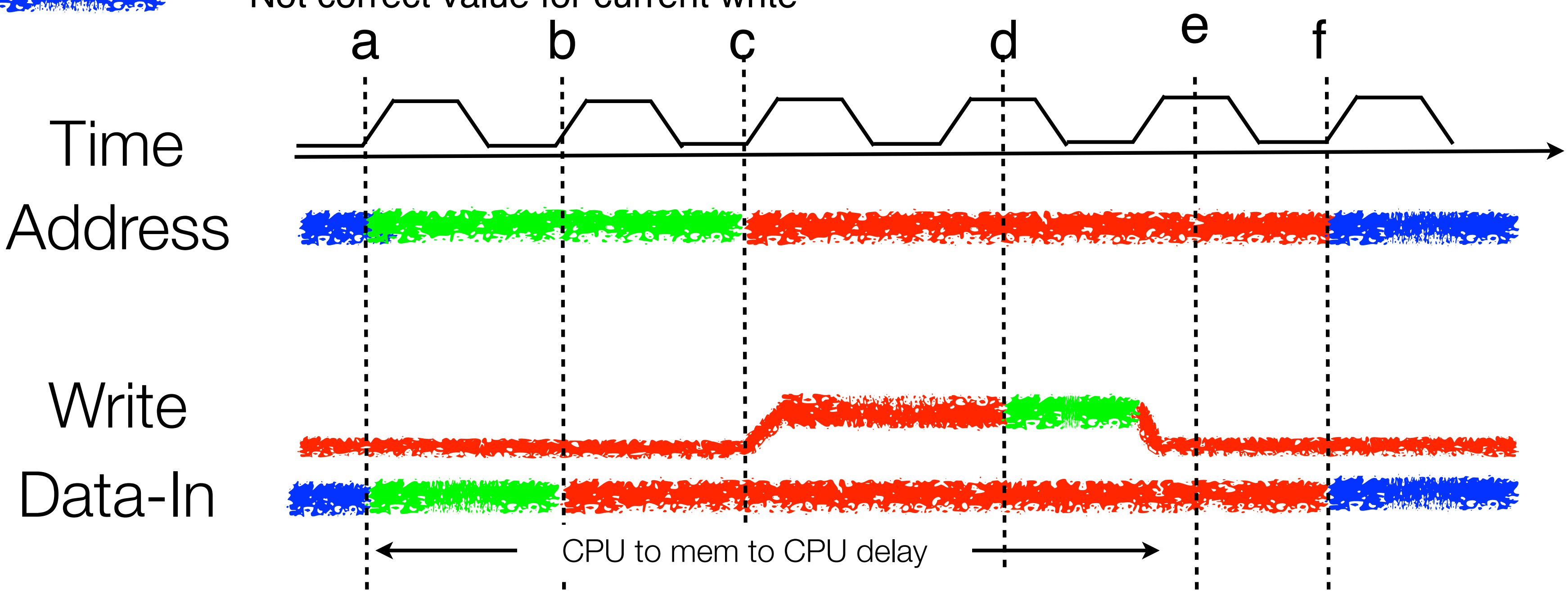
e. Memory “frozen” (no writing)

f. New instruction, new data

🟩 Correct Value arrives

🟥 Correct value “stabilized” in circuit

🟦 Not correct value for current write



Memory Timing: Read example

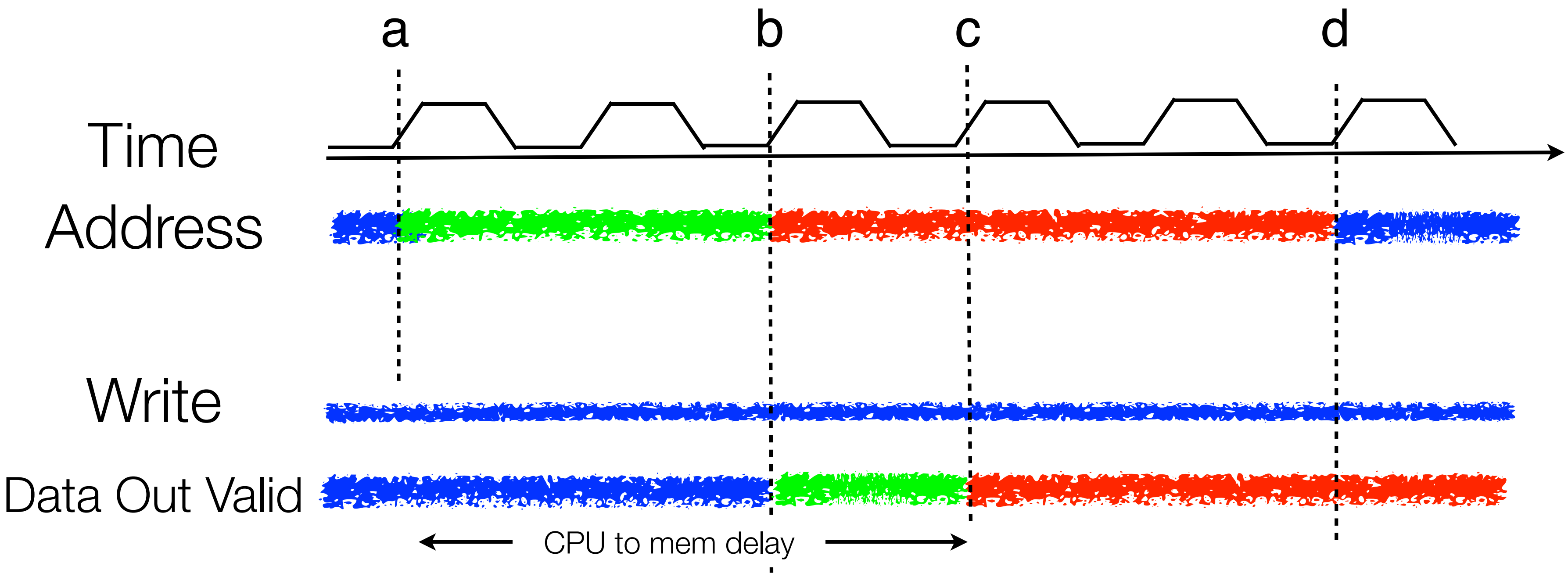
- Suppose:
  - 2 clock cycles to set address
  - 1 clock for data to flow from address to Data Out

a. Read request (address) flows to chip

b. Address stable, data from address flowing out

c. Data Out stable

d. New instruction, new data (read further at own risk…)


a
b
c
d
Time
Address
Write
Data Out Valid
CPU to mem delay