CEG 3320 - Digital System Design

Instructor: Travis Doom, Ph.D.
331 Russ Engineering Center
775-5105
travis.doom@wright.edu
http://www.wright.edu/~travis.doom

Lecture slides created by T. Doom for Wright State University's course in Digital System Design. Some slides contain fair use images or material used with permission from textbooks and slides by R. Haggard, F. Vahid, Y. Patt, J. Wakerly, M. Mano, and other sources.

Module VI:
Digital System Organization

RAM: SRAM, DRAM, SDRAM

Simple microprocessor data/control

A simple microprocessor: the DDmini

The machine cycle: IF OF EX WB

Pipelining

Contemporary Digital/VLSI Design

Random Access Memory

SRAM

DRAM

SDRAM

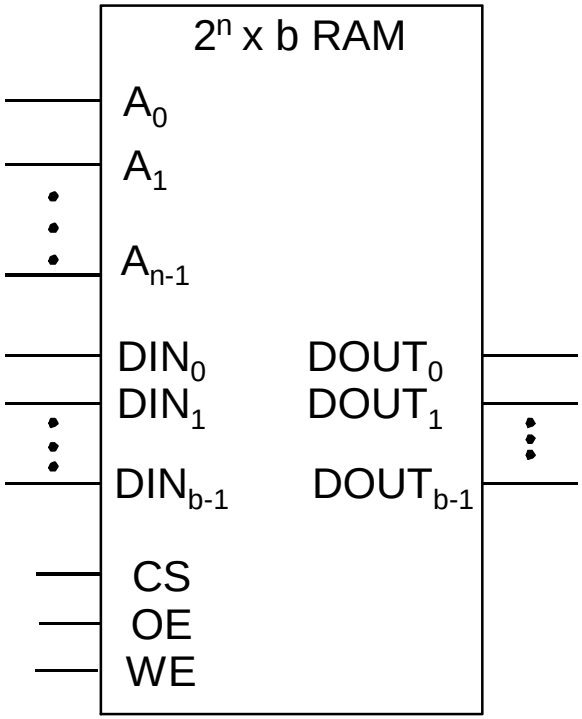
Read/Write Memory (RWM / RAM)

- RWM = RAM (Random Access Memory)
- Highly structured like ROM's and PLD's
- Can store and retrieve data at (relatively) the same speed

- Static RAM (SRAM) retains data in latches (while powered).
- Dynamic RAM (DRAM) stores data as capacitor charge; all capacitors must be recharged periodically.
- Synchronous Dynamic RAM (SDRAM) uses a clock signal to simplify timing characteristics.

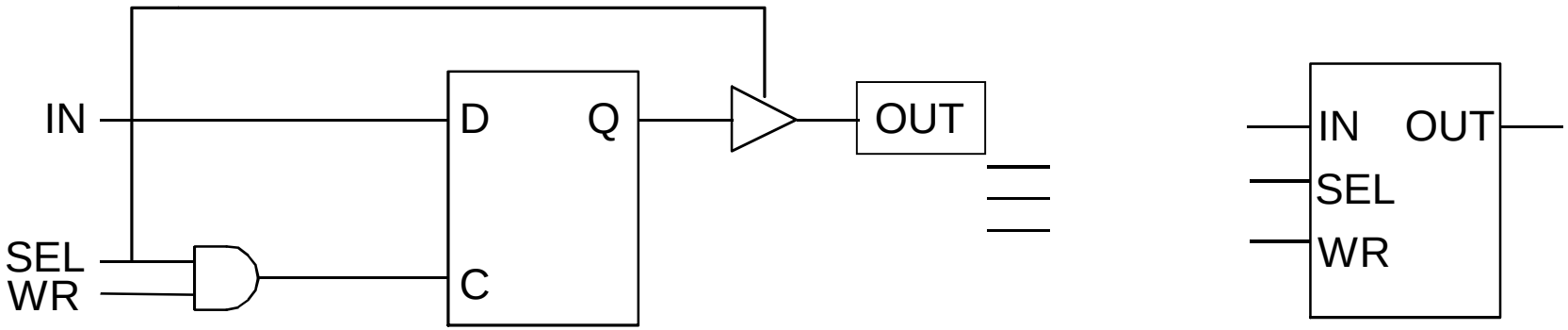
- Volatile Memory: Both Static and Dynamic RAM
- Nonvolatile Memory: Data retained when power lost
  = ROMs, NVRAM (w/battery), Flash Memory

Basic Structure of SRAM

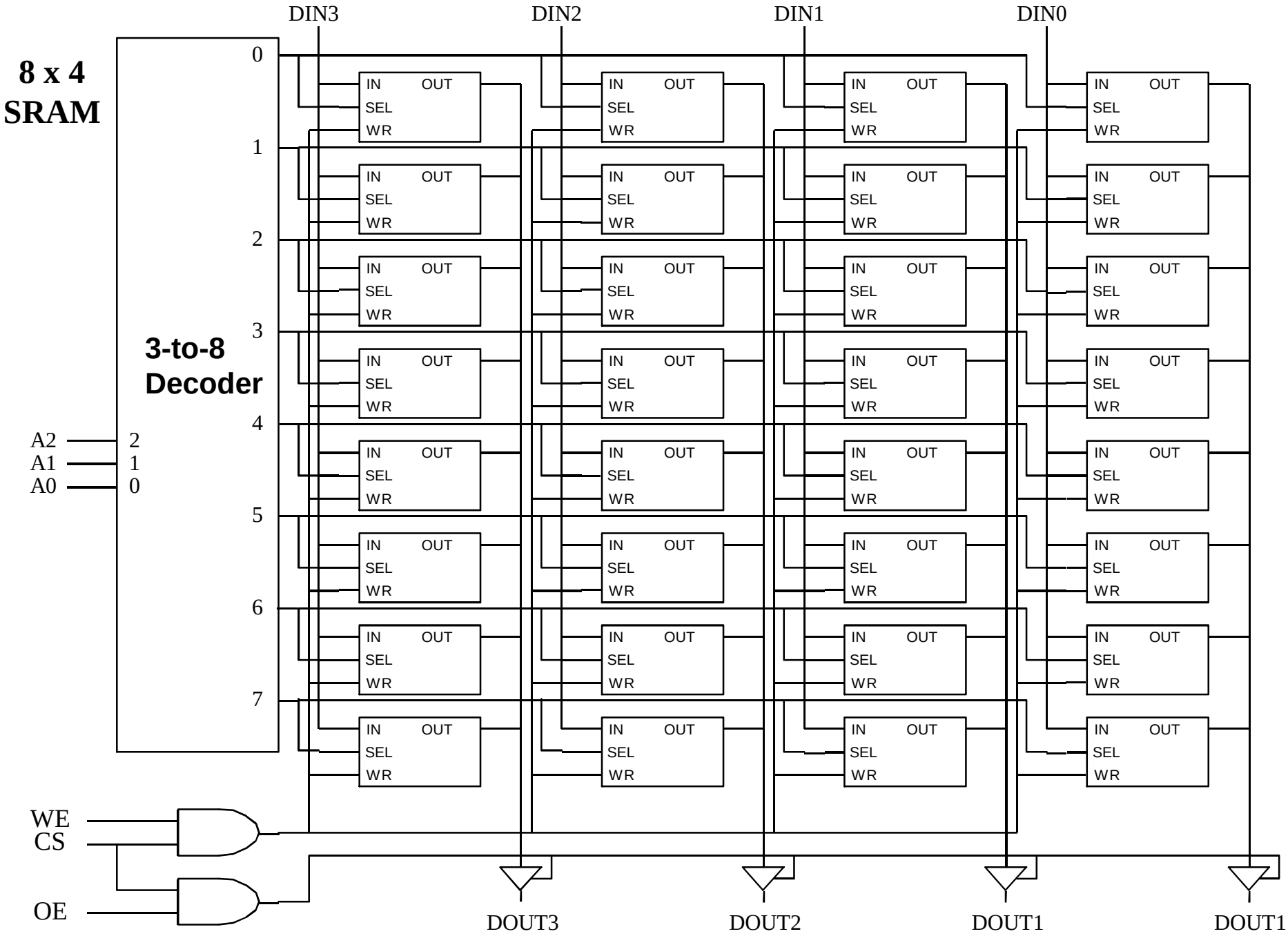


- Address/Control/Data Out lines like a ROM (Reading)
  + Write Enable (WE) and Data In (DIN) (Writing)

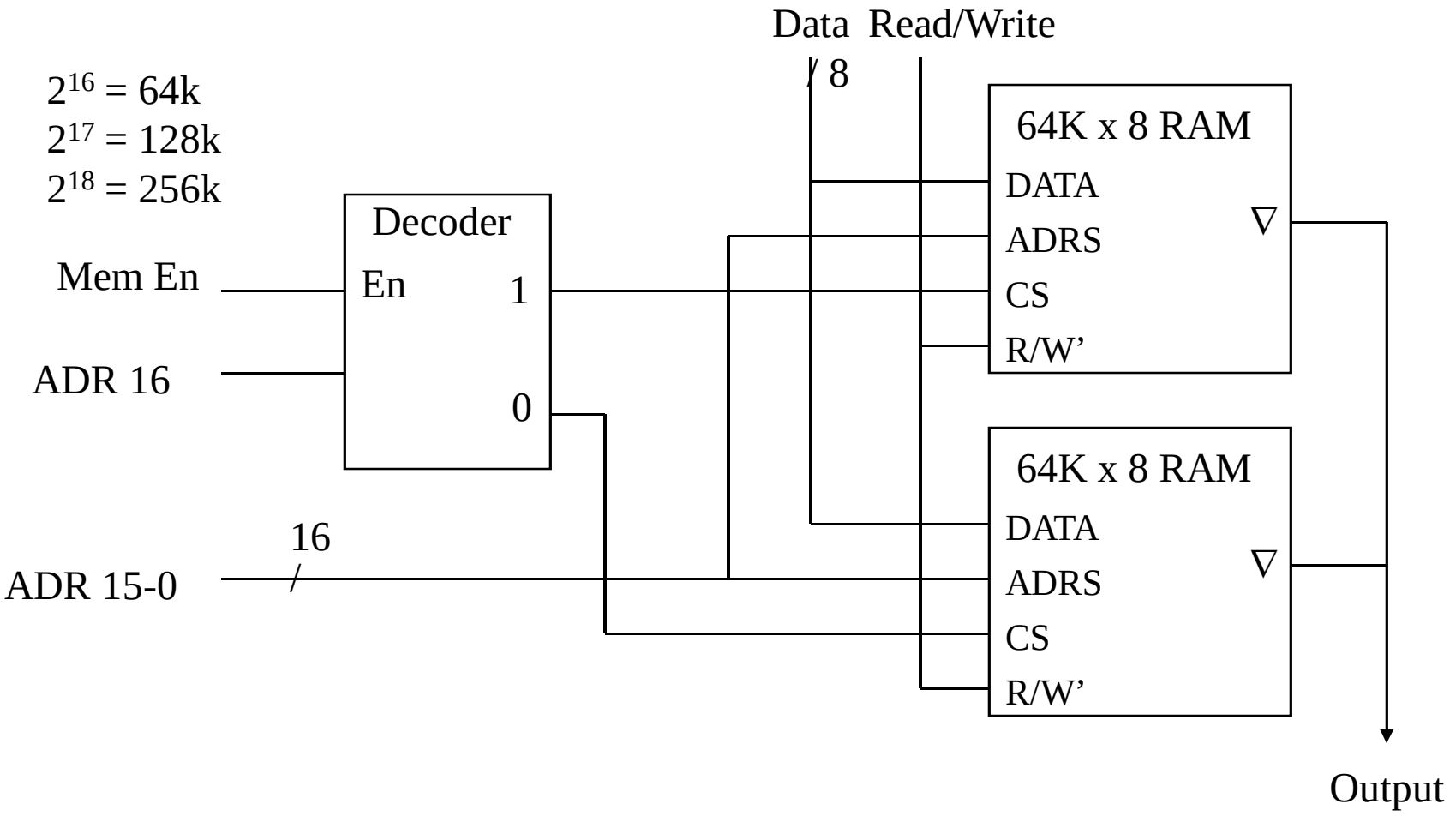
One Bit of SRAM



- SEL and WR asserted → IN data stored in D-latch (Write)
- SEL only asserted → D-latch output enabled (Read)
- SEL not asserted → No operation

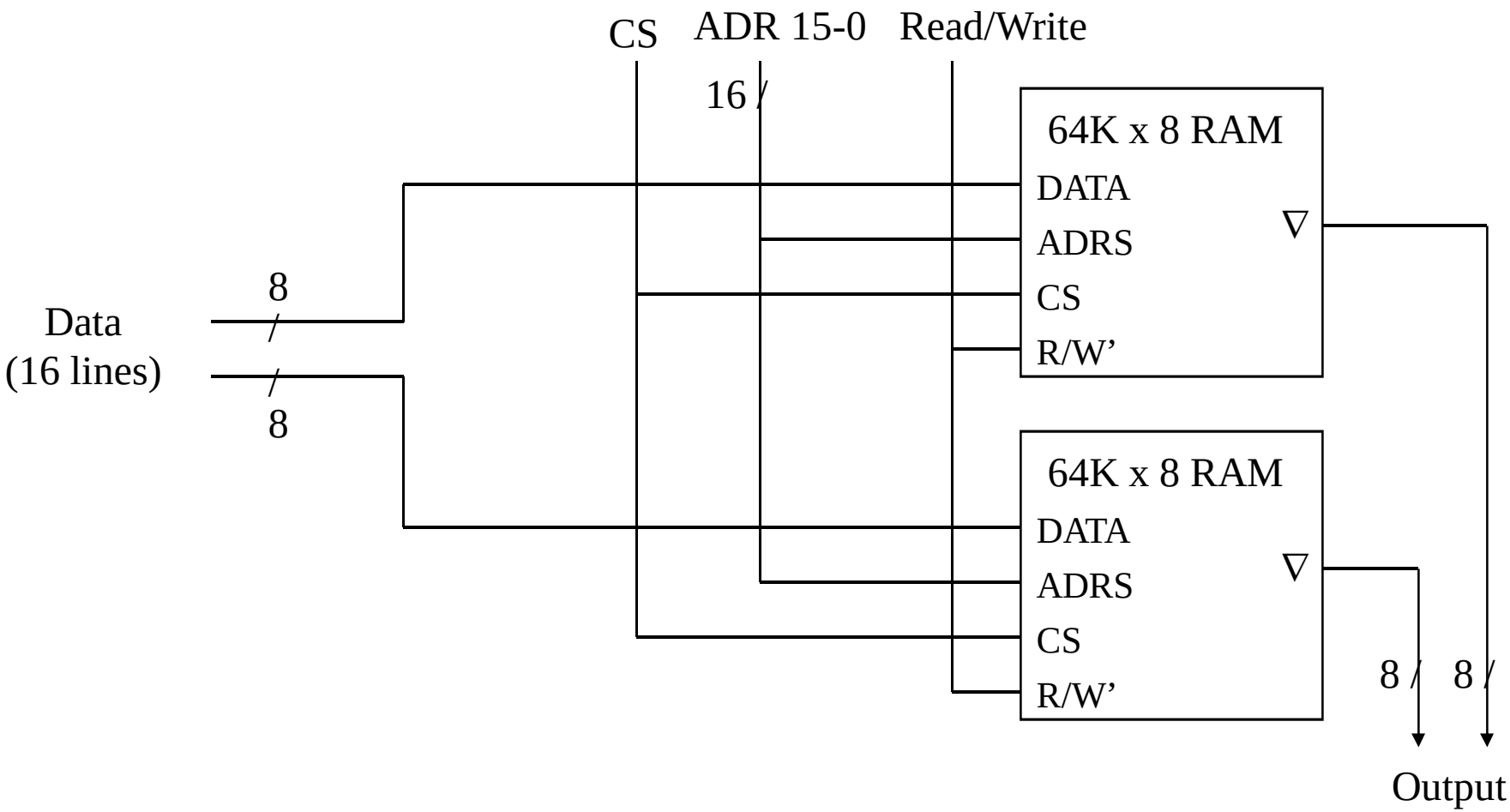

8 x 4
SRAM
3-to-8
Decoder
DIN3
DIN2
DIN1
DIN0
0
1
2
3
4
5
6
7
A2
A1
A0
2
1
0
IN
OUT
SEL
WR
WE
CS
OE
DOUT3
DOUT2
DOUT1
DOUT1

Composite 128K x 8 Memory

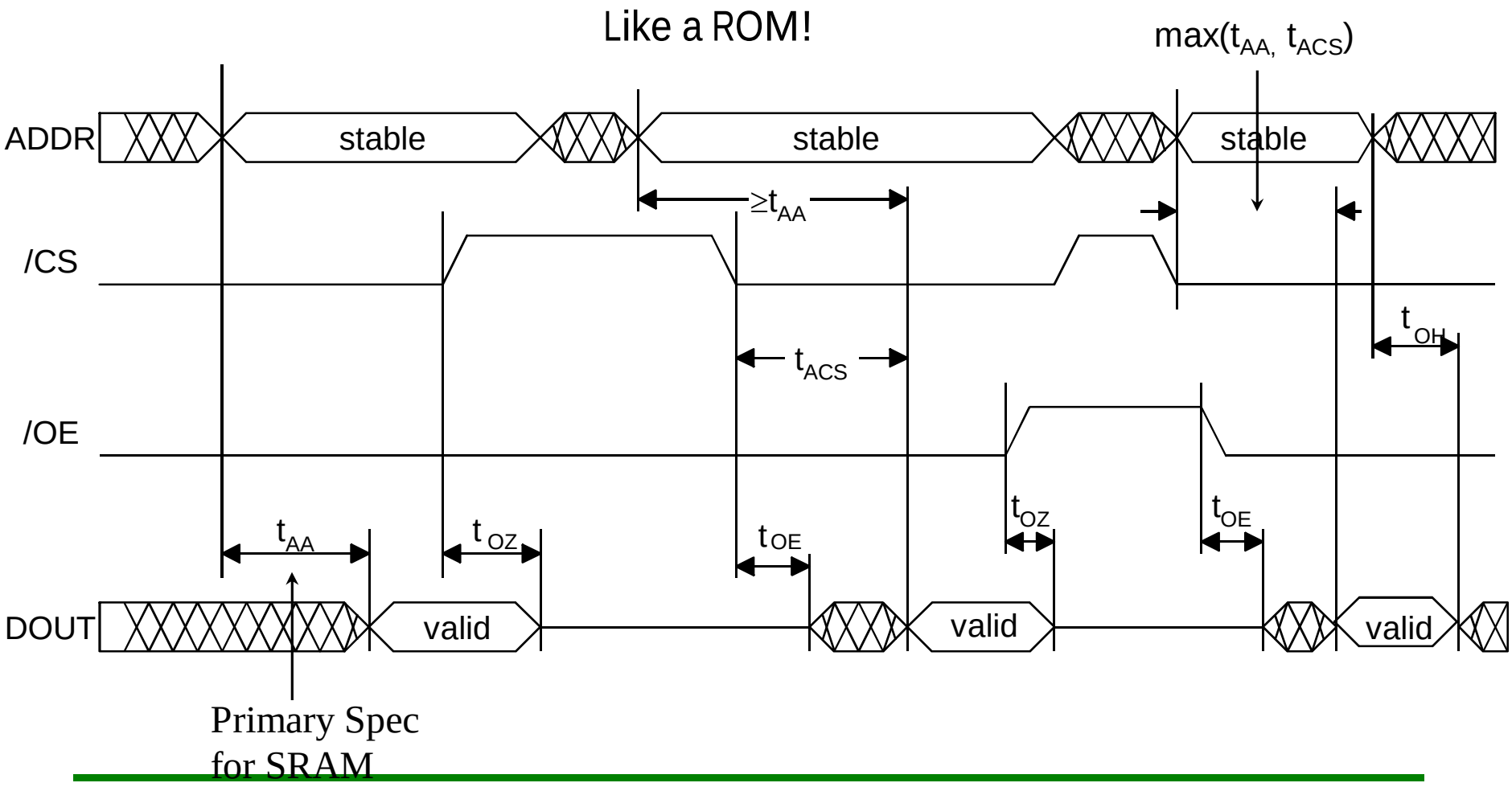


How many 64K x 8 RAMs do we need to create a 256k x 8 Memory?

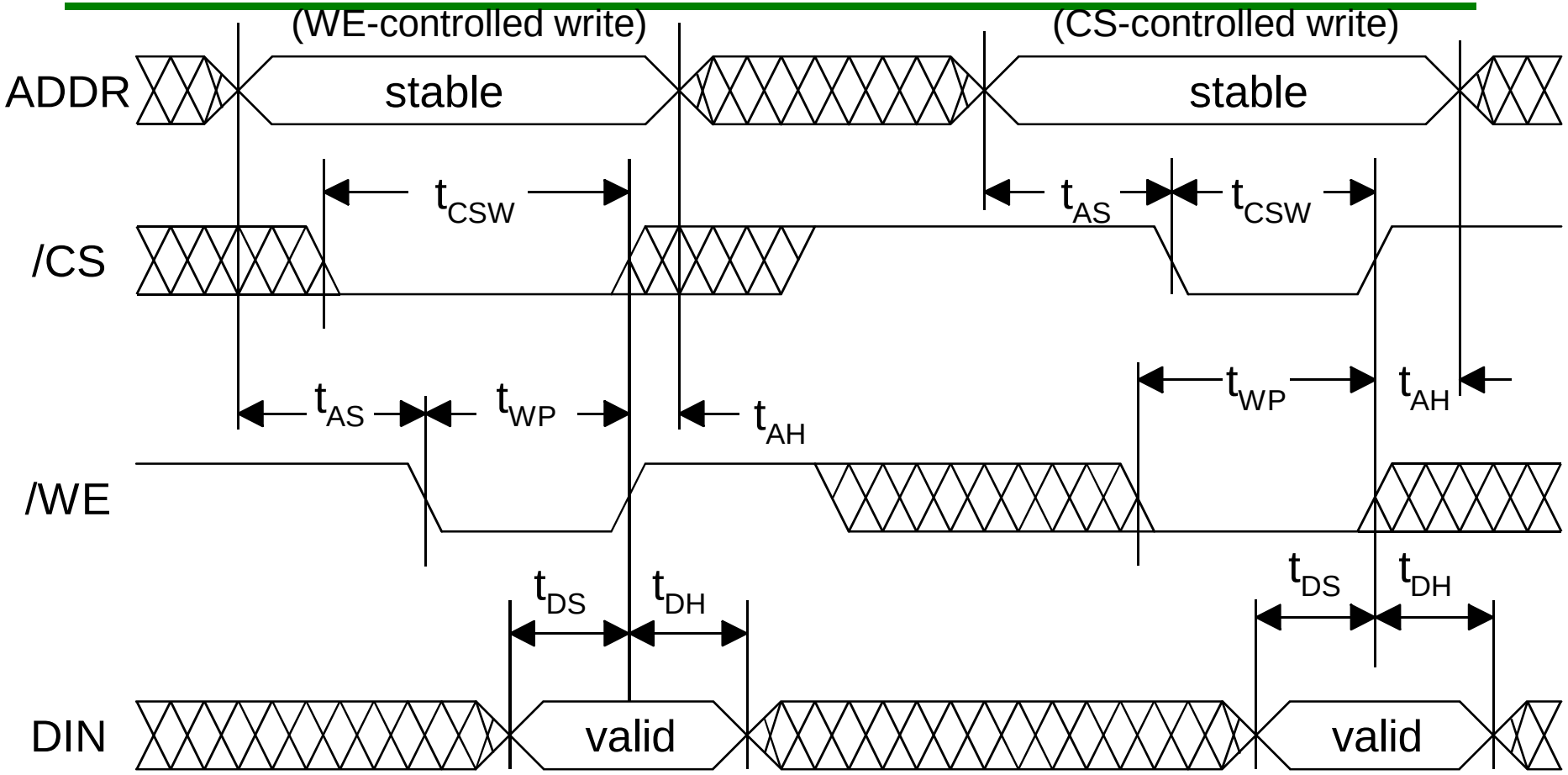
Composite 64K x 16 Memory



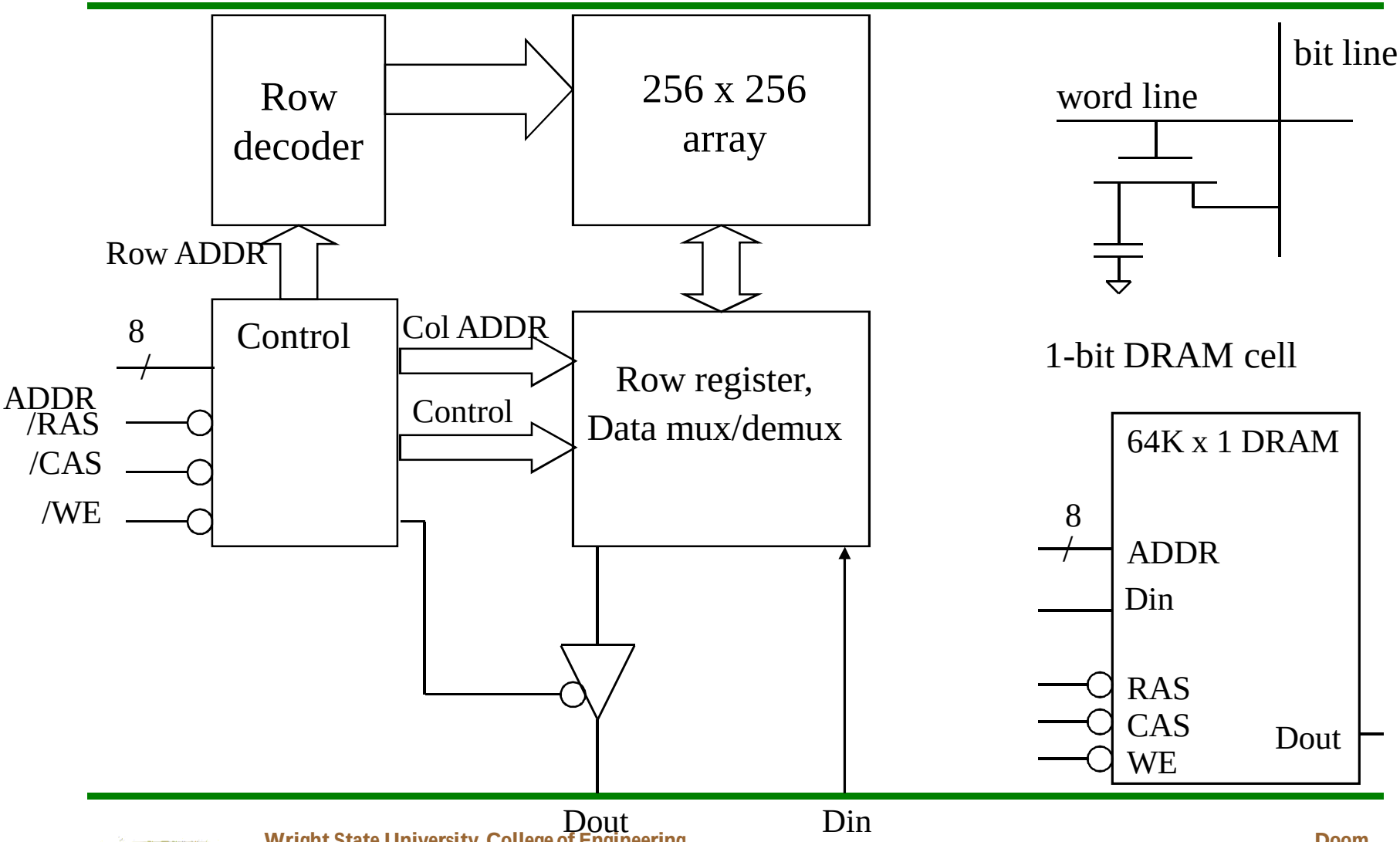
READ Timing (SRAM)



WRITE Timing (SRAM)



64K x 1 DRAM


Row decoder
256 x 256 array
Row ADDR
Control
Col ADDR
Row register, Data mux/demux
Control
8
ADDR
/RAS
/CAS
/WE
Dout
Din
bit line
word line
1-bit DRAM cell
64K x 1 DRAM
8
ADDR
Din
RAS
CAS
WE
Dout

DRAM Timing

- Refresh cycle - RAS Only
  - (RAS asserted) Entire row is latched in row-address register
  - (RAS deasserted) The data in the row-address register is rewritten
- Read cycle - RAS + CAS
  - (RAS asserted) Entire row is latched in row-address register
  - (CAS asserted) Data in register is multiplexed to output
  - (RAS deasserted) Data is row-address register is rewritten
  - (CAS deasserted) Output is released
- Write cycle - RAS + WE + CAS
  - (RAS asserted) Entire row is latched in row-address register
  - (WE asserted) Data_in is stable
  - (CAS asserted) Demultiplex Data_in to row address register
  - (WE deasserted) Data_in is no longer stable
  - (RAS deasserted) Data in row-address register is rewritten
  - (CAS deasserted) Operation complete

RAM Summary

SRAM:

- Fast
- Simple Interface
- Moderate bit density (4 gates → 4 to 6 transistors)
- Moderate cost/bit

Small systems
or
very fast
applications
(cache memory)

DRAM (Dynamic RAM):

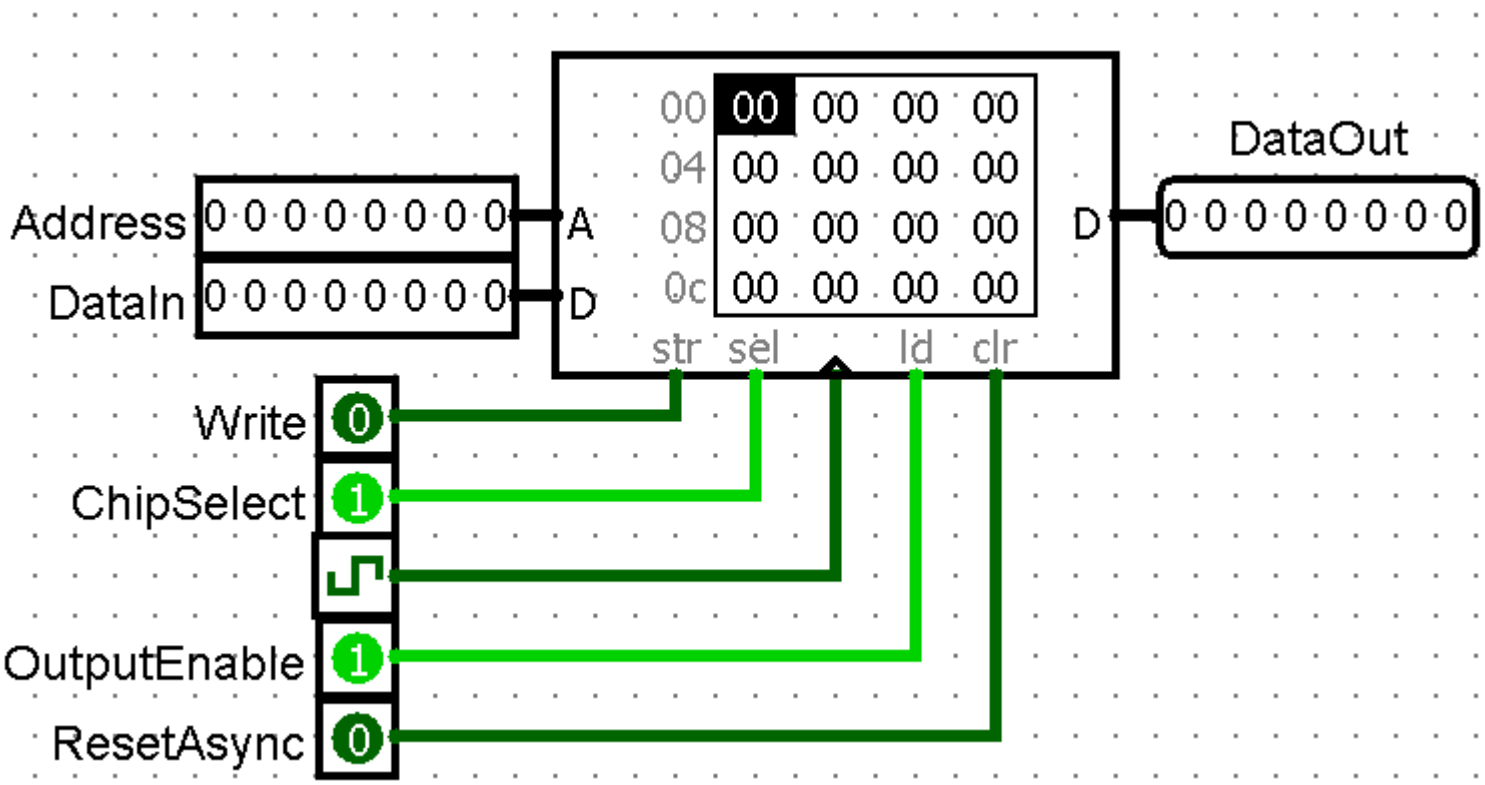
- moderate speed
- complex interface
- High bit density (1 transistor cell)
- Low cost/bit

Large Memories:
PC's
Mainframes

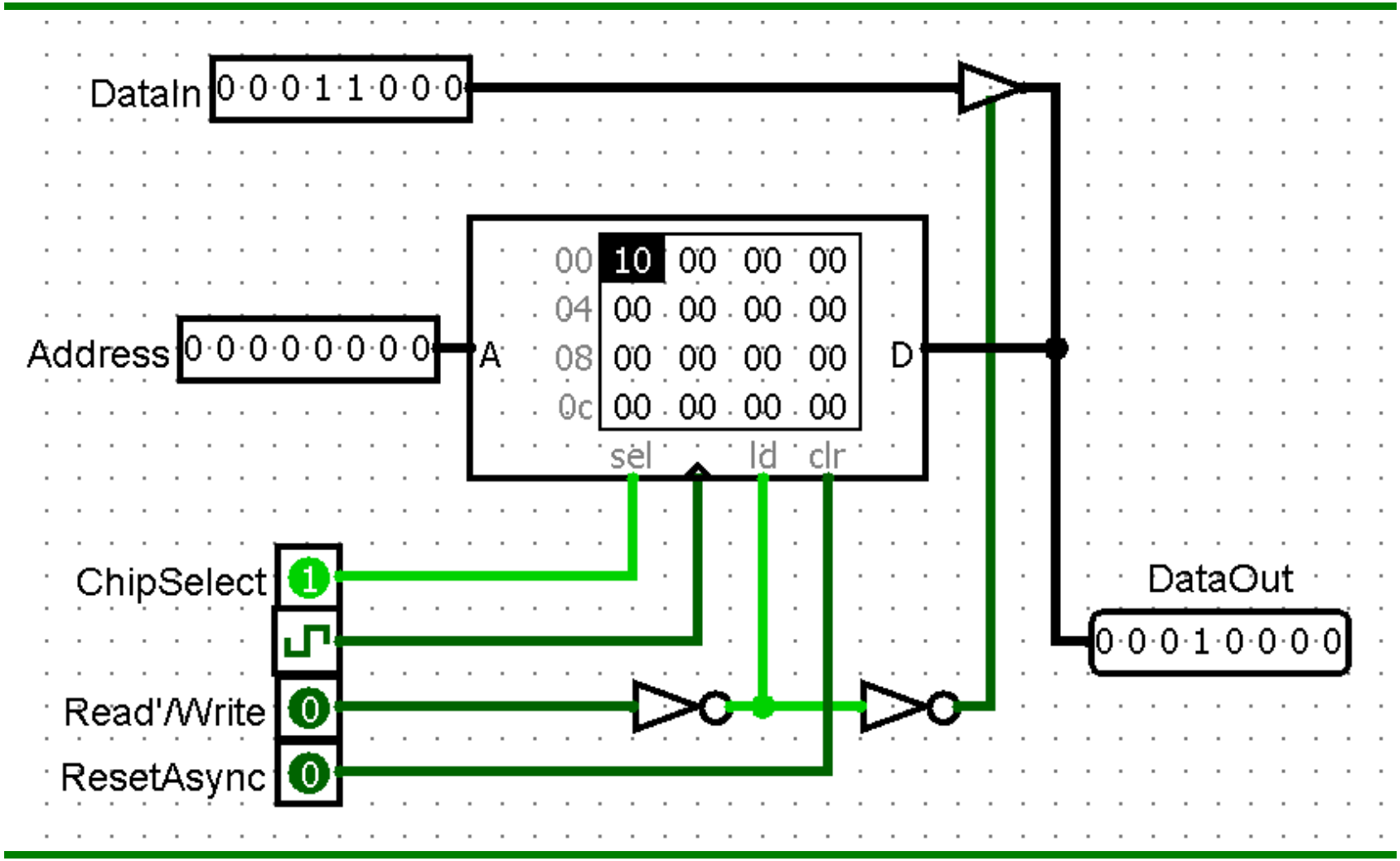
Modern DRAM Timing

- Fast-Page Mode (One RAS, multiple CAS)
  - Multiple bits of a row can be written before rewrite
  - Complex control, but much faster
- Extended Data Out (One RAS, multiple CAS)
  - Latches the column address so that the next address can be prepared *while* the output is read
  - Saves ~10ns/read, and increase of 10-15%
  - Even more complex control!
- SDRAM - Synchronous DRAM
  - Unlike normal DRAM, SDRAM is clocked!
  - Multiple signals and banks (row-address registers) allow “pipelined” operation

Synchronous DRAM with separate load and store ports



Synchronous DRAM with one load/store port



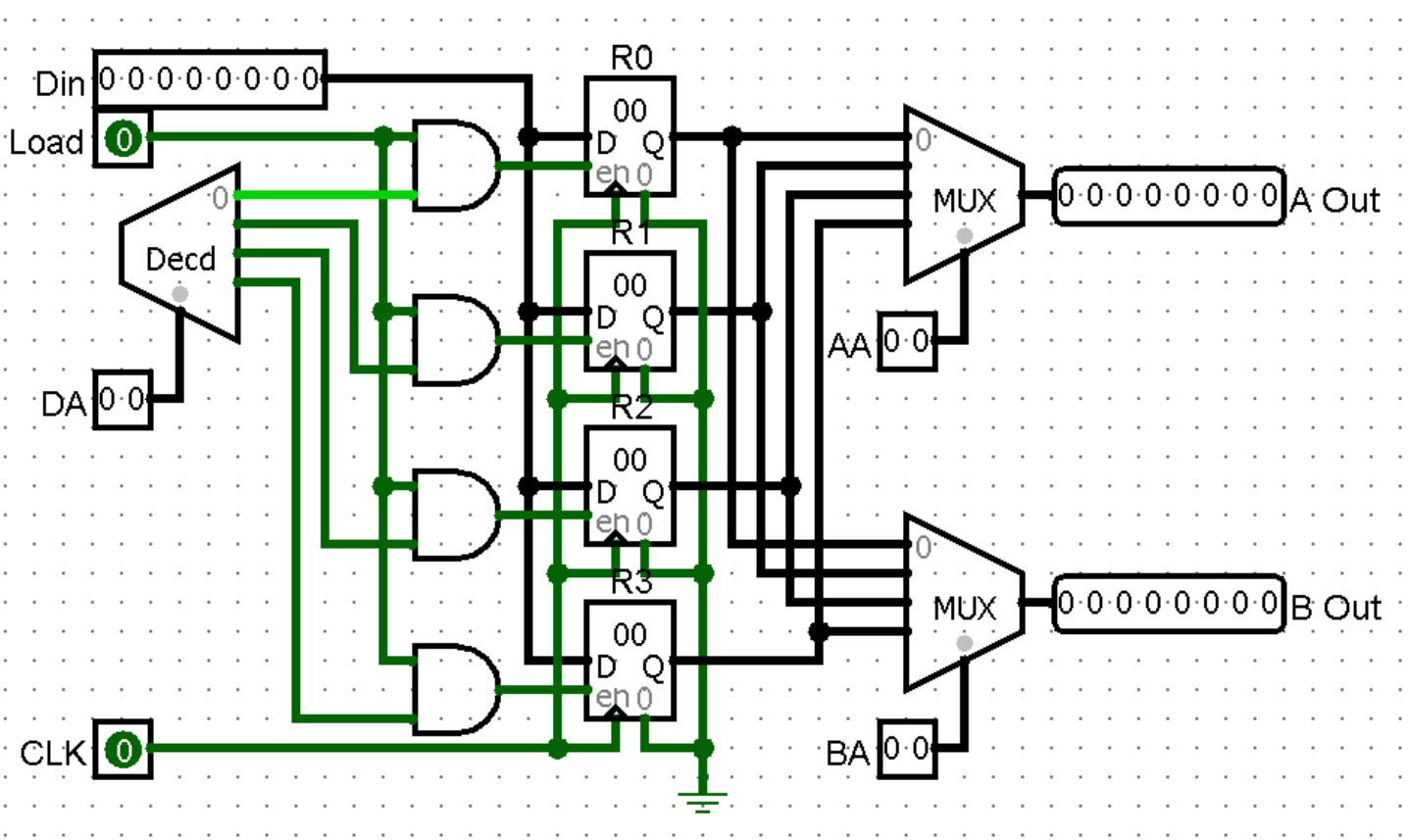
A simple microprocessor

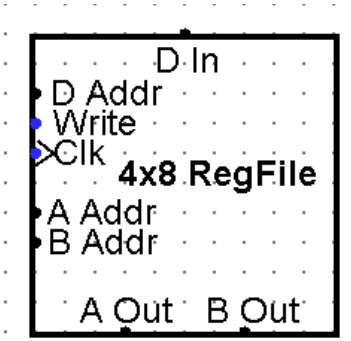
Datapath

Control

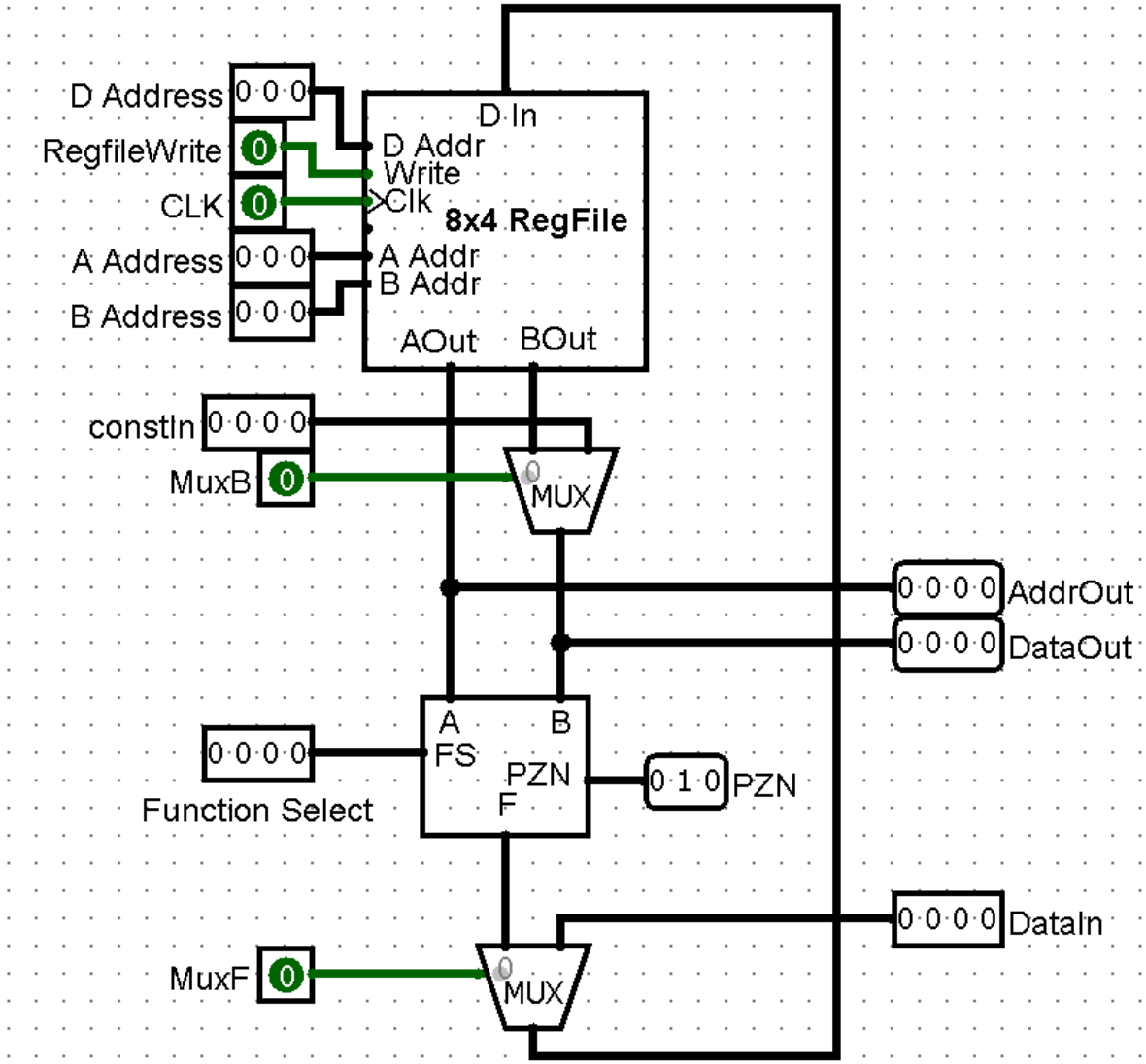
Micro programmed control

Register File





A simple microcomputer datapath



FS	Microoperation
x0	F = 0
x1	F = 1
x2	F = A
x3	F = B
x4	F = A + B
x5	F = A – B
x6	F = A * B
x7	F = A / B
x8	F = NOT(A)
x9	F = A AND B
xA	F = A OR B
xB	F = NOT(A) + 1
xC	F = A >> 1
xD	F = A << 1
xE	F = A >>> 1
xF	Reserved

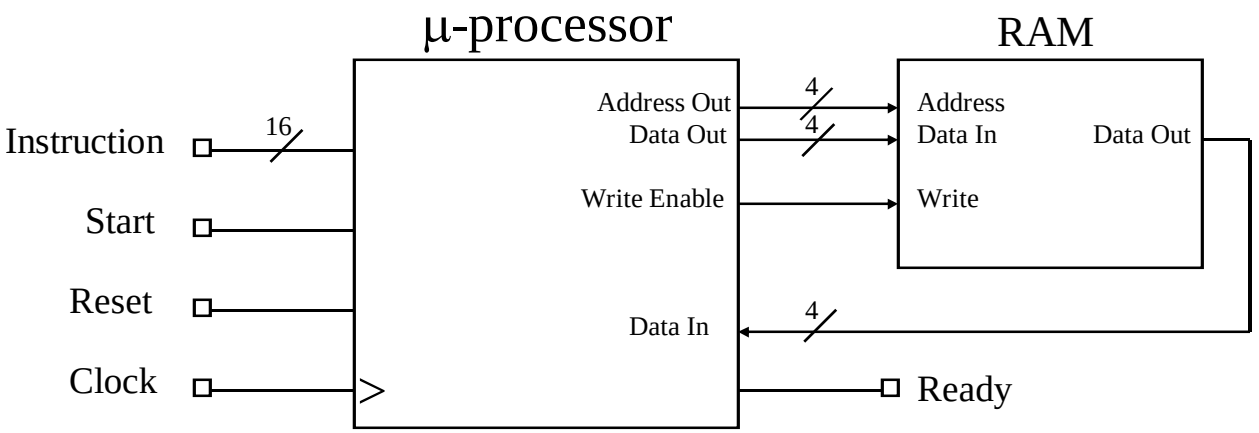
15:13	12	11:9	8	7:4	3:1	0
AA (3)	MB (1)	BA (3)	MF (1)	FS (4)	DA (3)	RW (1)

Symbolic Notation

AA, BA, DA		MB		FS		MD		RW	
Function	Code	Function	Code	Function	Code	Function	Code	Function	Code
R0	000	Register	0	F = 0	00 00	Function	0	No Write	0
R1	001	Constant	1	F = 1	00 01	Data In	1	Write	1
R2	010			F = A	00 10				
R3	011			F = B	00 11				
R4	100			F = A+B	01 00				
R5	101			F = A-B	01 01				
R6	110			F = A*B	01 10				
R7	111			F = A/B	01 11				
				F = A’	10 00				
				F = A ^ B	10 01				
				F = A v B	10 10				
				F = A’+1	10 11				
				F = A>>1	11 00				
				F = A<<1	11 01				
				F = A>>>1	11 10				

15:13	12	11:9	8	7:4	3:1	0
AA (3)	MB (1)	BA (3)	MF (1)	FS (4)	DA (3)	RW (1)

A Simple Computer: the DDmini



Instruction Format: Opcode [15:12], Op A [11:8], Op B [7:4], Op C [3:0]
Example Instruction: x1021
Opcode 2 Format: M[A] ←B
Instruction: M[x0] ← x2

Ram Contents (Before)

ADDR	DATA
x0	x4
x1	xF
x2	x9
…	…

Ram Contents (After)

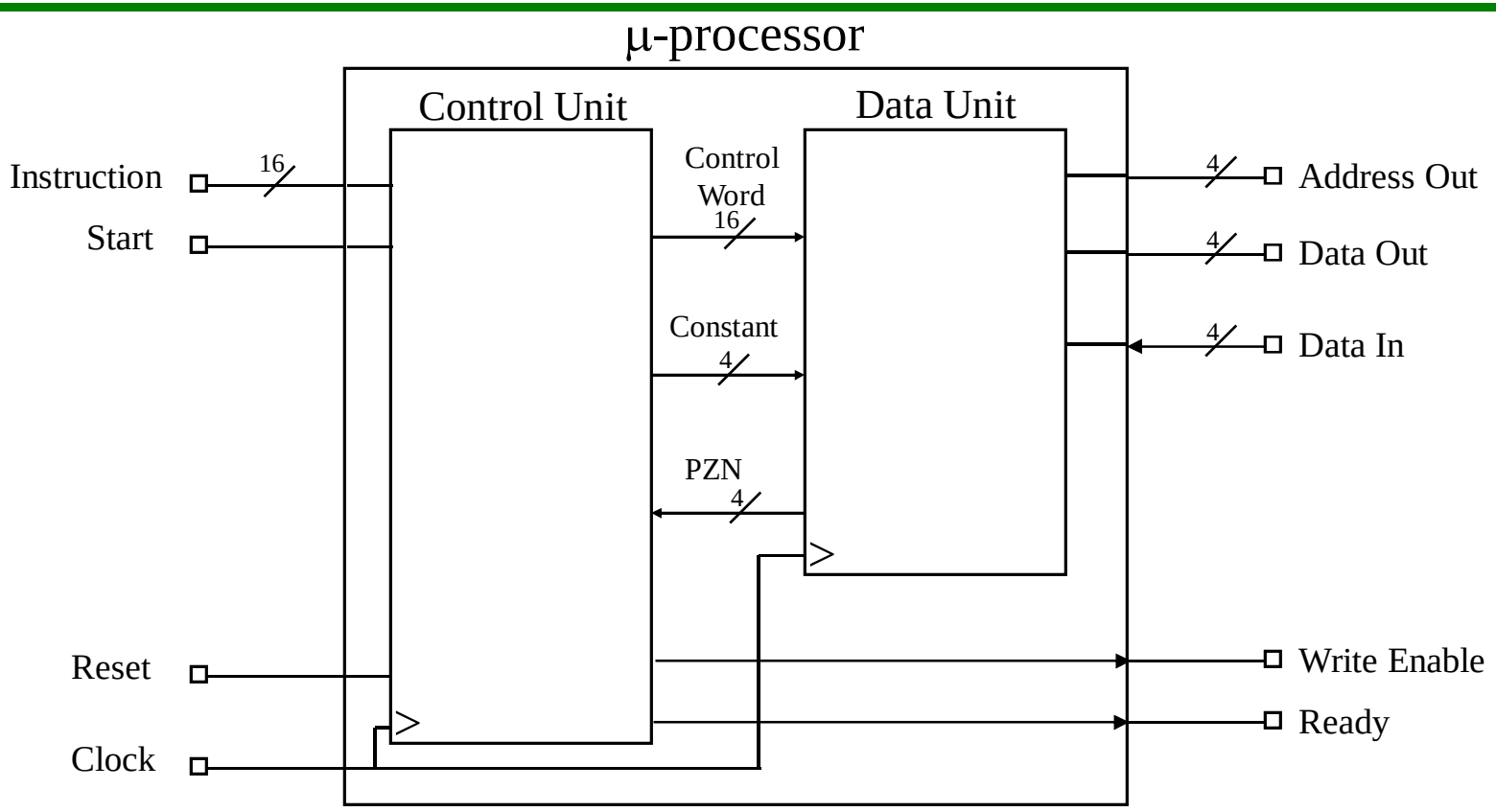
ADDR	DATA
x0	x2
x1	xF
x2	x9
…	…

15:12	11:8	7:4	3:0
OPCODE	OPERAND A	OPERAND B	OPERAND C

OPCODE	INSTRUCTION FUNCTION (RTL)
x0	NOP/IDLE
x1	M[A] ← B
x2	M[A] ← M[B]
x3	M[A] ← M[B] + M[C]
x4	M[A] ← M[B] - M[C]
x5	M[A] ← M[B] * M[C]
x6	M[A] ← M[B] >> 1
x7	M[A] ← NOT (M[B])
x8	M[A] ← M[B] AND M[C]
x9	IF (M[C] = 0) THEN M[A] ← M[B]
xA	IF (M[C] = M[B]) THEN (M[A] ← 0) ELSE (M[A] ← 1)
xB-xF	Reserved for TA madness!

FS	Microoperation
x0	F = 0
x1	F = 1
x2	F = A
x3	F = B
x4	F = A + B
x5	F = A – B
x6	F = A * B
x7	F = A / B
x8	F = NOT(A)
x9	F = A AND B
xA	F = A OR B
xB	F = NOT(A) + 1
xC	F = A >> 1
xD	F = A << 1
xE	F = A >>> 1
xF	Reserved

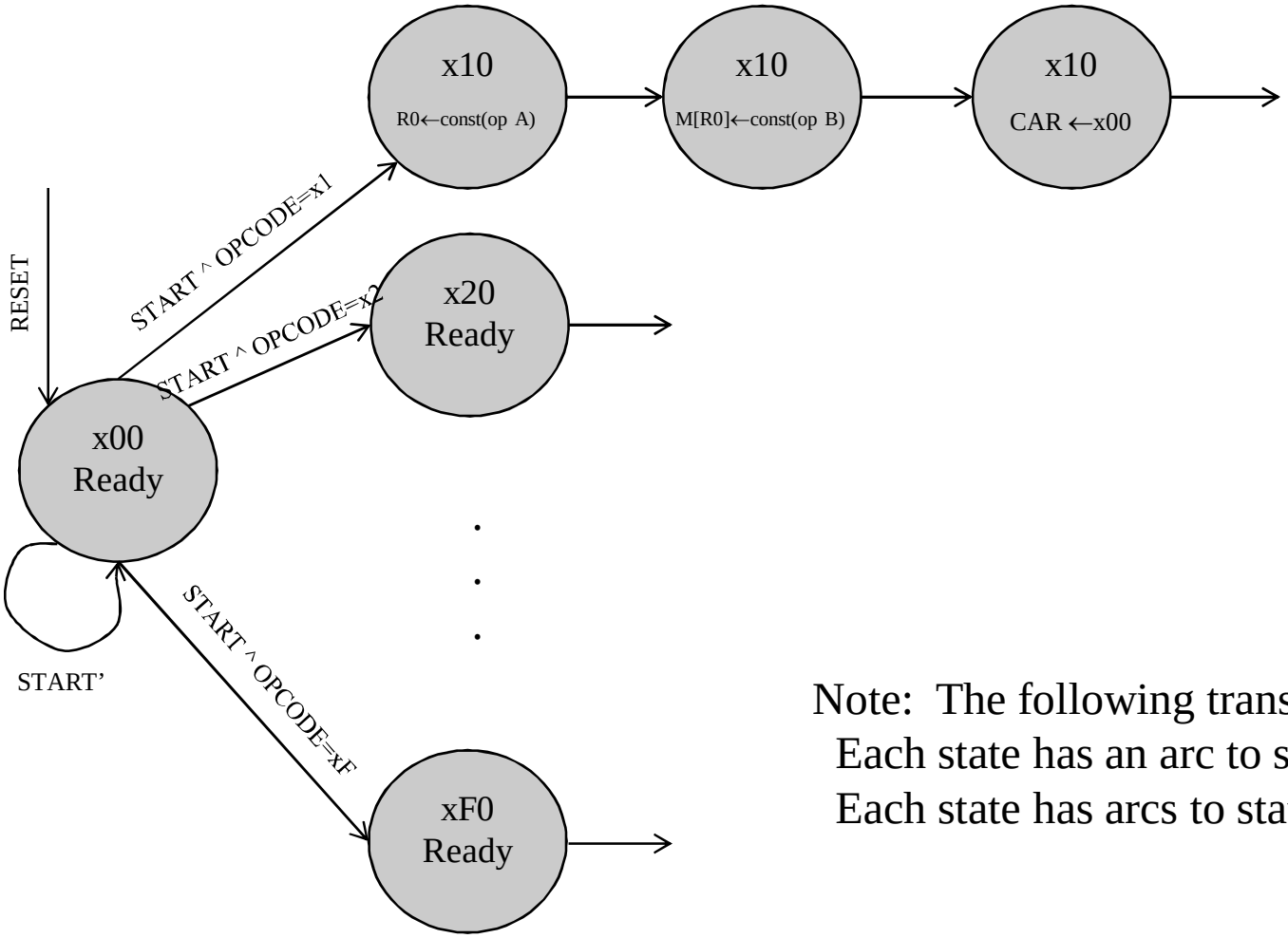
The DDmini microprocessor



Example Instruction: x1021
Opcode 2 Format: M[A] ←B
Instruction: M[x0] ← x2

- Microprogram required!

Diagram for Control



Note: The following transitions are not illustrated
Each state has an arc to state x00 on RESET
Each state has arcs to state x00-xF0 on START

15:12	11:8	7:4	3:0
OPCODE	OPERAND A	OPERAND B	OPERAND C

OPCODE	INSTRUCTION FUNCTION (RTL)
x0	NOP/IDLE
x1	M[A] ← B
x2	M[A] ← M[B]
x3	M[A] ← M[B] + M[C]
x4	M[A] ← M[B] - M[C]
x5	M[A] ← M[B] * M[C]
x6	M[A] ← M[B] >> 1
x7	M[A] ← NOT (M[B])
x8	M[A] ← M[B] AND M[C]
x9	IF (M[C] = 0) THEN M[A] ← M[B]
xA	IF (M[C] = M[B]) THEN (M[A] ← 0) ELSE (M[A] ← 1)
xB-xF	Reserved for TA madness!

FS	Microoperation
x0	F = 0
x1	F = 1
x2	F = A
x3	F = B
x4	F = A + B
x5	F = A – B
x6	F = A * B
x7	F = A / B
x8	F = NOT(A)
x9	F = A AND B
xA	F = A OR B
xB	F = NOT(A) + 1
xC	F = A >> 1
xD	F = A << 1
xE	F = A >>> 1
xF	Reserved

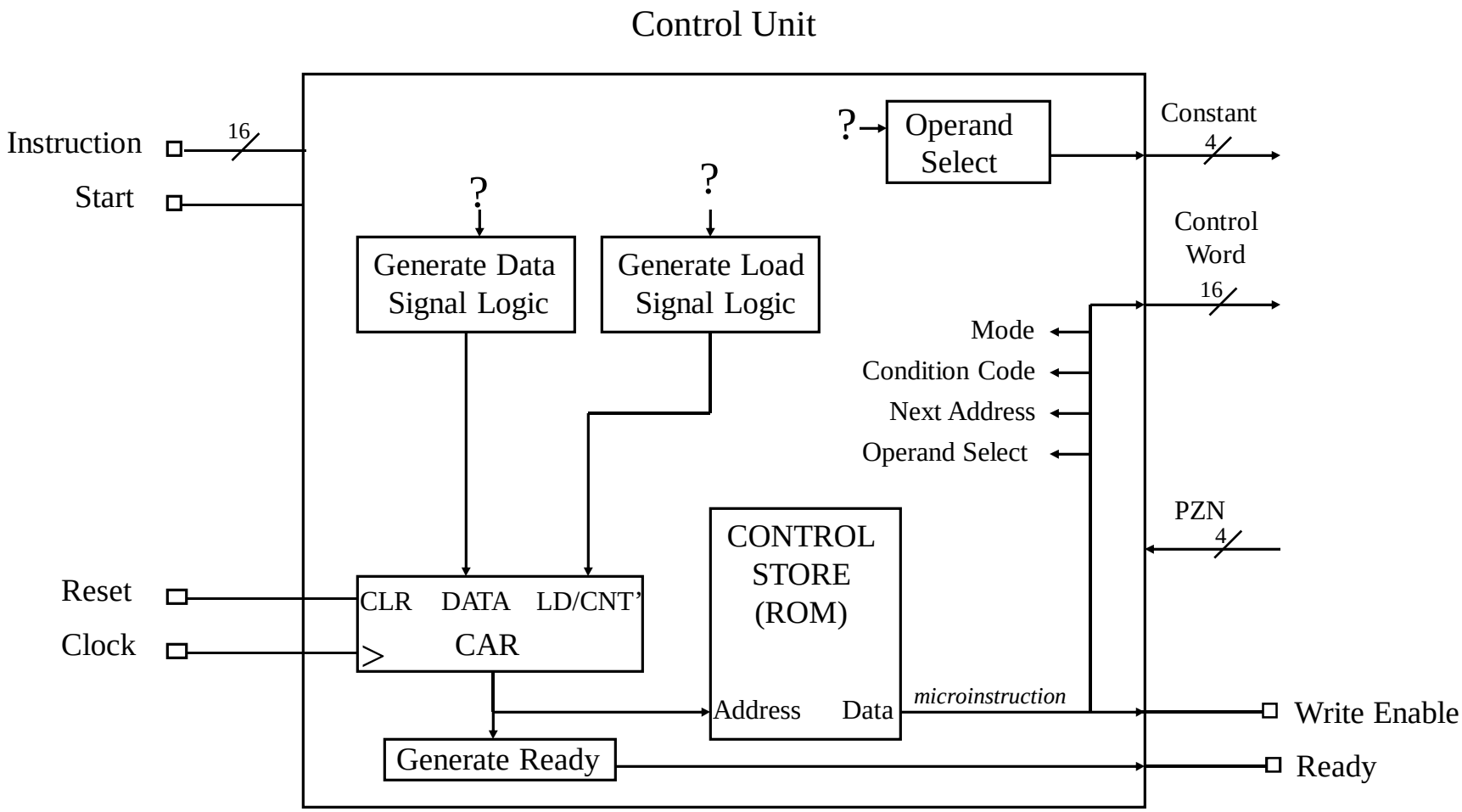
17	16:15	14:12	11:4	3:0
Mode=1	00	CONDITION (3)	NEXT_ADDRESS (8)	0000

17	16	15:0
Mode=0	MW	Control Word (16)

15:13	12	11:9	8	7:4	3:1	0
AA (3)	MB (1)	BA (3)	MF (1)	FS (4)	DA (3)	RW (1)

BA	000	001	010	011
CONSTANT_OUT	x0	OP A	OP B	OP C

DDmini Control Unit



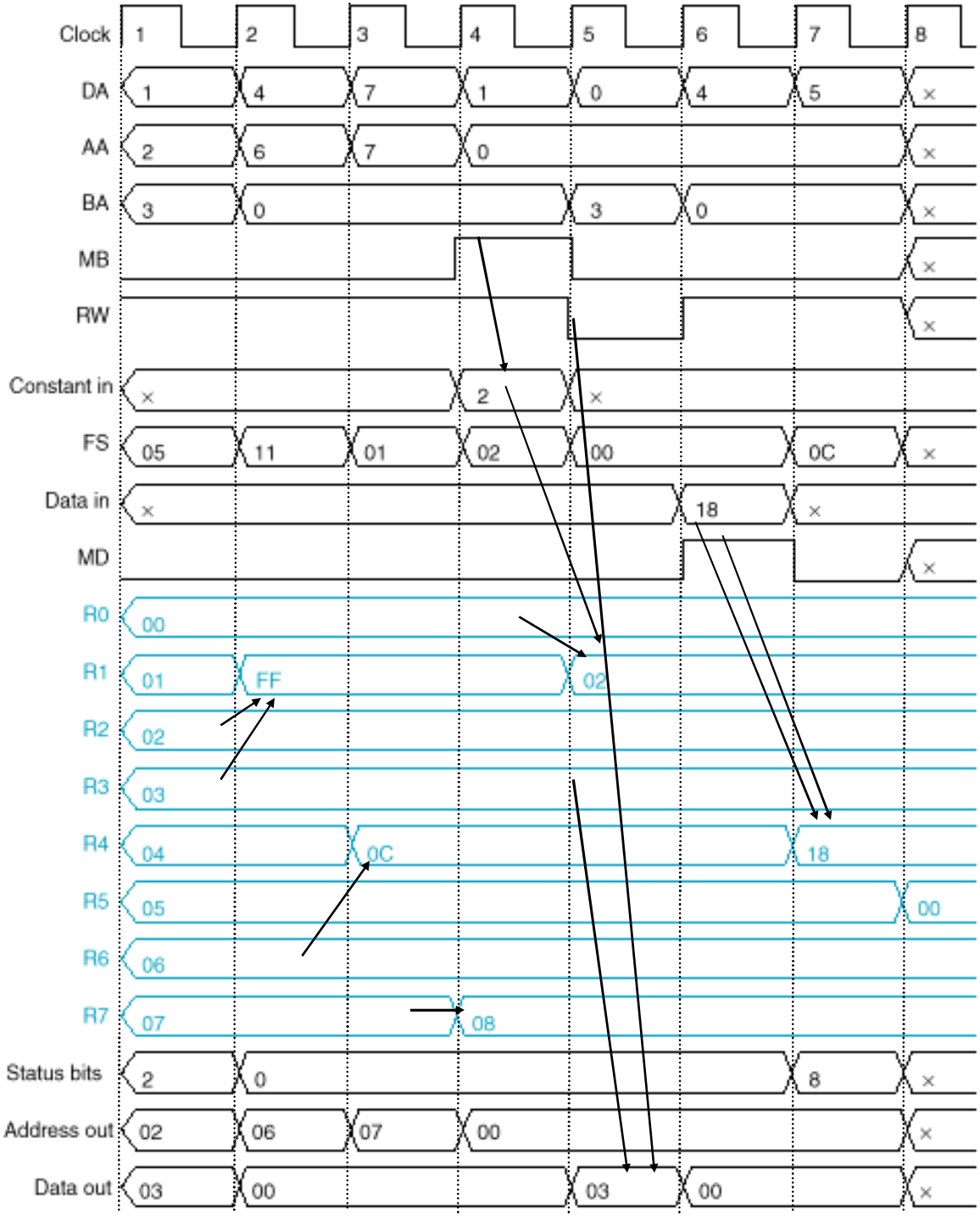
Pipelining

Pipelining the Datapath

Data hazards

Control hazards

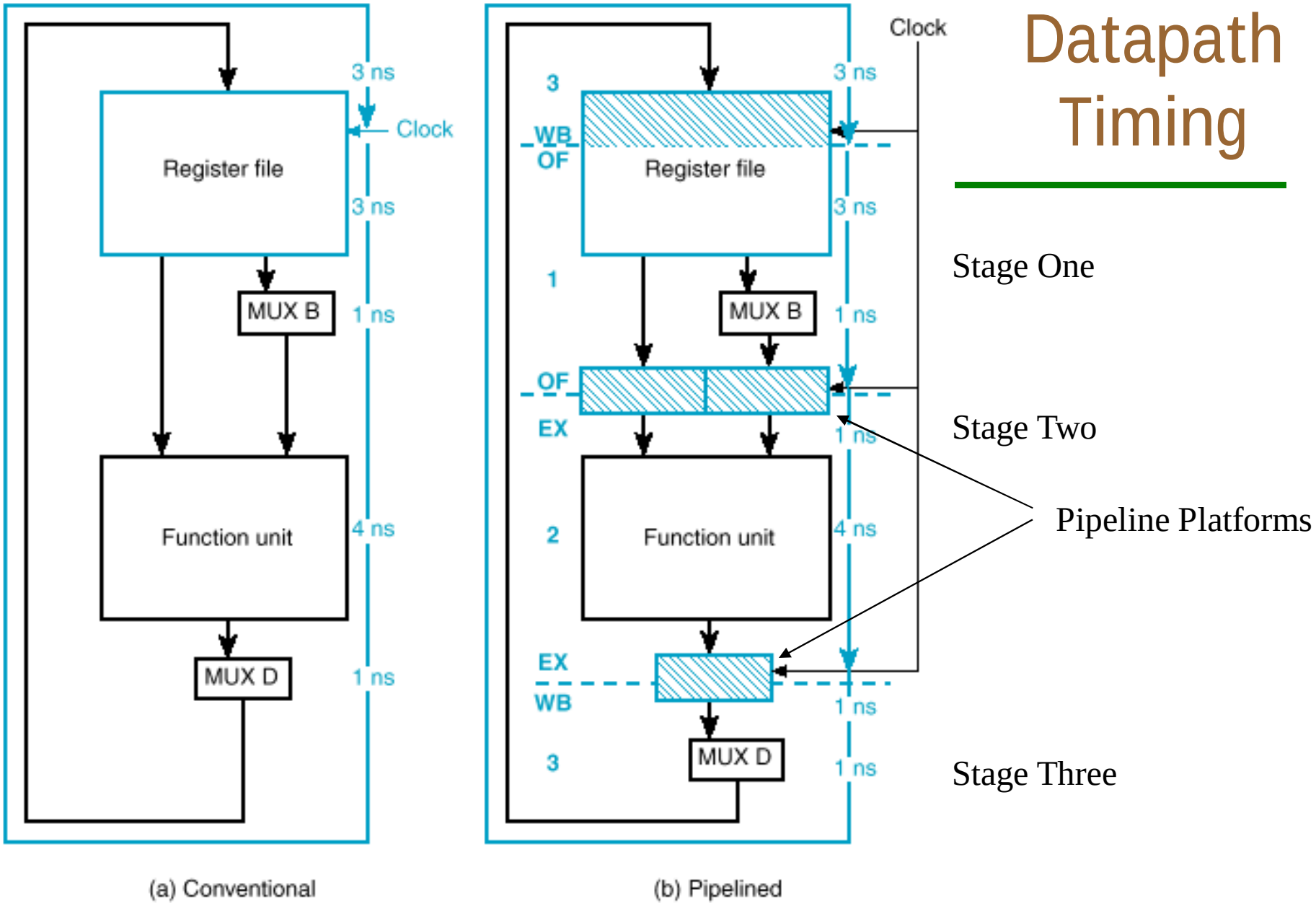
Timing Diagram



Pipelined Datapath

- Datapath throughput can be increased by breaking up the datapath with registers and increasing the clock speed.
  - This is known as pipelining the datapath.
- Throughput vs. Turnaround time
  - Throughput: The number of operations (units) per second (time period).
  - Turnaround time: The amount of time which elapses from the beginning of the operation's execution (unit processing) to its completion.
  - Consider an assembly line (or fast-food drive-thru!)
- Basic stages of a datapath microoperation:
  - Operand Fetch (OF)
  - Execute Function (EX)
  - Write Back Results (WB)

Datapath Timing



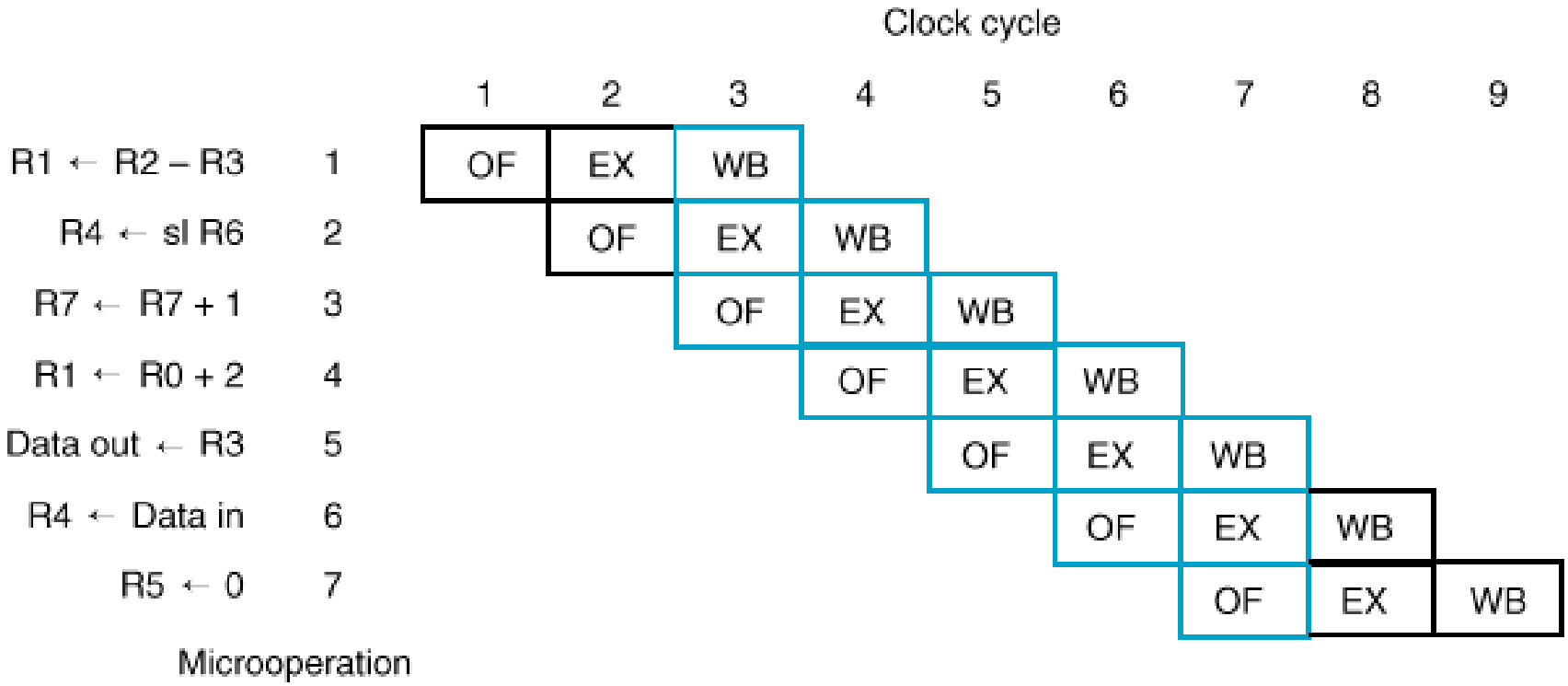
Min. Clock Period = 12 ns

83.3 MHz

Min. Clock Period = 5 ns

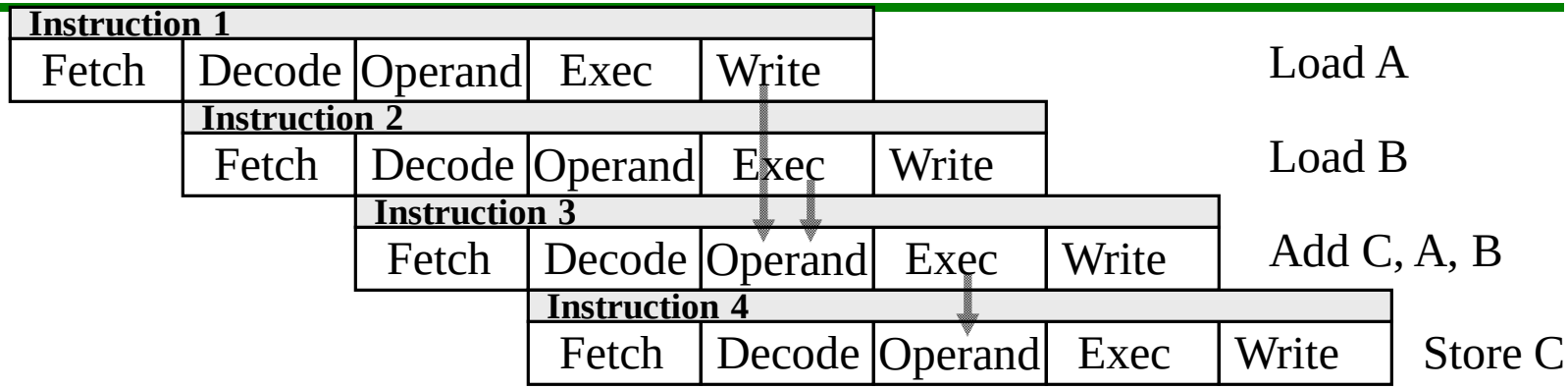
200 MHz

Pipeline Execution Pattern



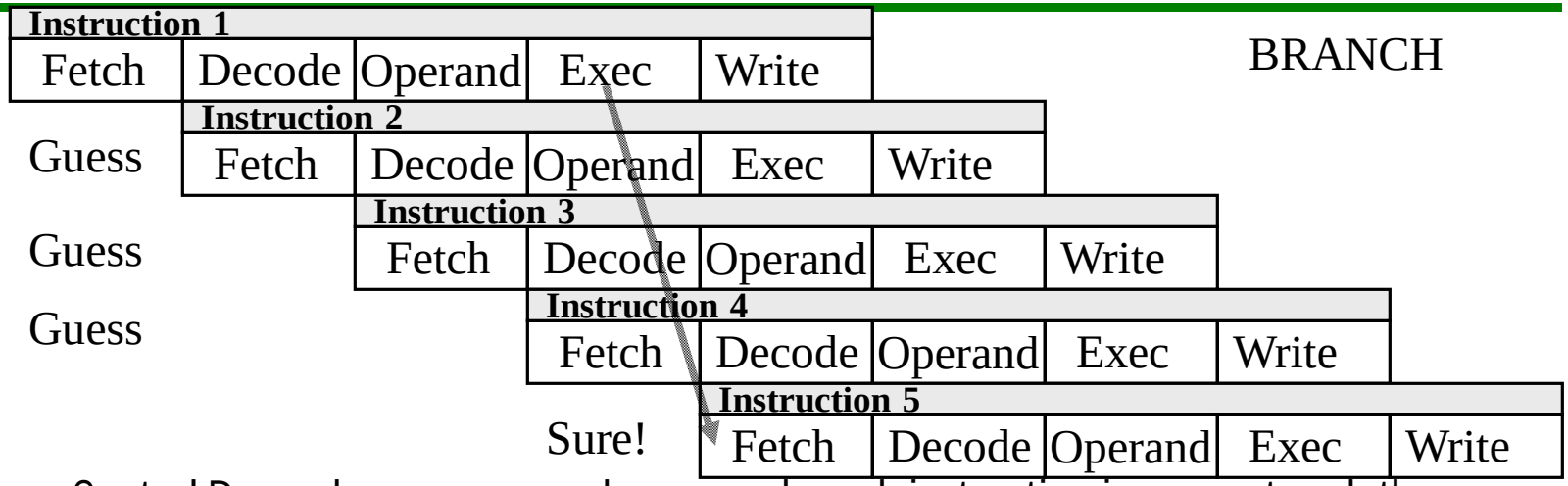
- Not all of the pipelined units are necessarily active at all times
  - Filling and emptying the pipeline “wastes” time
  - Hazards exist!

Pipeline Hazards (1)



- Basic Pipelining Hazards
  - Data Dependency - occurs whenever one instruction generates a value that is used by one or more successive instructions.
    - Data Forwarding - a technique to avoid delays due to data dependency hazards by allowing data to skip one or more pipeline stages

Pipeline Hazards (2)



– Control Dependency - occurs whenever a branch instruction is encountered; the address of the "next" instruction can not be determined until the branch conditions is evaluated during the execute phase.
  - Branch Prediction - reduces delays due to control dependencies by making a educated guess and "anulling" the operations in the pipeline if the guess is incorrect.
  - Most branches backwards are "taken", most branches forward are "not taken".
  - 2-bit history (traditional technique): If the guess is wrong twice in a row, change the future guess for that instruction in the future.
  - Trace caches: proposed designs use instruction traces to keep track of past program flow. Some proposals use 50% of the chip area for this cache. Why?

Contemporary VLSI Design

Modern (VLSI) Design

- The VLSI chips that are used in most modern designs come in three varieties:
  - Custom Approach: VLSI chips, or some of their parts, are designed by hand.
    - Full Custom Vs. Standard Cell - Using standard cell designs (same height, variable width) and routing channels simplifies the design process
    - Highest Density, Highest Manufacturing Cost
  - Semicustom Approach: The VLSI chips employ gate arrays and technology mapping.
    - **Gate array**: a partially prefabricated IC that incorporates a large number of identical gates (usually 3-input NAND or NOR gates) that are laid out in a regular two-dimensional array.
    - **Technology mapping**: The process of designing a logic function as a network of available devices.
    - Lower Density (10-25% more gates than an equivalent custom design).
    - Inexpensive: Requires only metal deposition (define interconnections), economy of scale.

Modern (VLSI) Design

- The VLSI chips that are used in most modern designs come in three varieties:
  - VLSI PLDs
    - Field Programmable Gate Arrays (FPGAs)
    - A VLSI modules that can be programmed to implement a digital system consisting of tens of thousands of gates.
    - LSI PLDs implement two-level combinational and sequential networks,
    - FPGAs allow the realization of multilevel networks and complex systems on a single chip!
    - Highly reprogrammable!
    - Low cost
    - May produce slower network
    - May require a larger silicon area