Central Processing Unit

Steven R. Bagley

Introduction

- So far looked at the technology underpinning computers
- Logic signals to cause things to happen, and represent numbers
- Boolean gates to combine and manipulate those signals
- Flip-Flops/RAM to store things

The CPU

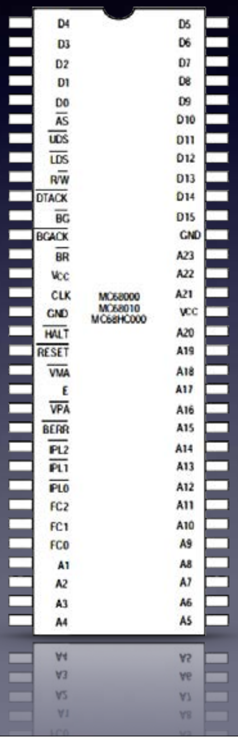
- Central Processing Unit — the ‘brain’ of the computer
- Constructed out of logic gates and flip-flops
- Talks to the rest of the system by sending logic signals on its various pins…

except the CPU is very dumb

68000

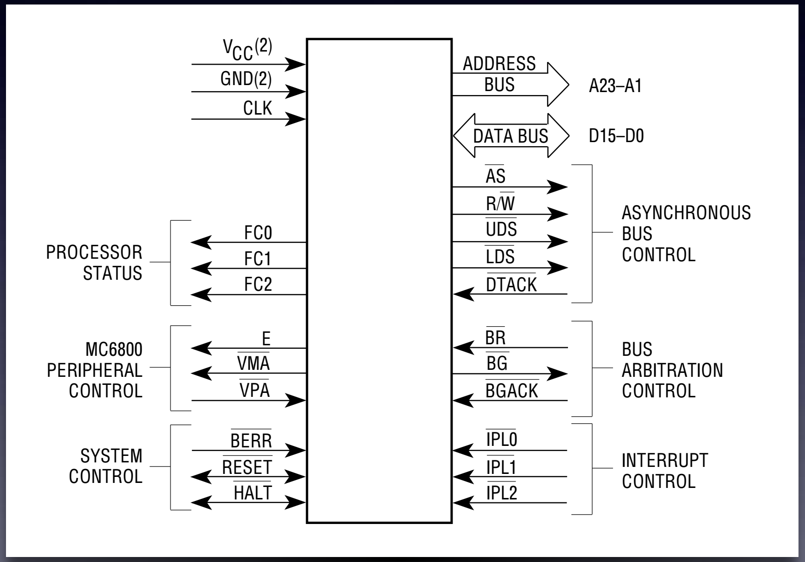
This is the 68000 processor by Motorola, a typical 16-bit CPU from the 1980s. The pinout is typical of most CPUs, although things aren't as straightforward now…

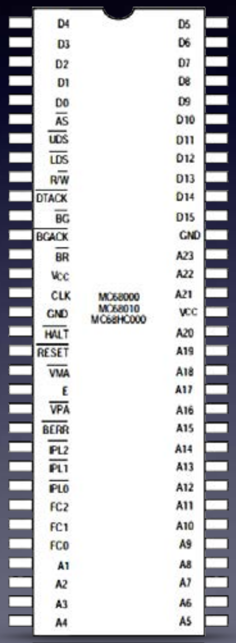
68000



This is the 68000 processor by Motorola, a typical 16-bit CPU from the 1980s. The pinout is typical of most CPUs, although things aren’t as straightforward now…

68000





This is the 68000 processor by Motorola, a typical 16-bit CPU from the 1980s. The pinout is typical of most CPUs, although things aren't as straightforward now…

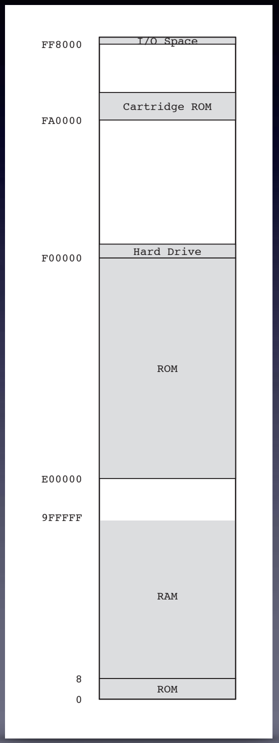
CPU-System Communication

- Main signals used to communicate with the system are:
  - Address Bus
  - Data Bus
  - Read/Write
- Easier to understand if we consider how they are used

Memory

- Memory is just a collection of flip-flops
- Usually arranged into groups that can be addressed individually
- Often multiples of eight (i.e.bytes)
- By an address, just a binary number…
- Not just RAM…

Example Memory Map



From Atari STe — not to scale!

Also ROM, and lots of computers use what's called Memory-mapped IO. Hardware devices appear as if they were parts of the computer's memory

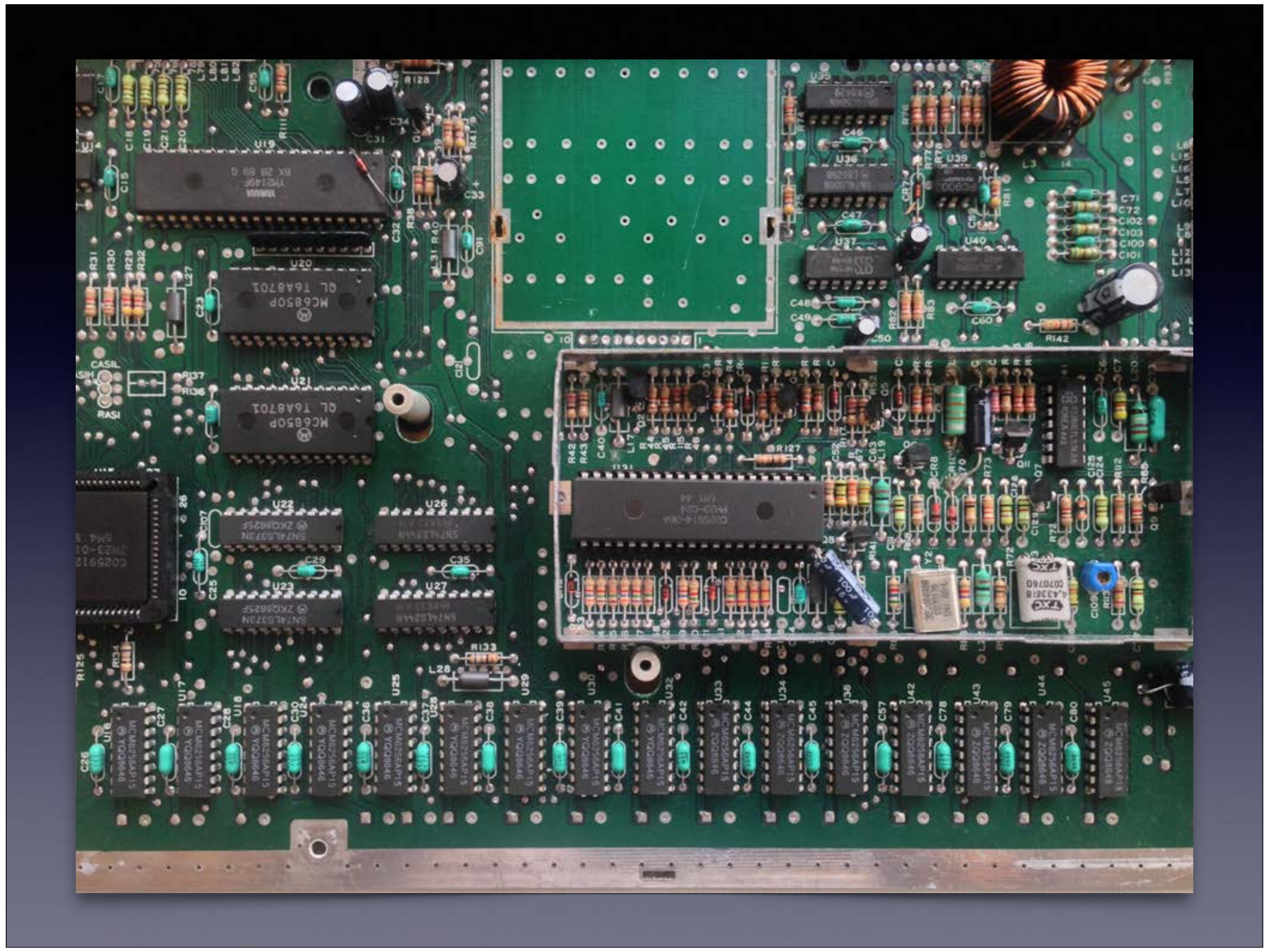


PHoto of 16 ram chips, one for each word in the ATARI st

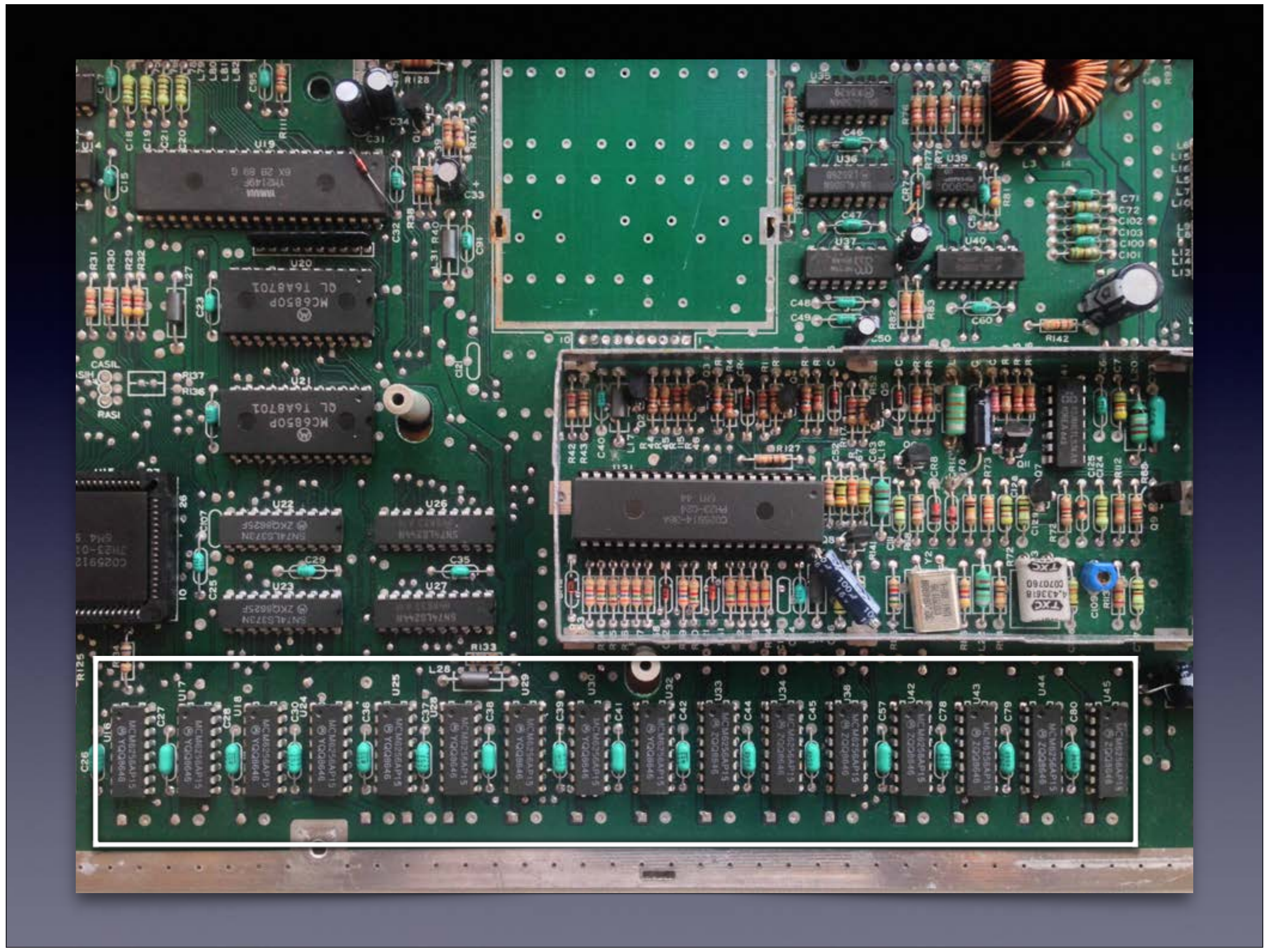
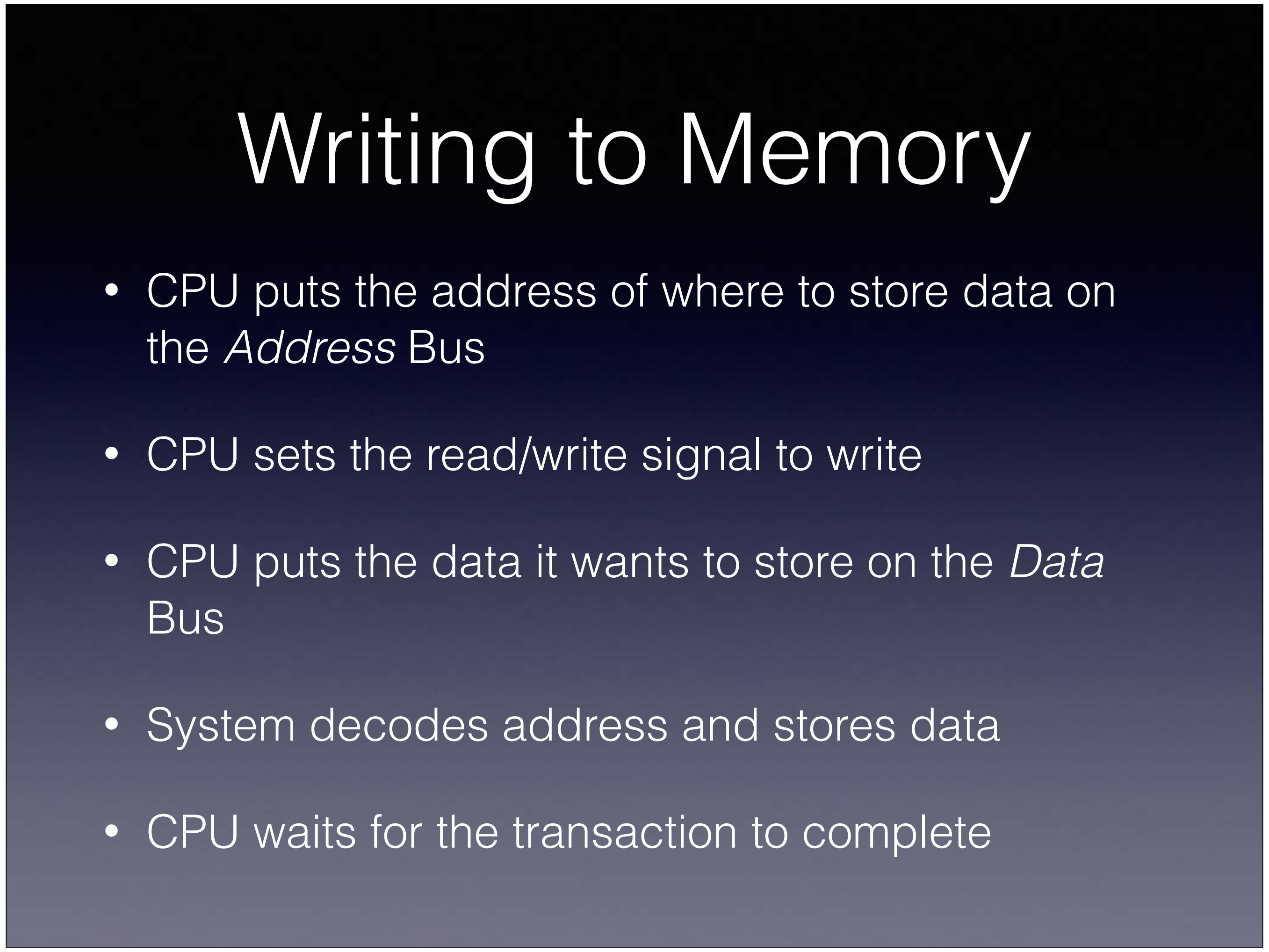


PHoto of 16 ram chips, one for each word in the ATARI st



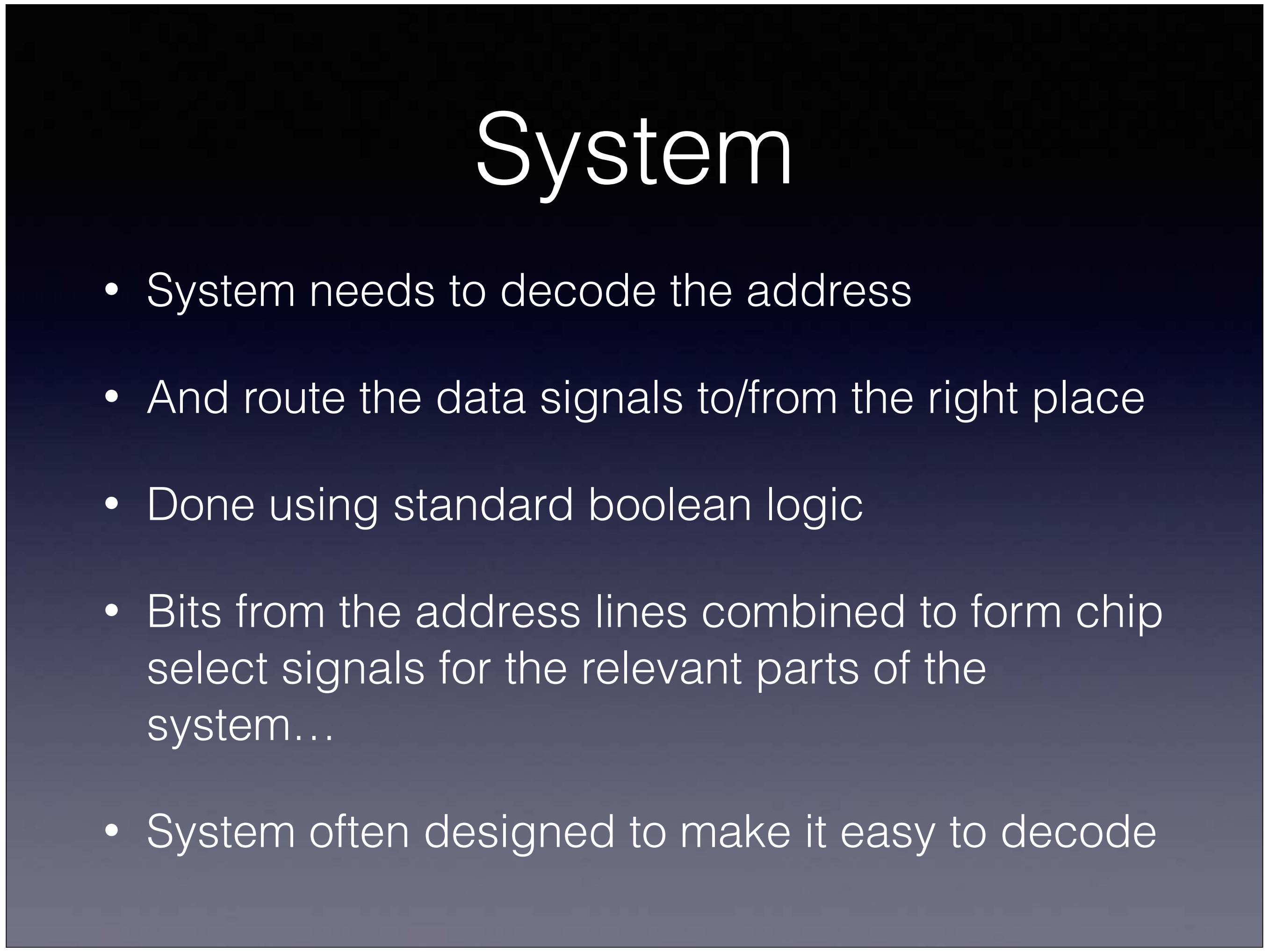
Writing to Memory

- CPU puts the address of where to store data on the *Address* Bus
- CPU sets the read/write signal to write
- CPU puts the data it wants to store on the *Data* Bus
- System decodes address and stores data
- CPU waits for the transaction to complete

How transaction finishing is done depends on the CPU — with the 68000, there's a 'Data Transfer ACKnowledge' signal (DTACK)

Reading from Memory

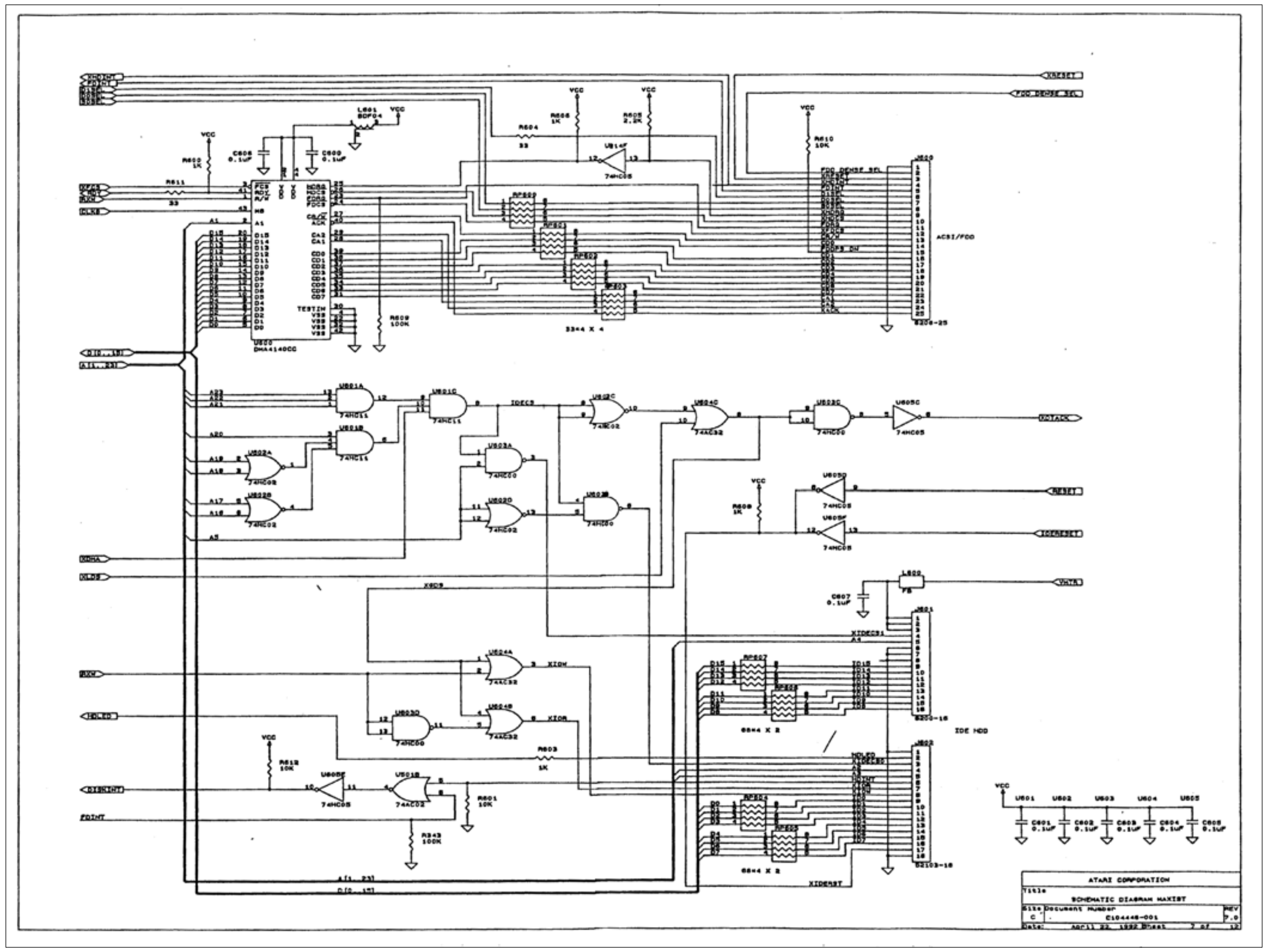
- CPU sets the read/write signal to read
- CPU puts the address of where of data to be read from on *Address* Bus
- System decodes address and places data on *Data* bus
- CPU waits for the transaction to complete and then reads data from *Data* bus



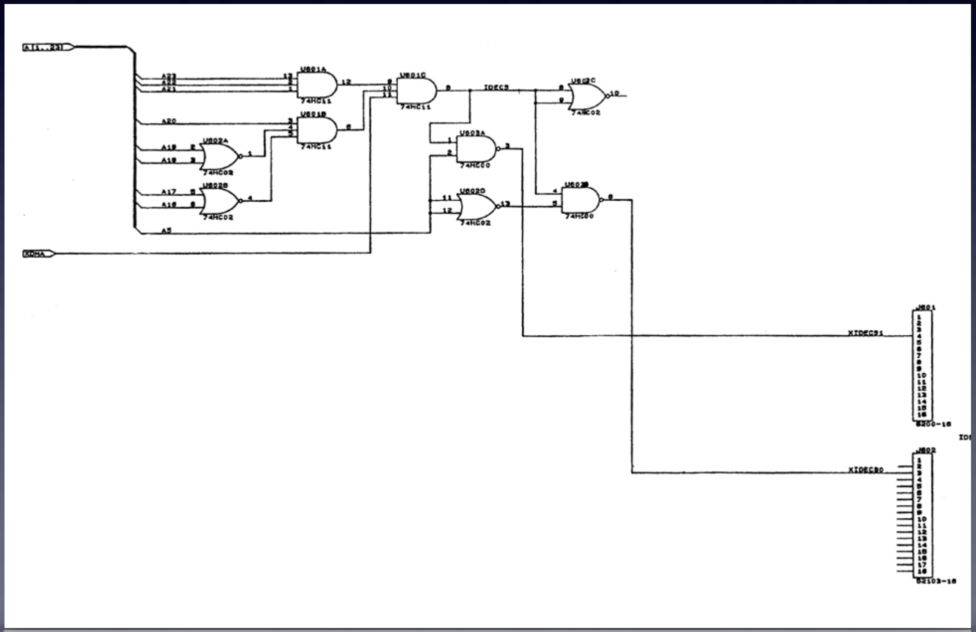
System

- System needs to decode the address
- And route the data signals to/from the right place
- Done using standard boolean logic
- Bits from the address lines combined to form chip select signals for the relevant parts of the system…
- System often designed to make it easy to decode

Saw this last week with the ST hard drive interface

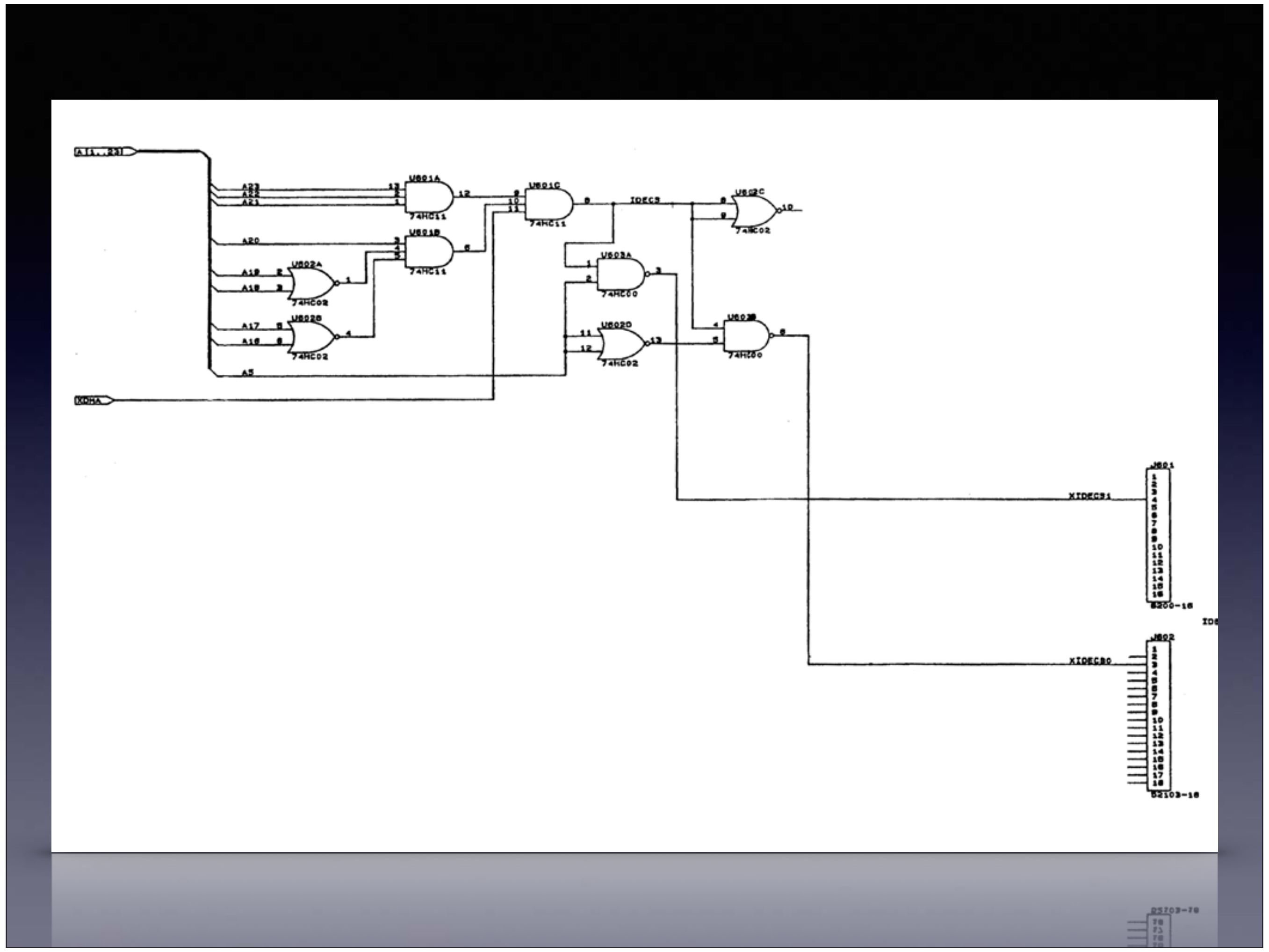


Example of the ATARI ST Book’s IDE hard drive circuit



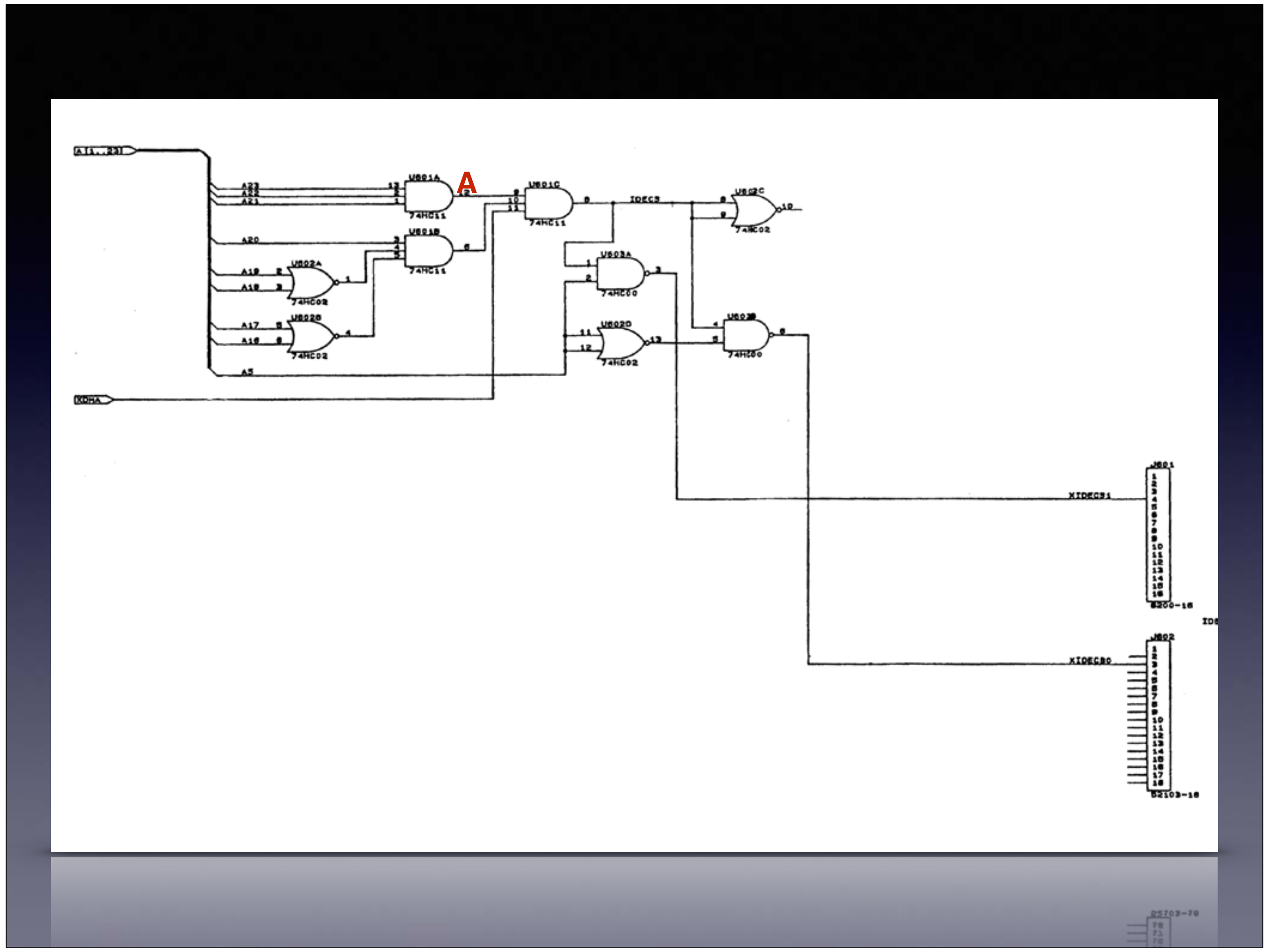
Work out what is happening at each gate connected to the input

Then combine those gates

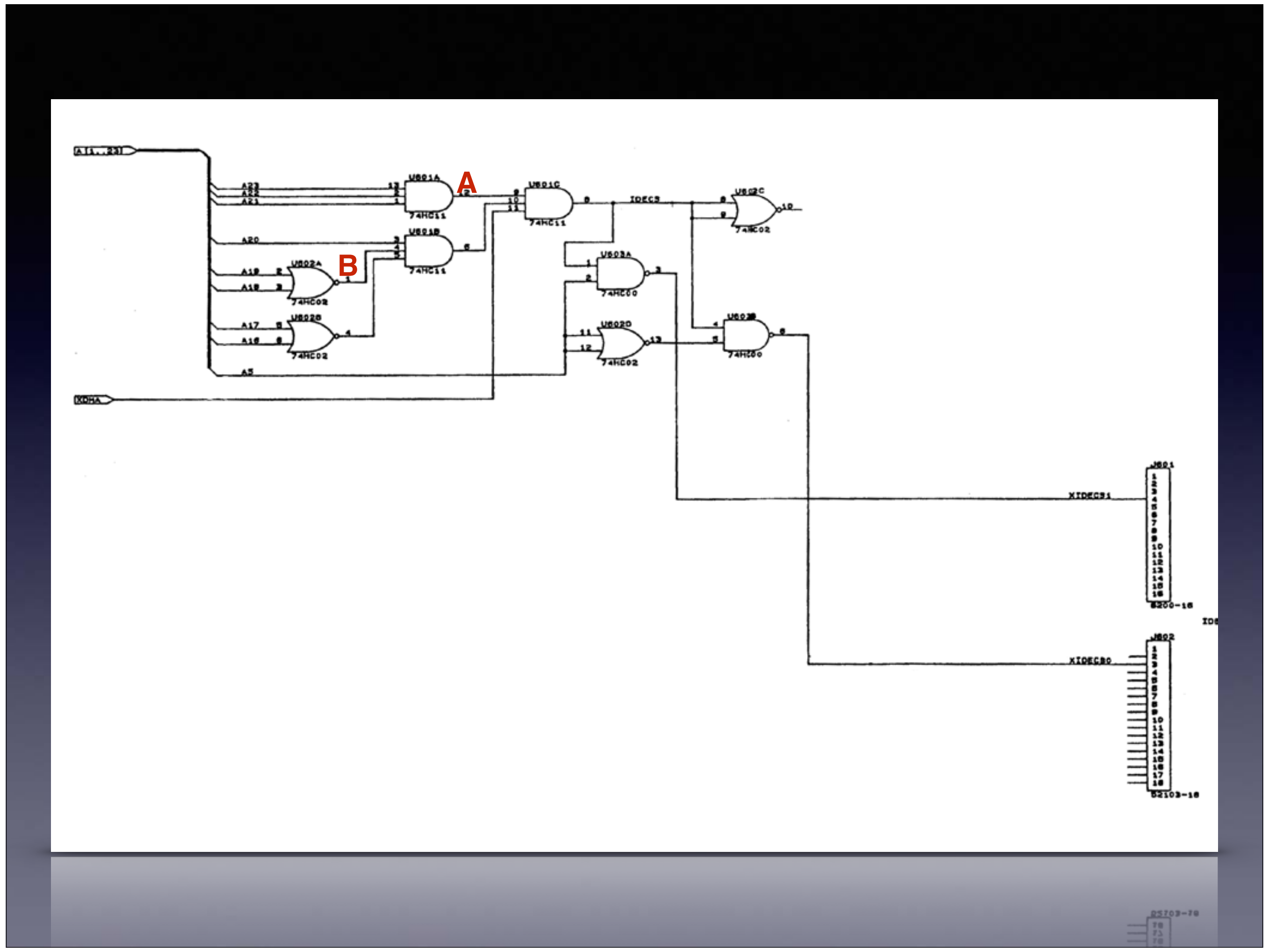


Work out what is happening at each gate connected to the input

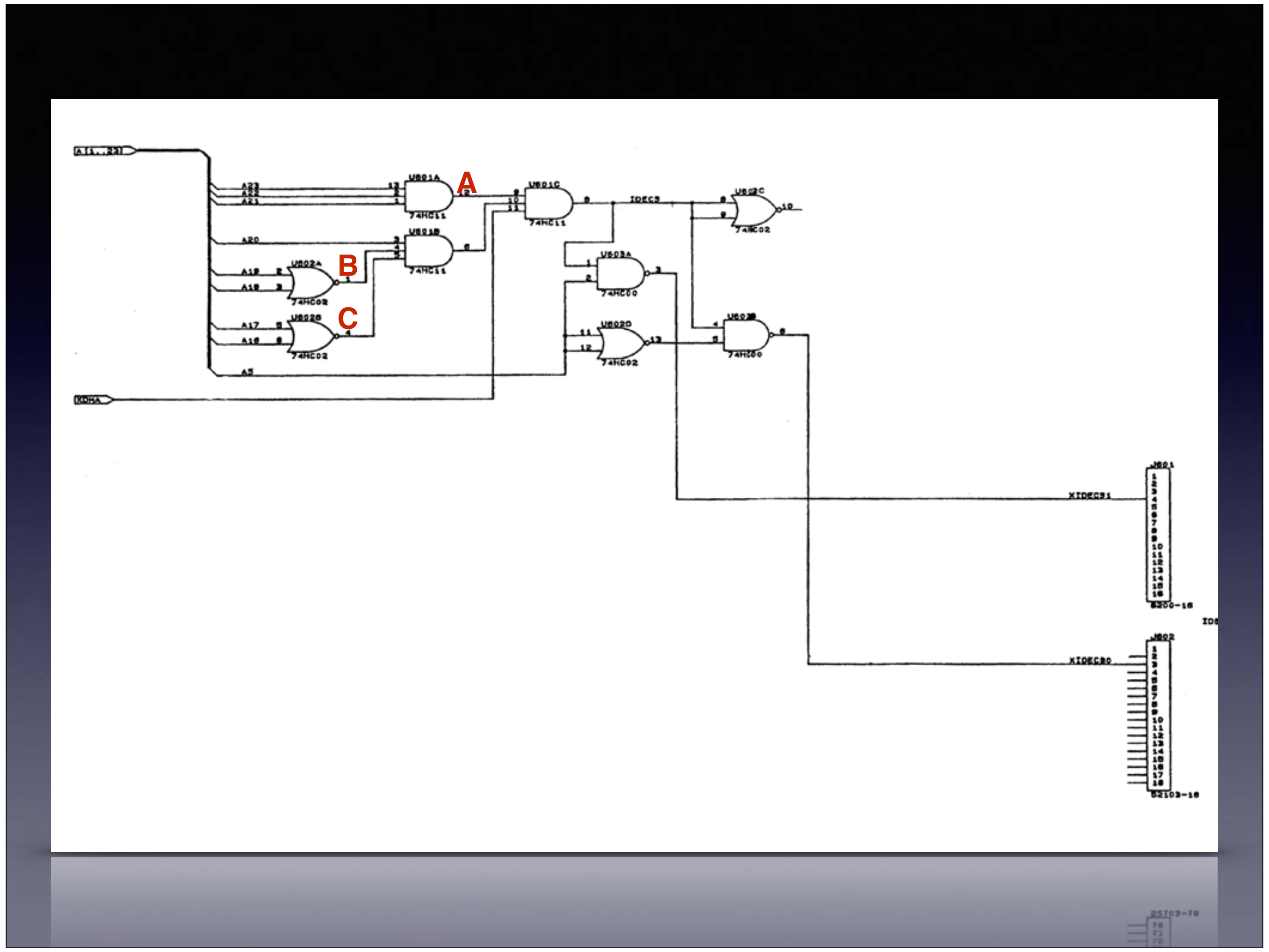
Then combine those gates



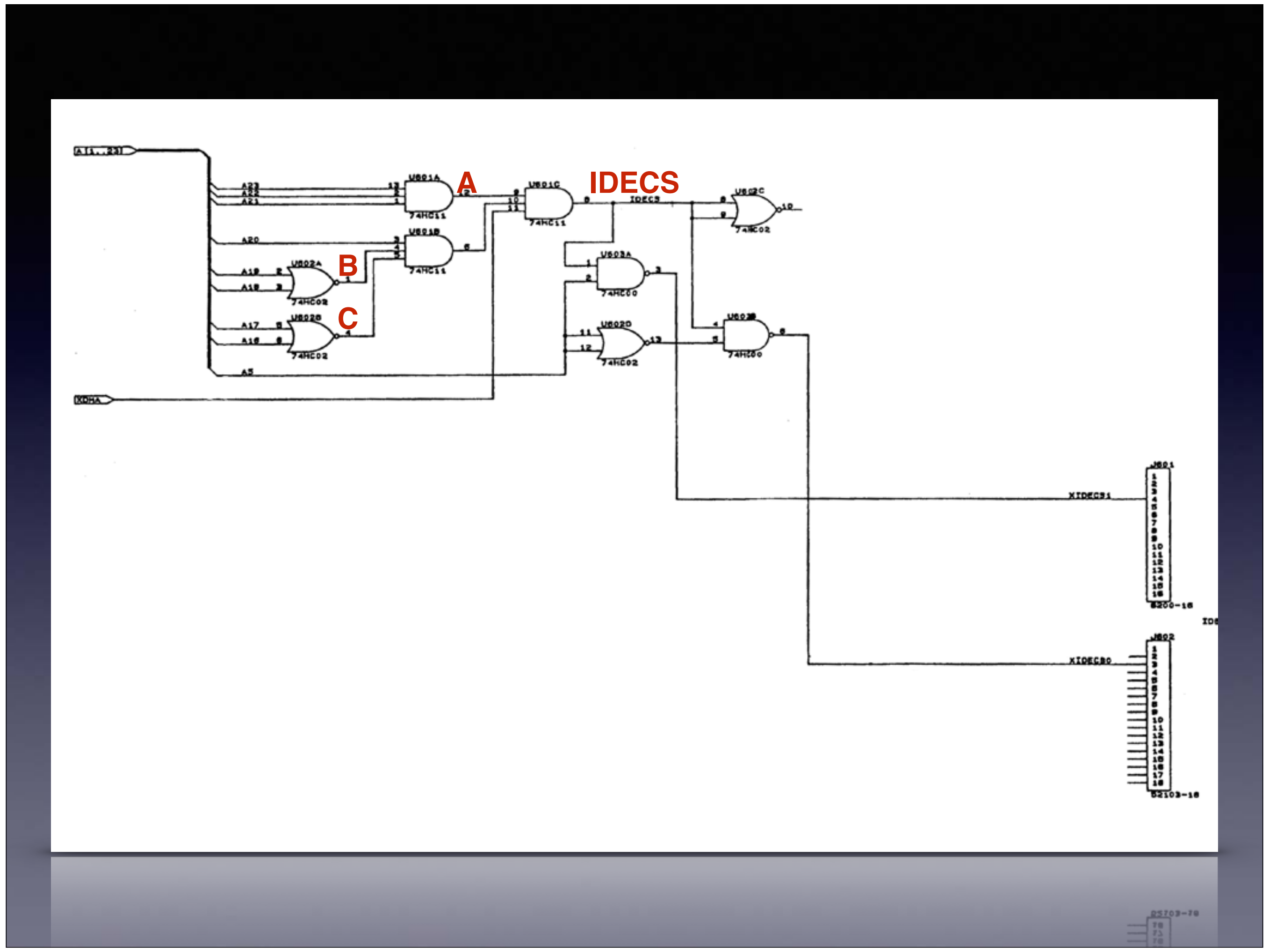
Build up on paper the logic equations for the various points



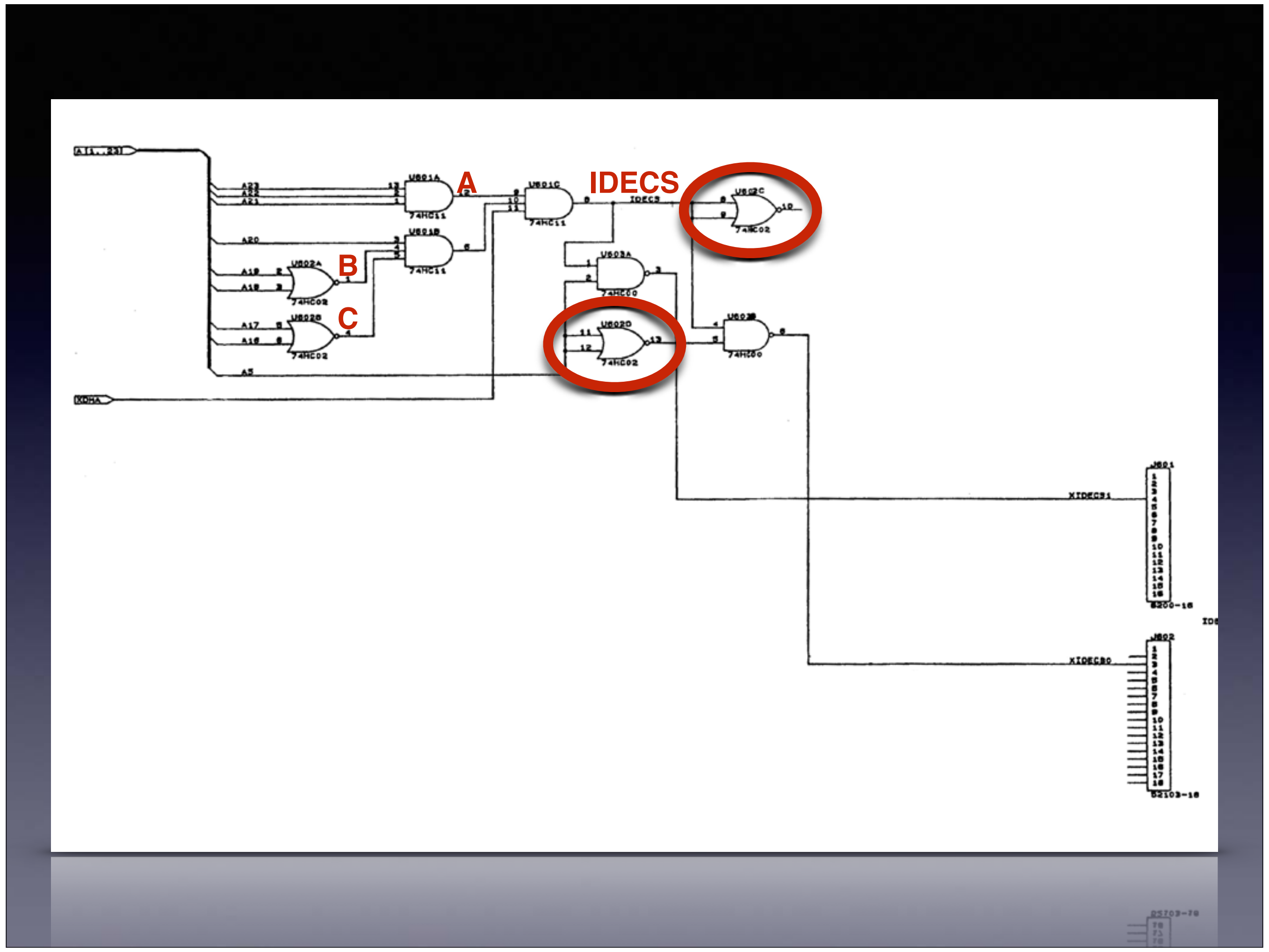
Build up on paper the logic equations for the various points



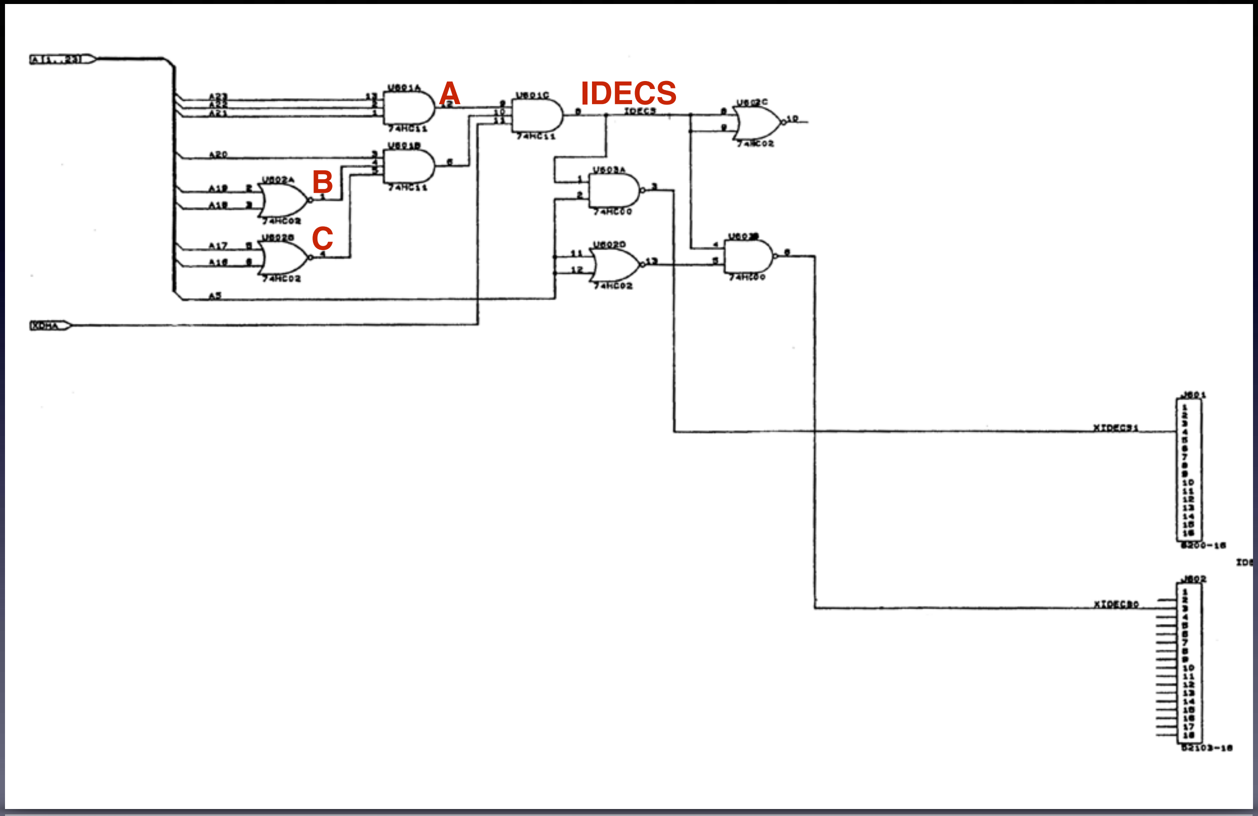
Build up on paper the logic equations for the various points



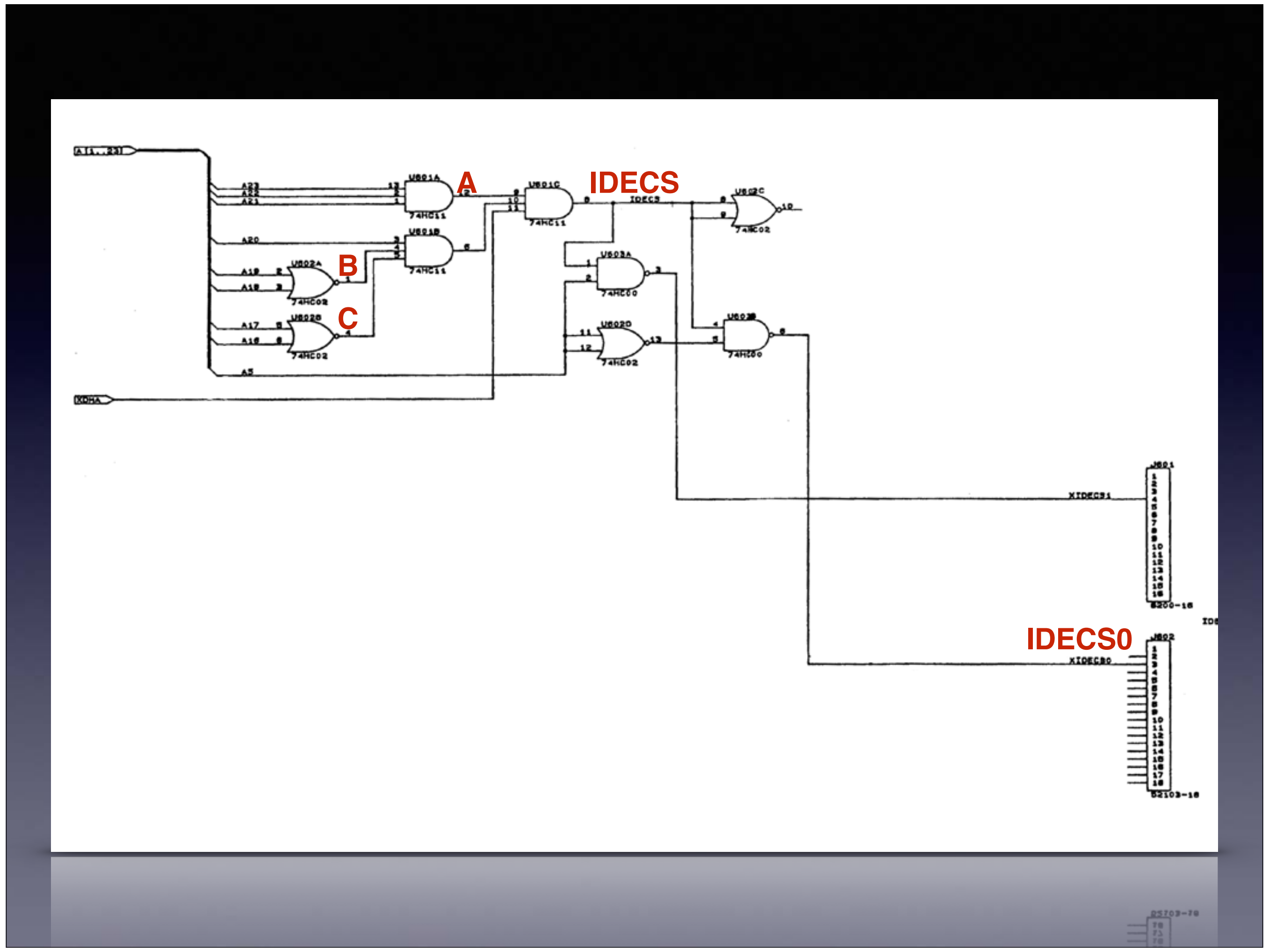
Build up on paper the logic equations for the various points



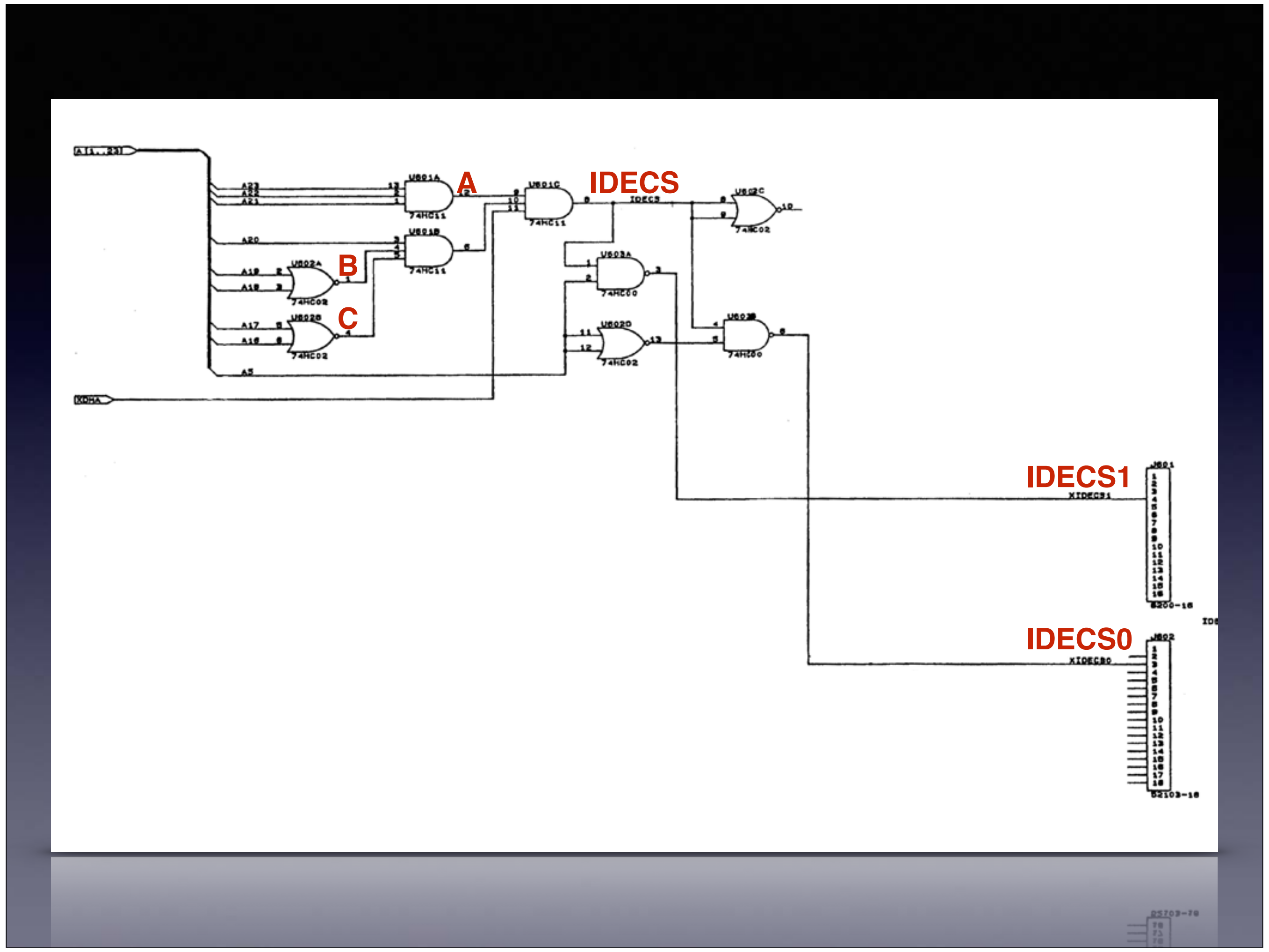
Build up on paper the logic equations for the various points



Build up on paper the logic equations for the various points



Build up on paper the logic equations for the various points



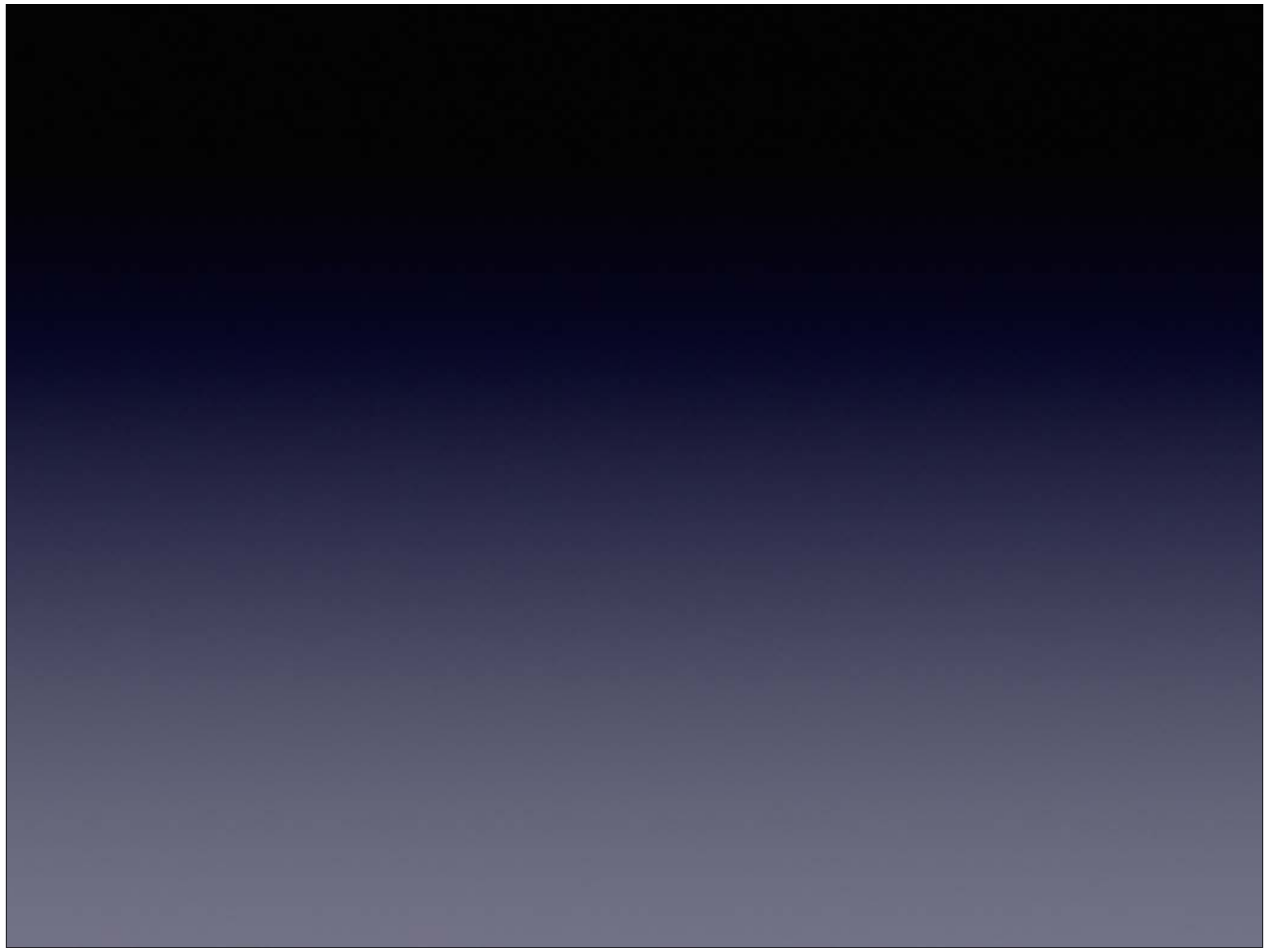
Build up on paper the logic equations for the various points

CPU

- But how does the CPU know what to do?
- It executes a sequence of instructions
- Each instruction tells the CPU to do one thing
- But where does it get these instructions from?

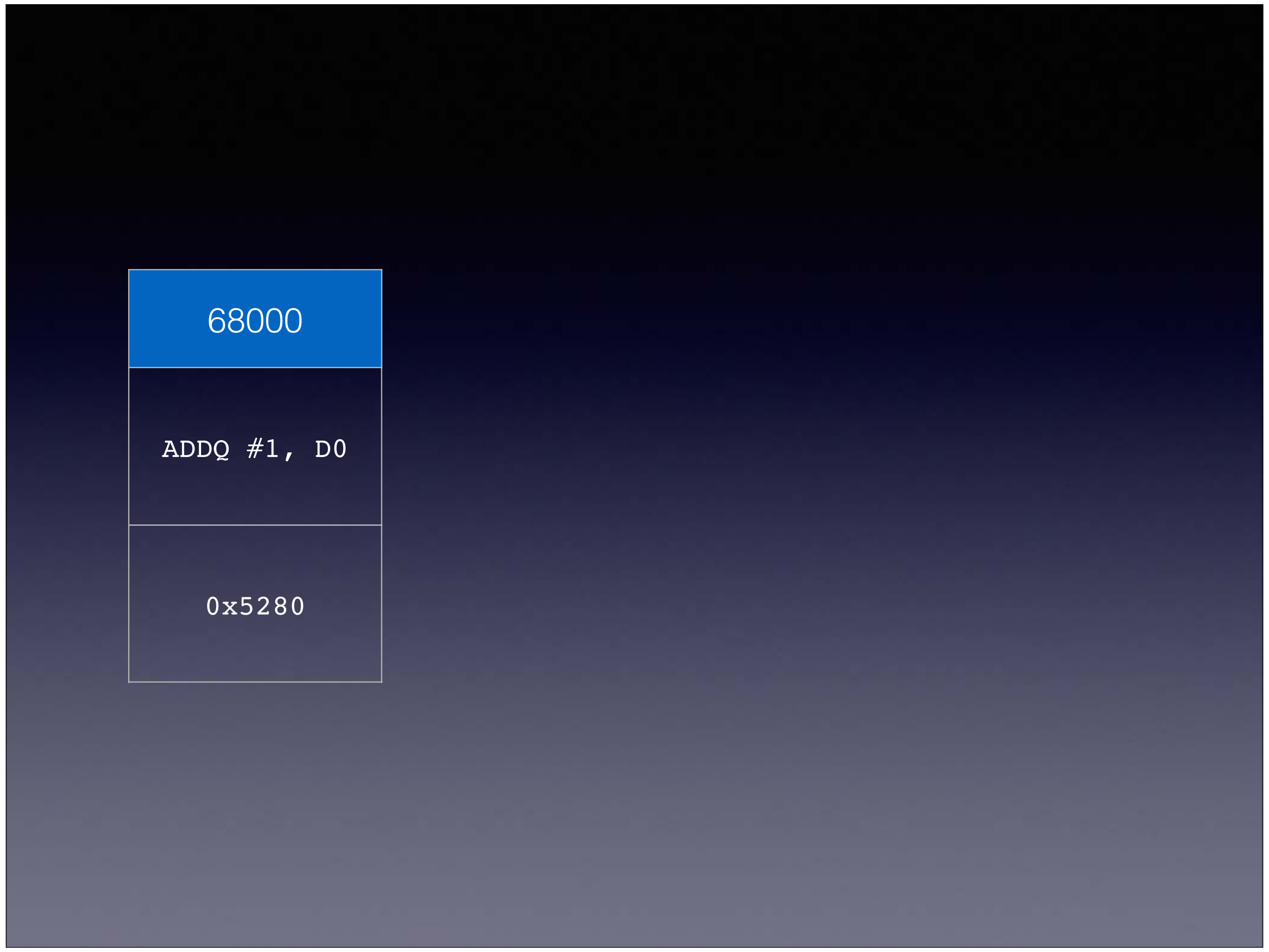
CPU Instructions

- CPU gets instructions from the computer's memory
- Each instruction is encoded as a binary pattern (an opcode)
- Each CPU family has its own *Instruction Set*
- Which are usually incompatible with other CPU families



4 different CPUs instruction to increment a register. Although in most cases not the only way you can do it.

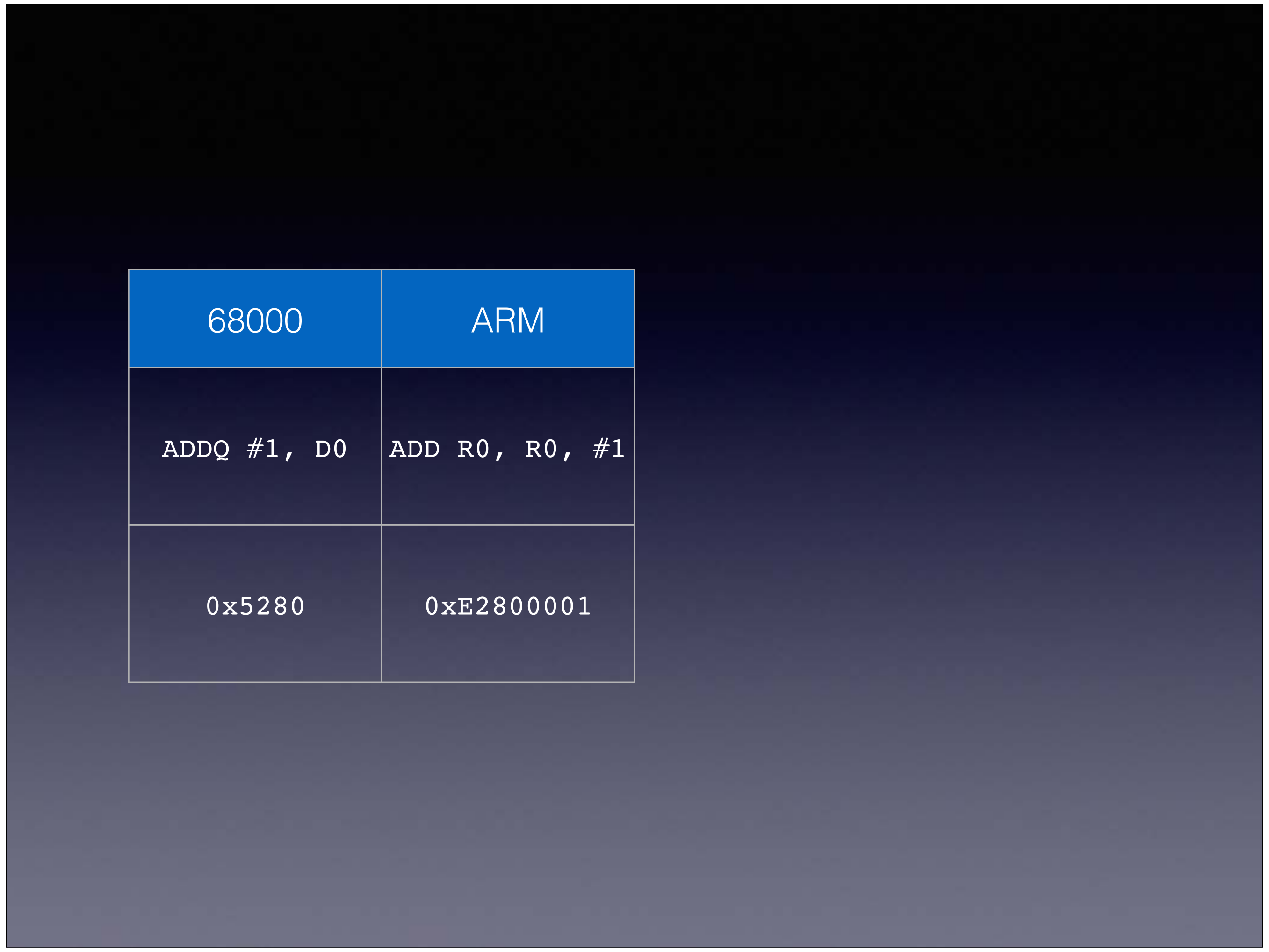
I’m not even going to attempt to work out what the value would be on the x86 — probably around 0x40



68000
`ADDQ #1, D0`
`0x5280`

4 different CPUs instruction to increment a register. Although in most cases not the only way you can do it.

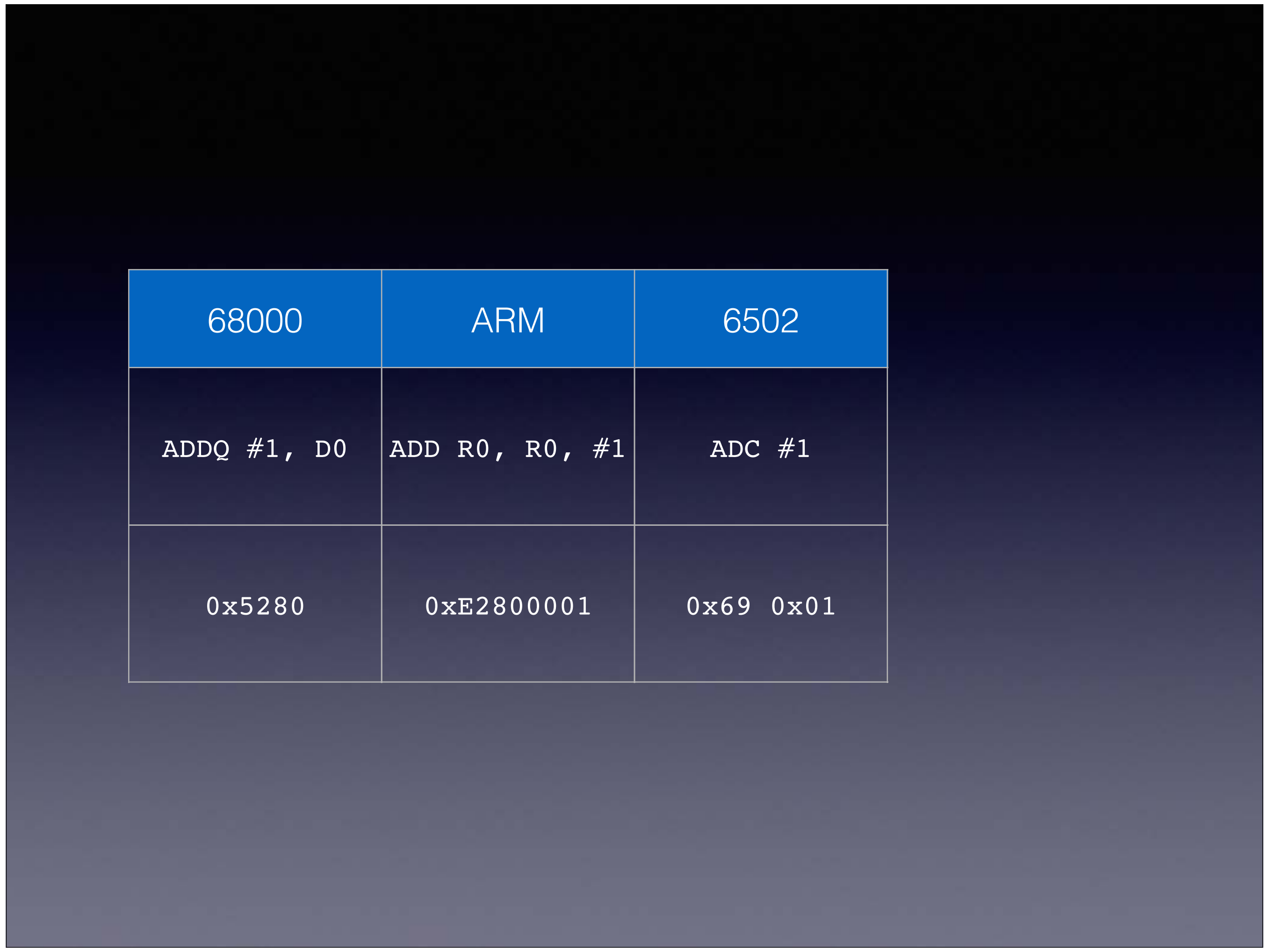
I’m not even going to attempt to work out what the value would be on the x86 — probably around 0x40



68000	ARM
`ADDQ #1, D0`	`ADD R0, R0, #1`
`0x5280`	`0xE2800001`

4 different CPUs instruction to increment a register. Although in most cases not the only way you can do it.

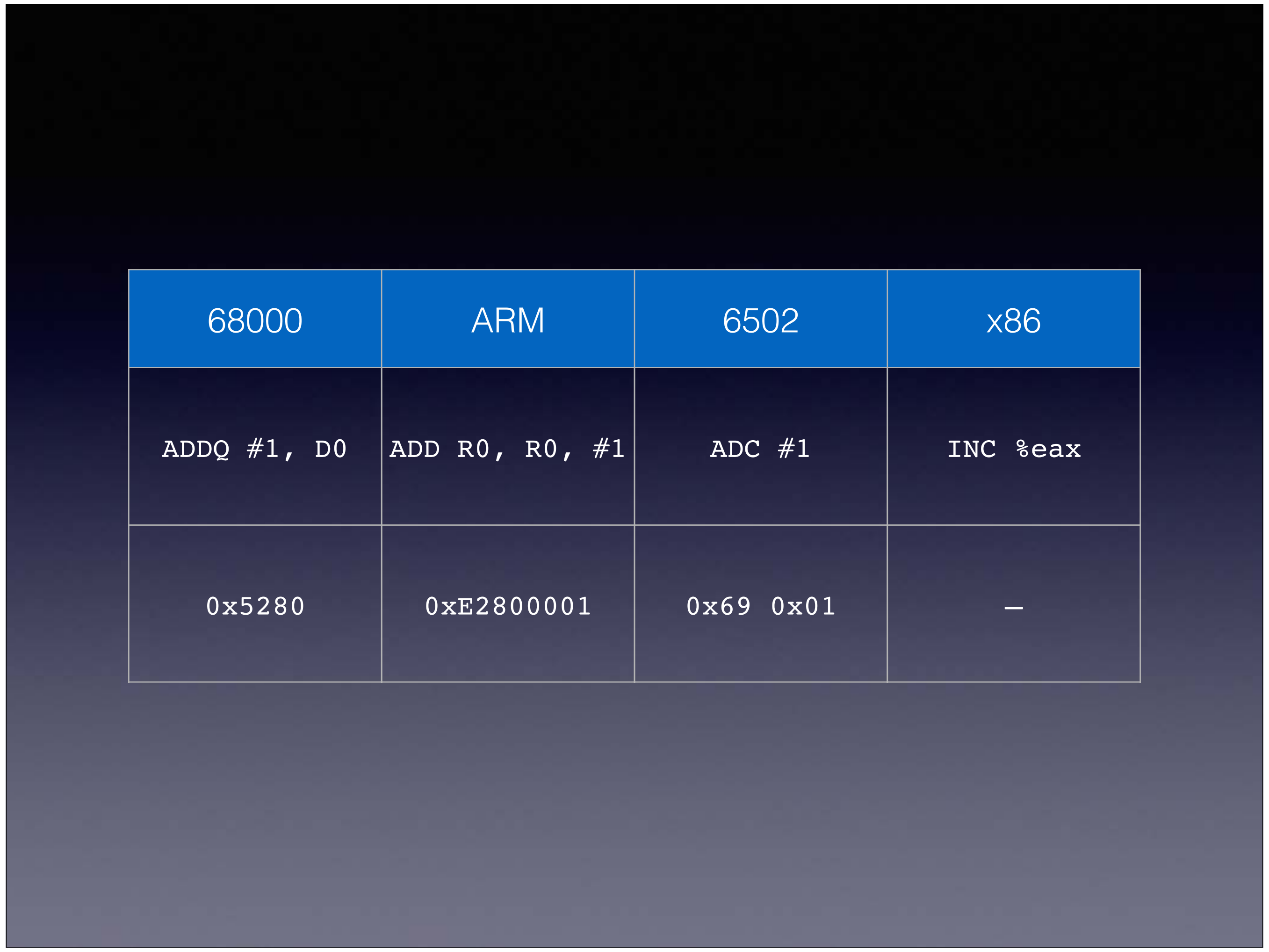
I’m not even going to attempt to work out what the value would be on the x86 — probably around 0x40



68000	ARM	6502
`ADDQ #1, D0`	`ADD R0, R0, #1`	`ADC #1`
`0x5280`	`0xE2800001`	`0x69 0x01`

4 different CPUs instruction to increment a register. Although in most cases not the only way you can do it.

I'm not even going to attempt to work out what the value would be on the x86 — probably around 0x40



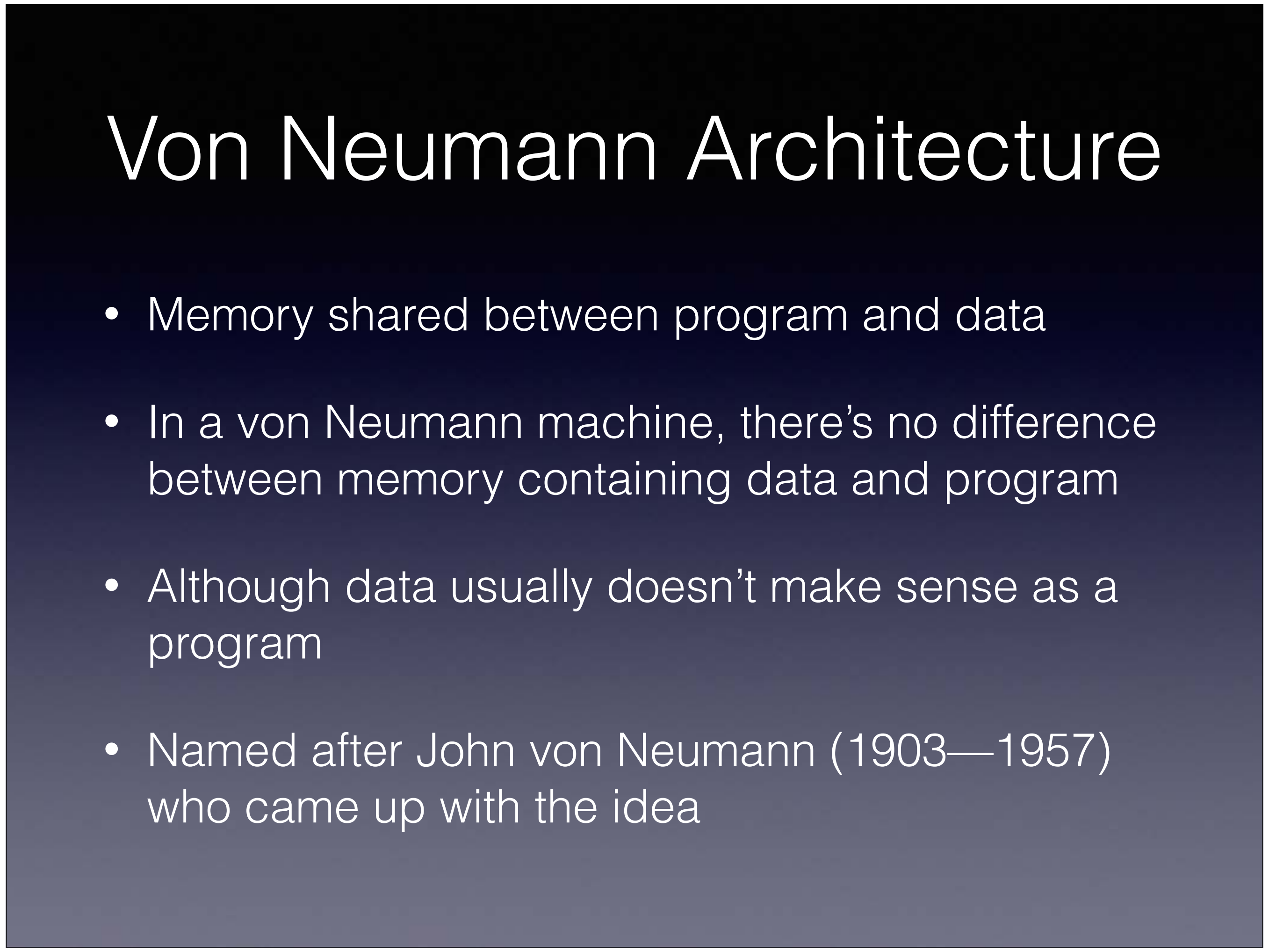
68000	ARM	6502	x86
`ADDQ #1, D0`	`ADD R0, R0, #1`	`ADC #1`	`INC %eax`
`0x5280`	`0xE2800001`	`0x69 0x01`	–

4 different CPUs instruction to increment a register. Although in most cases not the only way you can do it.

I’m not even going to attempt to work out what the value would be on the x86 — probably around 0x40

CPU Instructions

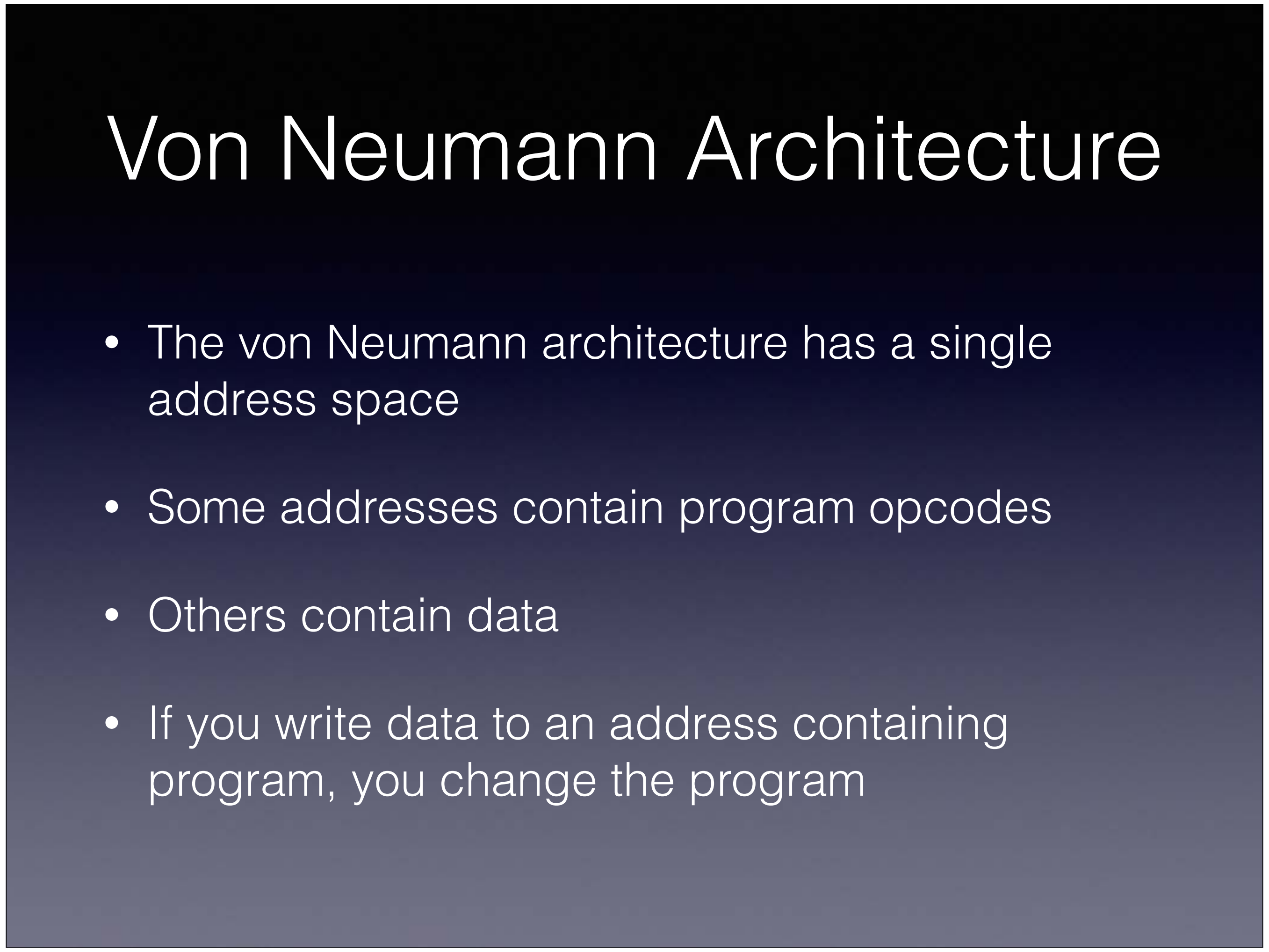
- Opcode size varies with different Instruction Sets
- ARM ISA has opcodes that are exactly 32-bits wide
- x86 has a variable instruction width (1 to 16 bytes)
- Fixed instruction width makes it much easier to build a CPU
- x86 doesn’t know how long the instruction is until it starts decoding it…



Von Neumann Architecture

- Memory shared between program and data
- In a von Neumann machine, there's no difference between memory containing data and program
- Although data usually doesn't make sense as a program
- Named after John von Neumann (1903—1957) who came up with the idea

Although it does in some cases



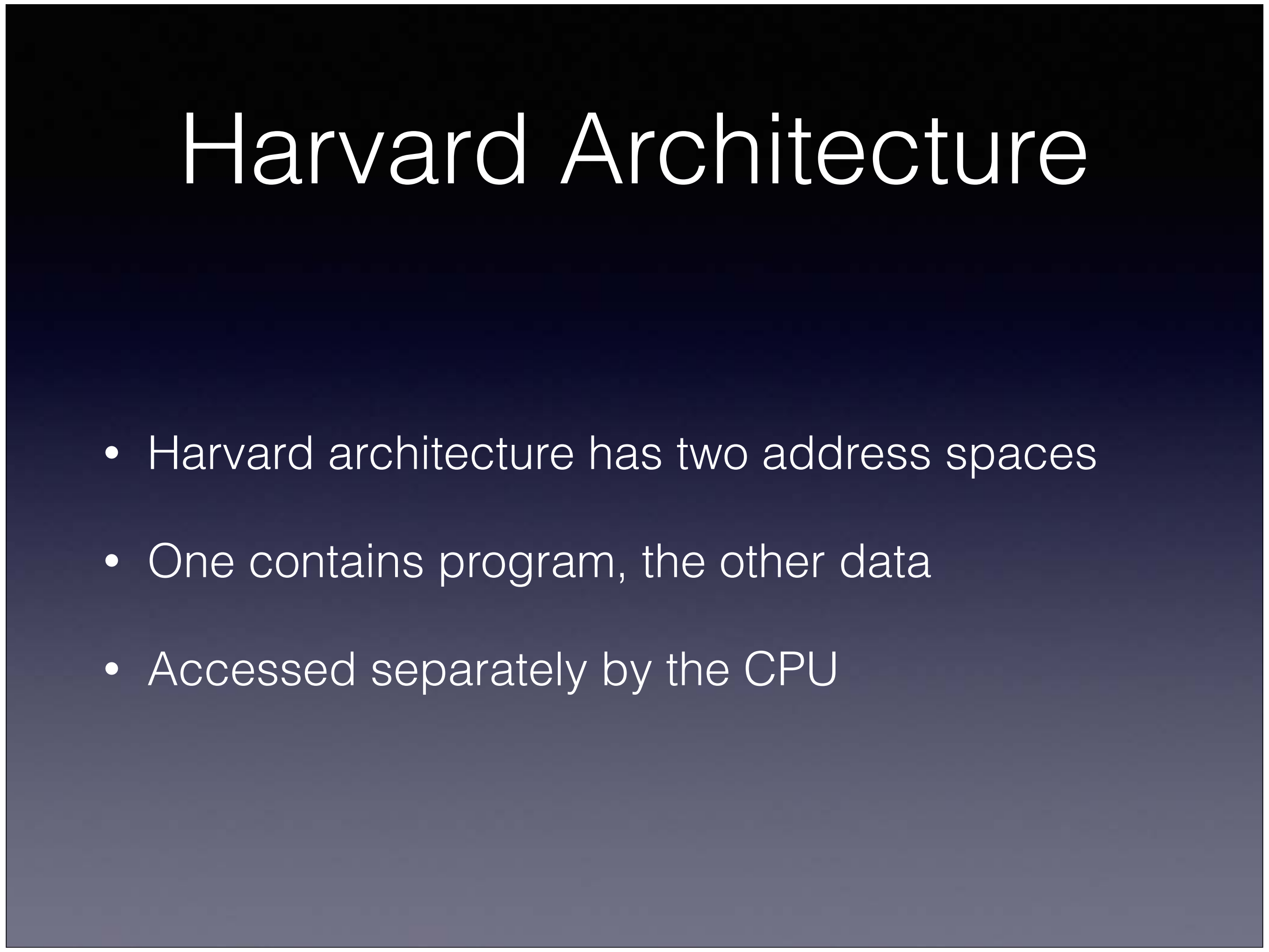
Von Neumann Architecture

- The von Neumann architecture has a single address space
- Some addresses contain program opcodes
- Others contain data
- If you write data to an address containing program, you change the program

Named after John von Neumann (1903-1957)

Harvard Architecture

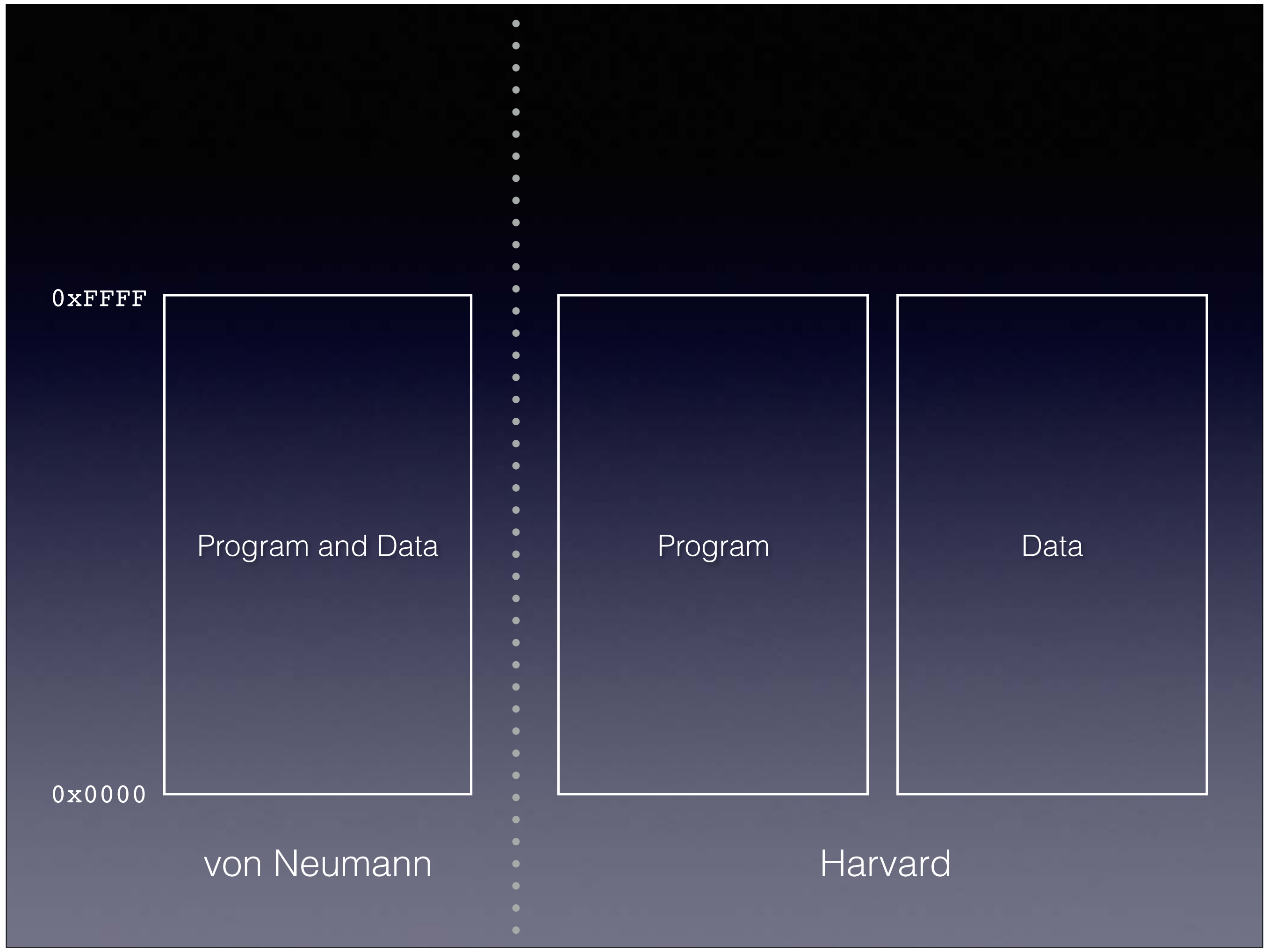
- Not the only approach
- Harvard architecture has separate memory spaces for program and data
- So storing data at an address won't affect program memory (which it would in the von Neumann model)
- Modern CPUs are a hybrid of Harvard and von Neumann architecture



Harvard Architecture

- Harvard architecture has two address spaces
- One contains program, the other data
- Accessed separately by the CPU

Named after John von Neumann (1903-1957)


0xFFFF
0x0000
Program and Data
Program
Data
von Neumann
Harvard

Modern CPUs

- Ostensibly use a von Neumann architecture
- But the implementations (to make them fast) give them Harvard-like
- Unless you are writing specialised forms of code (self-modifying/self-generating code) then they can be thought of as von Neumann

Fetch - Decode - Execute

- CPU sits in a cycle of:
  - Fetching an instruction
  - Decoding the instruction
  - Executing it
    - Which may involve accessing memory again
- Next instruction then fetched