EECE416 Microcomputer Fundamentals & Design

Computer Architecture

Dr. Charles Kim

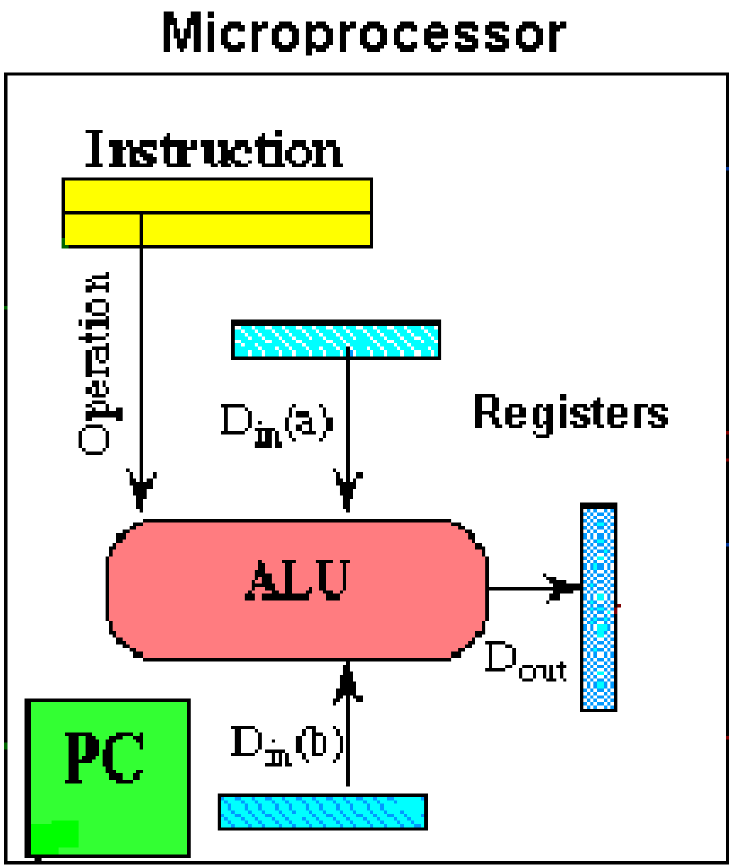
Howard University

“Computer Architecture”

- Computer Architecture
  - Art of selecting and interconnecting hardware components to create functional unit (or computer)
  - 2 points of view
    - **Instruction Set architecture (ISA):**
      - the code that a CPU reads and acts upon. It is the machine language (or assembly language), including the instruction set, word size, memory address modes, processor registers, and address and data formats
      - Interface between H/W and S/W
      - programmers’ point of view
    - **Microarchitecture** (or computer organization):
      - describes the data paths, data processing elements and data storage elements, size of cache, and describes how they should implement the ISA
      - Optimization
      - Power Management
      - system designers’ point of view.
  - Analogy:
    - House (rooms) – views of builders and residents
    - Car – views of manufacturers (or mechanics) and drivers

Micro-Architecture

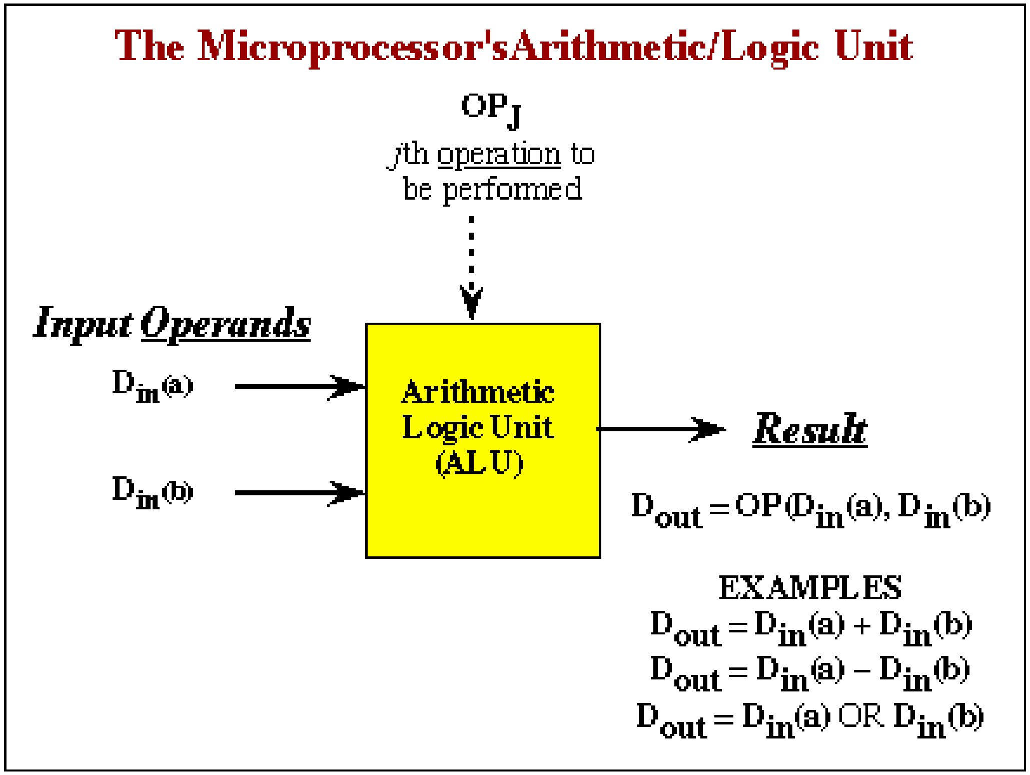
- **Computer System**
  - CPU (with PC, Register, SR) + Memory
- **Micro-Architecture:**
  - “conceptual design and fundamental operational structure of a computer system”
  - “blueprint and functional description of requirements and design implementations of a computer”
  - focusing on the way the CPU performs and accesses memory.



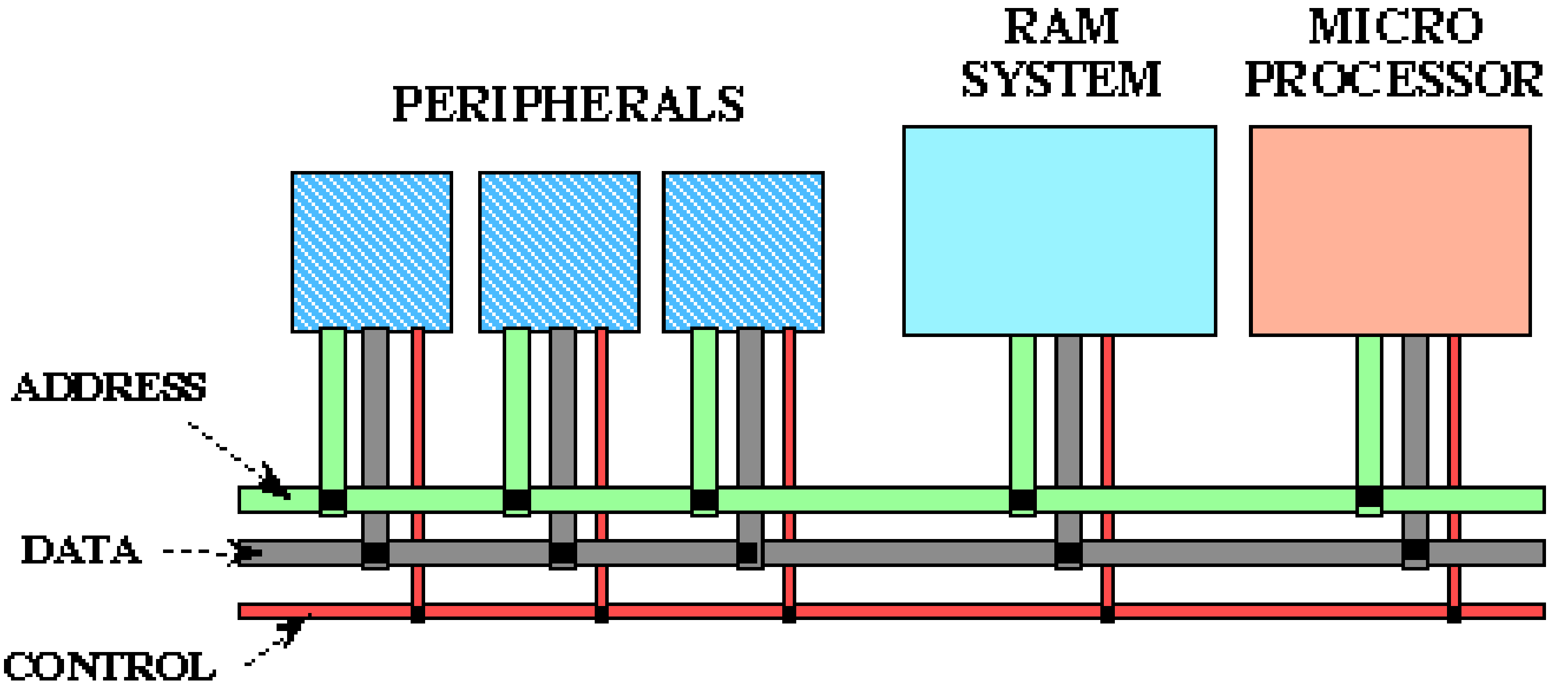
Micro-Architecture

•**ALU (Arithmetic Logic Unit)**

- **Fundamental building block of CPU**
- **Binary Full Adder**

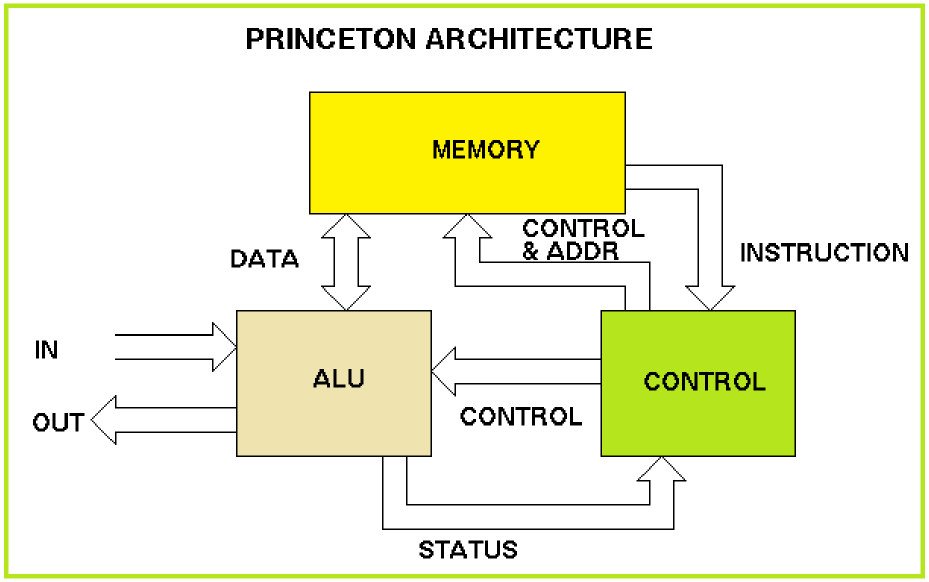

The Microprocessor'sArithmetic/Logic Unit
OP_J
jth operation to
be performed
Input Operands
$D_{in}(a)$
$D_{in}(b)$
Arithmetic
Logic Unit
(ALU)
Result
$D_{out} = OP(D_{in}(a), D_{in}(b)$
EXAMPLES
$D_{out} = D_{in}(a) + D_{in}(b)$
$D_{out} = D_{in}(a) - D_{in}(b)$
$D_{out} = D_{in}(a)$ OR $D_{in}(b)$

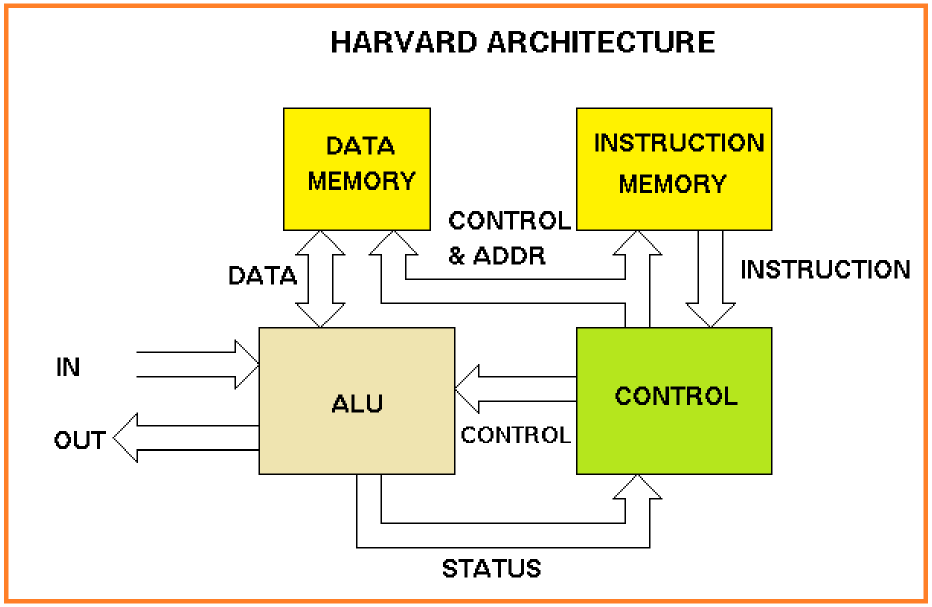
Microprocessor Bus



Architecture by CPU+MEM organization

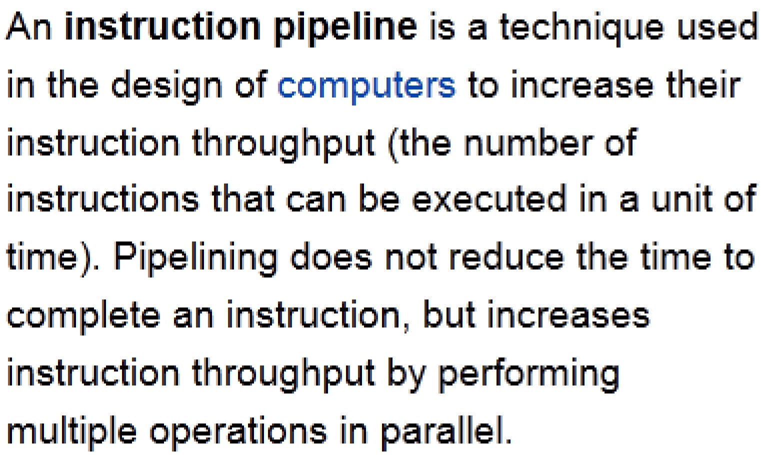
- Princeton (or von Neumann) Architecture
  - MEM contains both Instruction and Data
  - Von Neumann Bottleneck – CPU <→ Memory
  - Cache
- Harvard Architecture
  - Data MEM and Instruction MEM
  - Higher Performance – via Pipeline
  - Better for DSP
  - Higher MEM Bandwidth





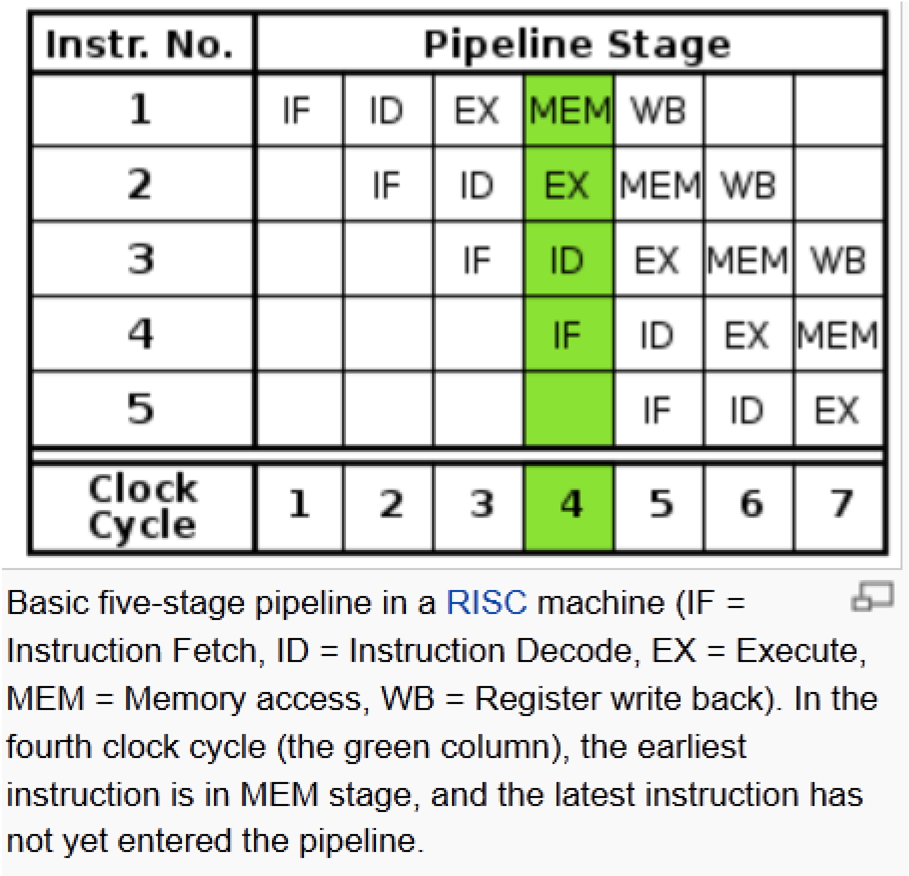
“Pipeline”?

⌘Instruction Pipeline



An **instruction pipeline** is a technique used in the design of computers to increase their instruction throughput (the number of instructions that can be executed in a unit of time). Pipelining does not reduce the time to complete an instruction, but increases instruction throughput by performing multiple operations in parallel.

The term pipeline is an analogy to the fact that there is fluid in each link of a pipeline, as each part of the processor is occupied with work.



Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

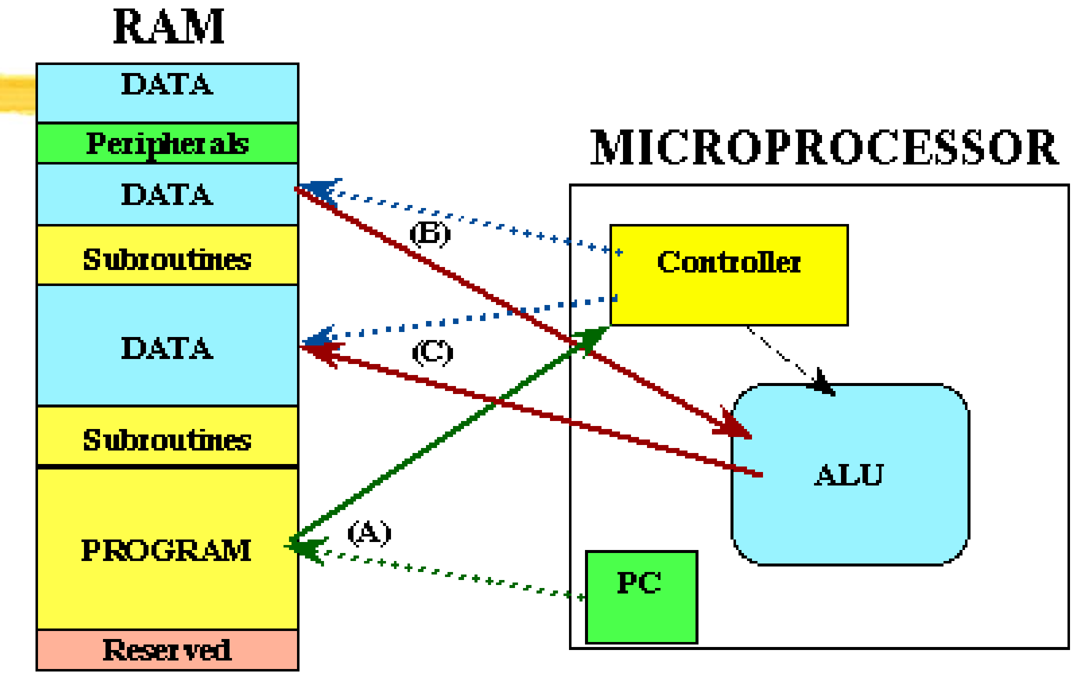
Basic five-stage pipeline in a RISC machine (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back). In the fourth clock cycle (the green column), the earliest instruction is in MEM stage, and the latest instruction has not yet entered the pipeline.

Princeton Architecture

1.**Step (A):** The address for the instruction to be next executed is read into

2. **Step (B):** The controller "decodes" the instruction

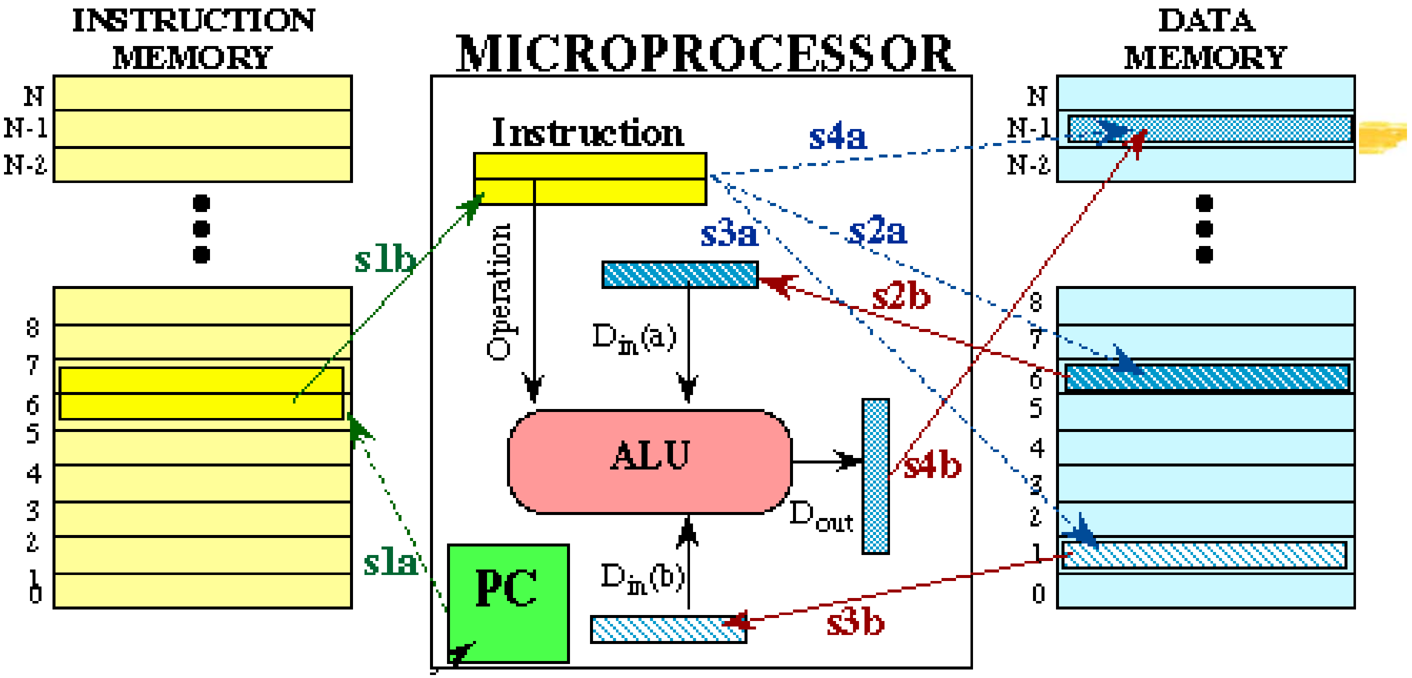
3.**Step (C)**: Following completion of the instruction, the controller provides the address, to the memory unit, at which the data result generated by the operation will be stored.



- CPU can be either reading an instruction or reading/writing data from/to the memory.
- Both cannot occur at the same time since the instructions and data use the same bus system

Harvard Architecture

- 1. CPU can both read an instruction and perform a data memory access at the same time.
- 2. Faster for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway.



"PC" is the microprocessor's Program Counter

Address		Action
s1a	---------	s1b: get instruction
s2a	---------	s2b: get first data input
s3a	---------	s3b: get second data input
s4a	---------	s4b: store data output

Architecture by Instructions and Executions

- CISC (Complex Instruction Set Computer)
  - Variety of instructions for complex tasks directly to hardware
  - Easy to translate high-level language to assembly
  - Complex Hardware
  - Instructions of varying length
- RISC (Reduced Instruction Set Computer)
  - Fewer and simpler instructions
  - Each instruction takes the same amount of time
  - Less complex hardware
  - High performance microprocessors
  - Pipelined instruction execution (several instructions are executed in parallel)

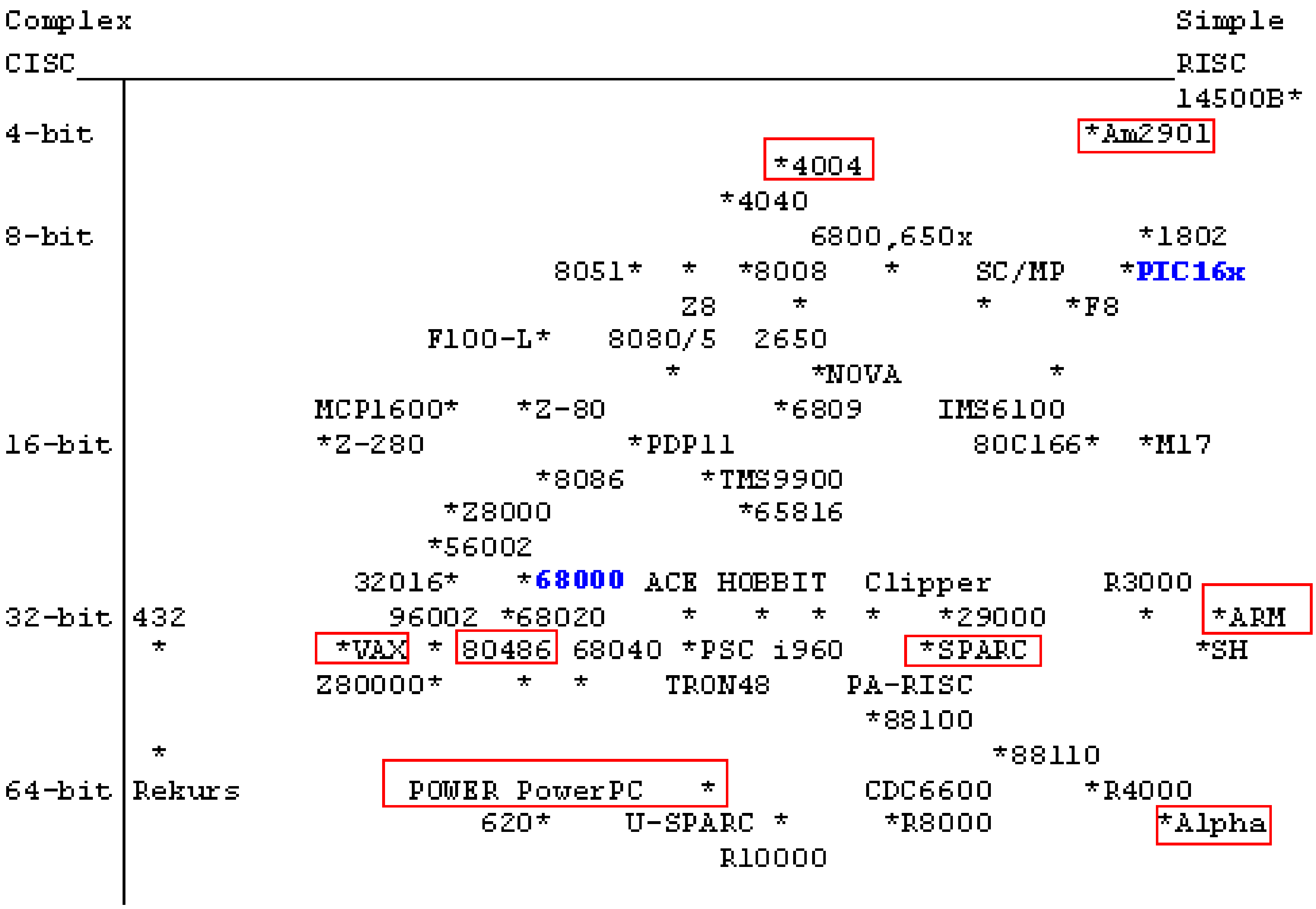
CISC

- Architecture of prior to mid-1980’s
  - IBM390, Motorola 680x0, Intel80x86
- Basic Fetch-Execute sequence to support a large number of complex instructions
- Complex decoding procedures
- Complex control unit
- One instruction achieves a complex task

RISC

- Favorable changes for RISC
  - **Caches** to speed instruction fetches
  - Dramatic memory size increases/cost decreases
  - Better ***pipelining***
  - Advanced optimizing compilers
- Characteristics of RISC
  - Instructions are of a uniform length
  - Increased number of registers to hold frequently used variables (16 - 64 Registers)
  - Central to High Performance Computing

Processor Classification



Complex Simple

CISC RISC

14500B*

4-bit *Am2901

*4004

*4040

8-bit 6800,650x *1802

8051* * *8008 * SC/MP ***PIC16x**

Z8 * * *F8

F100-L* 8080/5 2650

* *NOVA *

MCP1600* *Z-80 *6809 IMS6100

16-bit *Z-280 *PDP11 80C166* *M17

*8086 *TMS9900

*Z8000 *65816

*56002

32016* ***68000** ACE HOBBIT Clipper R3000

32-bit 432 96002 *68020 * * * * *29000 * *ARM

* *VAX * 80486 68040 *PSC i960 *SPARC *SH

Z80000* * * TRON48 PA-RISC

*88100

* *88110

64-bit Rekurs POWER PowerPC * CDC6600 *R4000

620* U-SPARC * *R8000 *Alpha

R10000

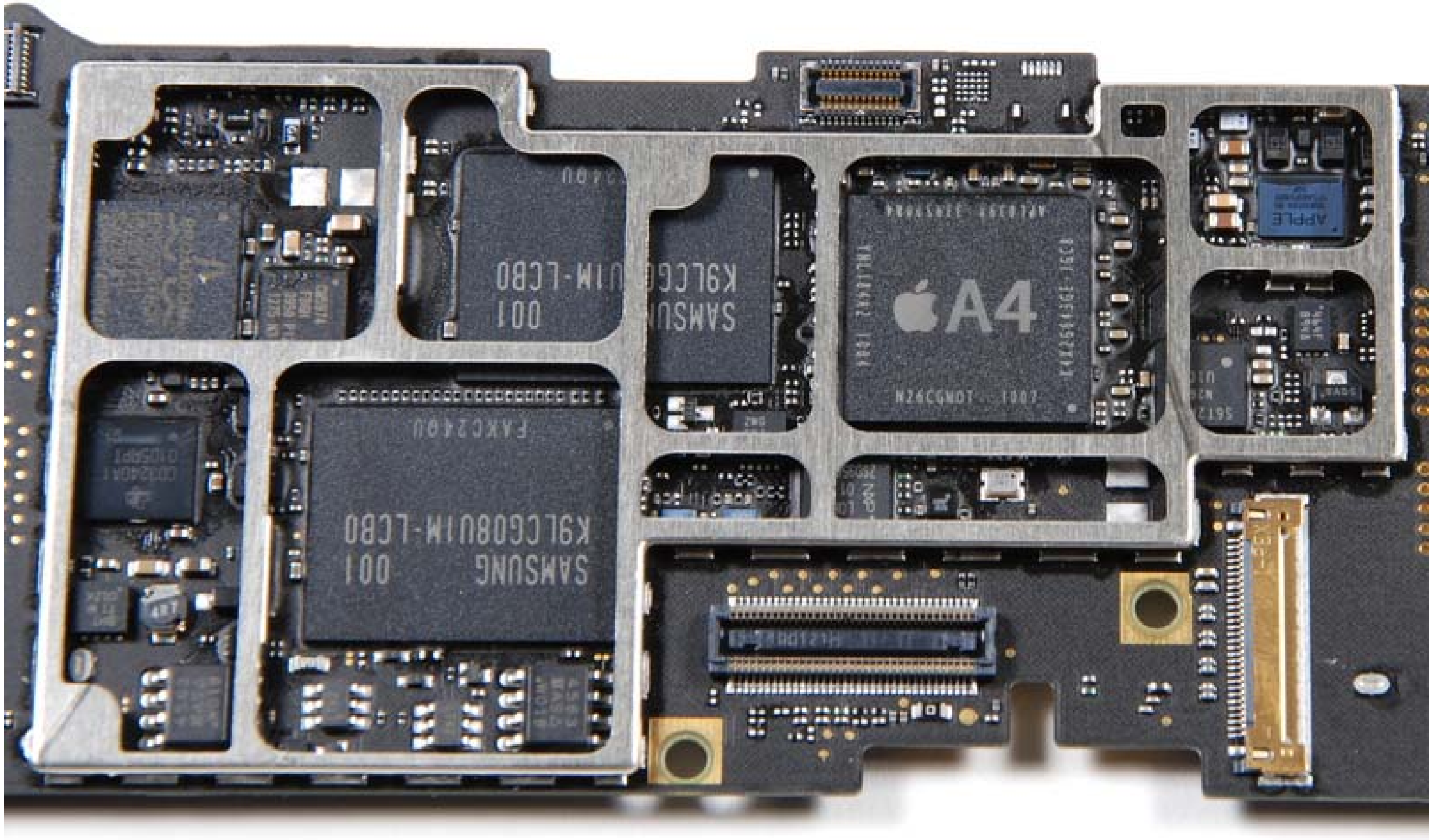
Intel inside?

- Next PCs or Mobile Computing Devices
  - Smart phones
    - Apple Processors
    - ARMs
    - Qualcomm
  - Mobile Devices – Smartphones, MP3, Digicam (on ARM)
  - Run on Intel's x86? --- Intel's wish

?

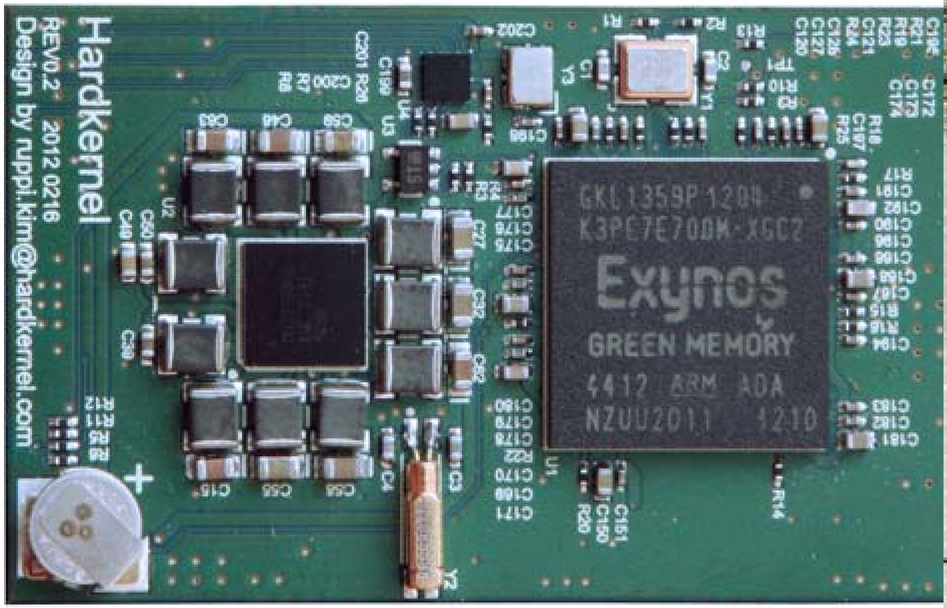
What’s inside?

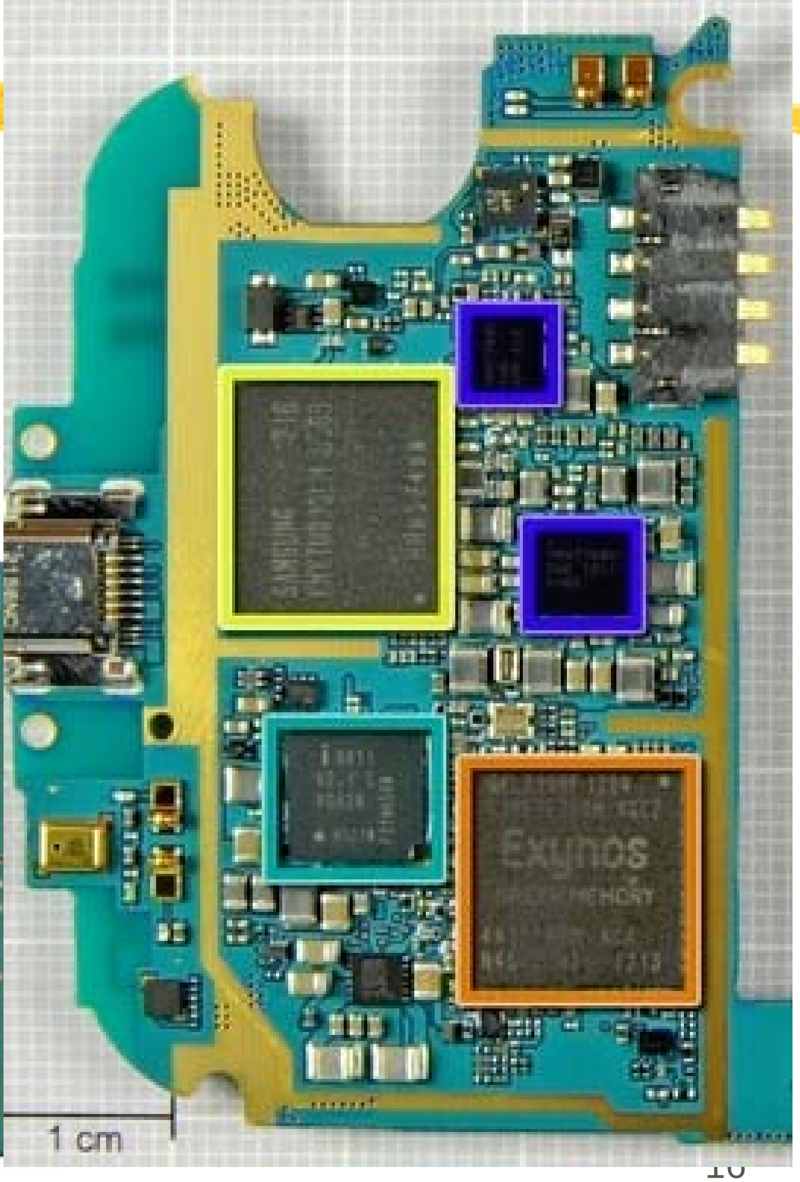
- iPhone: 1GHz-A4 microprocessor, 256MB Samsung RAM,



What’s inside?

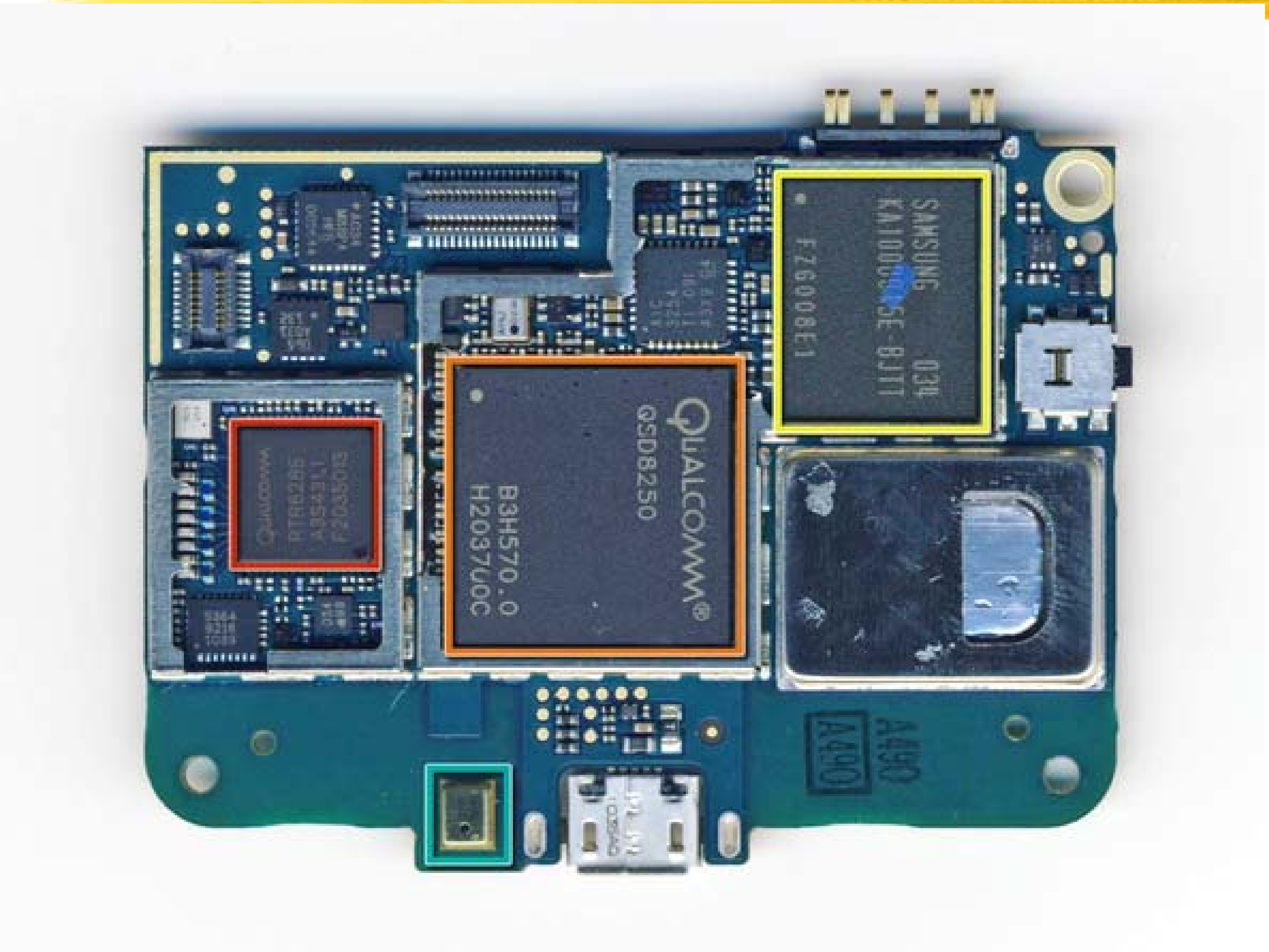
- Samsung Galaxy
  - Samsung Exynos quad-core A9 processor
  - 1GB Memory
  - Intel Wireless processor
  - Broadcom Global Navigation Satellite System receiver





What’s Inside?

- HTC

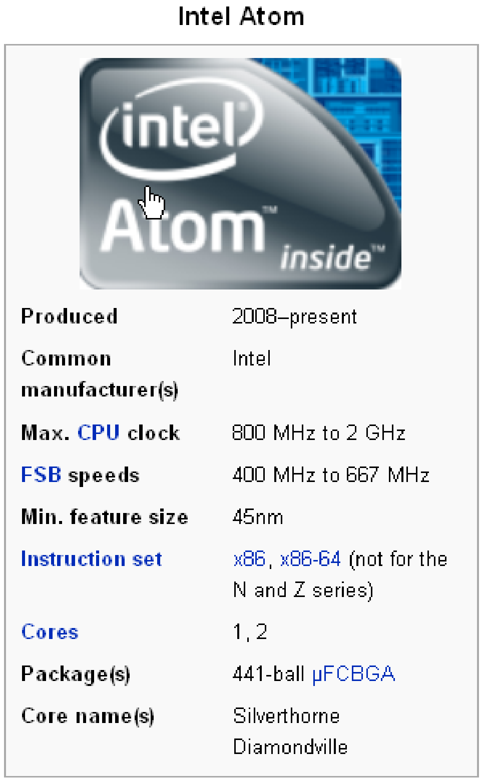


What’s inside?

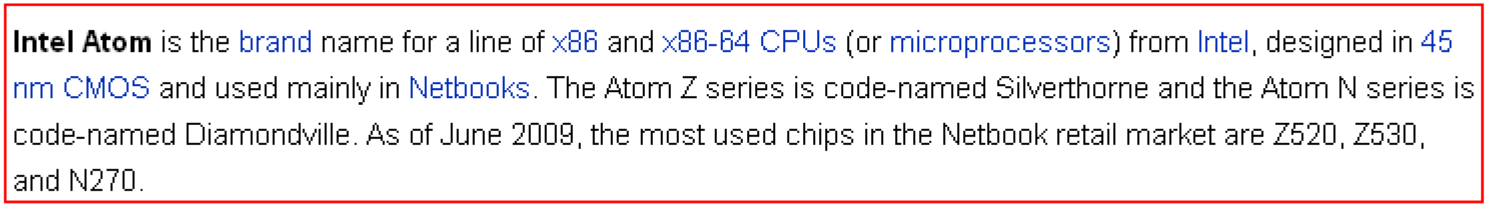
- Nokia Lumina
  - 1.4GHz Qualcomm CPU, 512MB RAM, 16GB Storage,

INTEL VS. ARM (“Advanced RISC Machine”)

- ARMs
  - No chip hardware – license only (powerful and variety of licensees) → cell phones etc
  - SoC device (CPU + I/O + Peripherals+ Memory + etc)
- INTEL
  - Does not want to License x86 (Lesson from AMD)
  - New approach for SoC: Atom based X86 SoC
- Recent Stride with “Intel Atom Inside”
  - Main processor for Laptops and Netbooks and Tablets
  - Motorola Phones: Razr

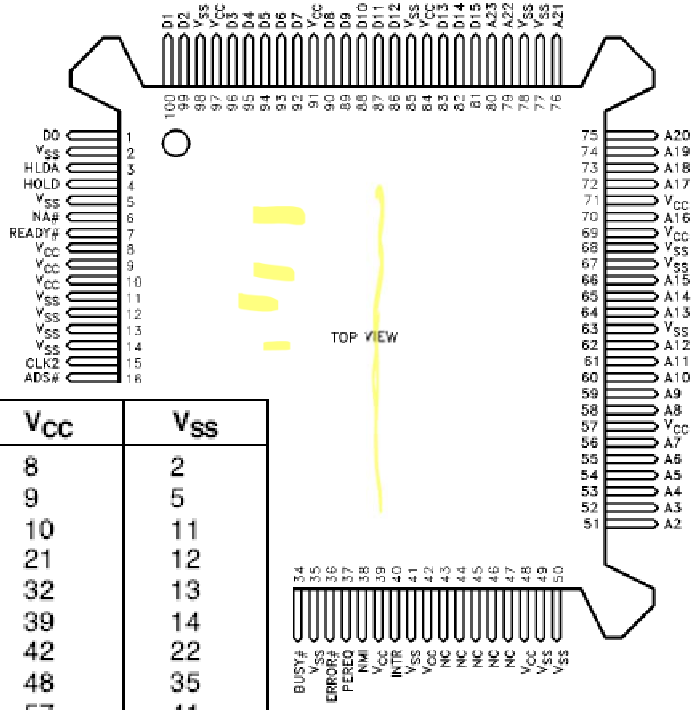

Intel Atom

Produced	2008–present
Common manufacturer(s)	Intel
Max. CPU clock	800 MHz to 2 GHz
FSB speeds	400 MHz to 667 MHz
Min. feature size	45nm
Instruction set	x86, x86-64 (not for the N and Z series)
Cores	1, 2
Package(s)	441-ball μFCBGA
Core name(s)	Silverthorne Diamondville


Intel Atom is the brand name for a line of x86 and x86-64 CPUs (or microprocessors) from Intel, designed in 45 nm CMOS and used mainly in Netbooks. The Atom Z series is code-named Silverthorne and the Atom N series is code-named Diamondville. As of June 2009, the most used chips in the Netbook retail market are Z520, Z530, and N270.

Intel 386 - Brief

- Address: A23- A1 (where is A0?)
  - BLE# and BHE# (“Byte Enable”)
- Data: D15 – D0
- Control



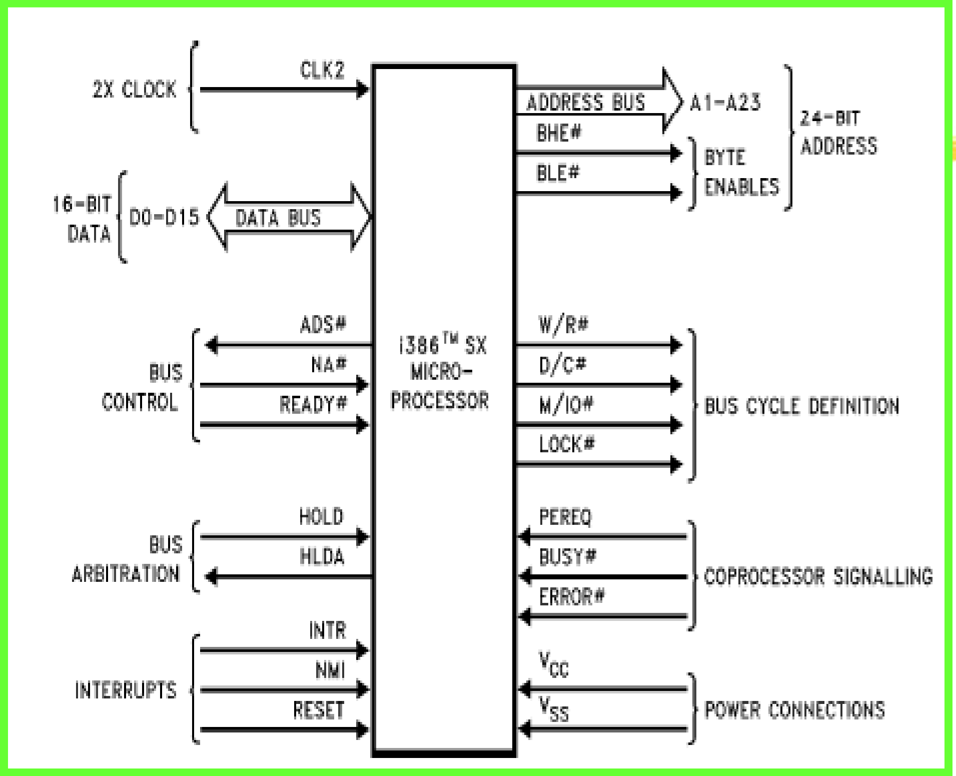
Address		Data		Control		N/C	V_{CC}	V_{SS}
A_1	18	D_0	1	ADS#	16	20	8	2
A_2	51	D_1	100	BHE#	19	27	9	5
A_3	52	D_2	99	BLE#	17	29	10	11
A_4	53	D_3	96	BUSY#	34	30	21	12
A_5	54	D_4	95	CLK2	15	31	32	13
A_6	55	D_5	94	D/C#	24	43	39	14
A_7	56	D_6	93	ERROR#	36	44	42	22
A_8	58	D_7	92	FLT#	28	45	48	35
A_9	59	D_8	90	HLDA	3	46	57	41
A_{10}	60	D_9	89	HOLD	4	47	69	49
A_{11}	61	D_{10}	88	INTR	40		71	50
A_{12}	62	D_{11}	87	LOCK#	26		84	63
A_{13}	64	D_{12}	86	M/IO#	23		91	67
A_{14}	65	D_{13}	83	NA#	6		97	68
A_{15}	66	D_{14}	82	NMI	38			77
A_{16}	70	D_{15}	81	PEREQ	37			78
A_{17}	72			READY#	7			85
A_{18}	73			RESET	33			98
A_{19}	74			W/R#	25			
A_{20}	75							
A_{21}	76							
A_{22}	79							
A_{23}	80							

Review on Number Systems

	Binary	Hexadecimal	Decimal
1.	100	________	________
2.	10101101	________	________
3.	1101110101	________	________
4.	11111011110	________	________
5.	10000000001	________	________
6.	________	8EF	________
7.	________	10	________
8.	________	A52E	________
9.	________	70C	________
10.	________	6BD3	________
11.	________	________	100
12.	________	________	527
13.	________	________	4128
14.	________	________	11947
15.	________	________	59020

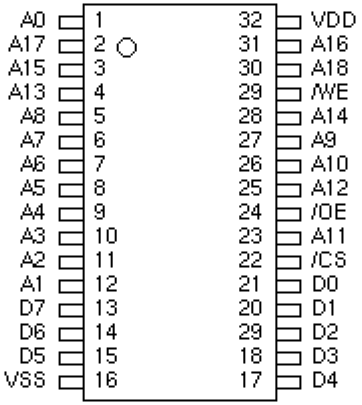
Intel 386 - Brief

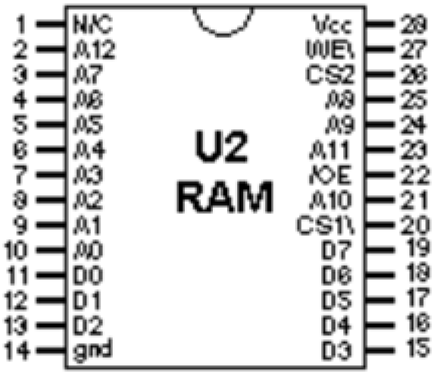
- Address: A23- A1 (where is A0?)
  - BLE# and BHE# (“Byte Enable”)
- Data: D15 – D0
- Control

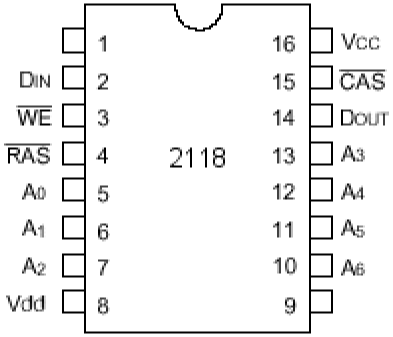


Address		Data		Control				
A_{1}	18	D_{0}	1	ADS#	16			
A_{2}	51	D_{1}	100	BHE#	19			
A_{3}	52	D_{2}	99	BLE#	17			
A_{4}	53	D_{3}	96	BUSY#	34			
A_{5}	54	D_{4}	95	CLK2	15			
A_{6}	55	D_{5}	94	D/C#	24			
A_{7}	56	D_{6}	93	ERROR#	36			
A_{8}	58	D_{7}	92	FLT#	28			
A_{9}	59	D_{8}	90	HLDA	3			
A_{10}	60	D_{9}	89	HOLD	4	47	69	49
A_{11}	61	D_{10}	88	INTR	40		71	50
A_{12}	62	D_{11}	87	LOCK#	26		84	63
A_{13}	64	D_{12}	86	M/IO#	23		91	67
A_{14}	65	D_{13}	83	NA#	6		97	68
A_{15}	66	D_{14}	82	NMI	38			77
A_{16}	70	D_{15}	81	PEREQ	37			78
A_{17}	72			READY#	7			85
A_{18}	73			RESET	33			98
A_{19}	74			W/R#	25			
A_{20}	75							
A_{21}	76							
A_{22}	79							
A_{23}	80							

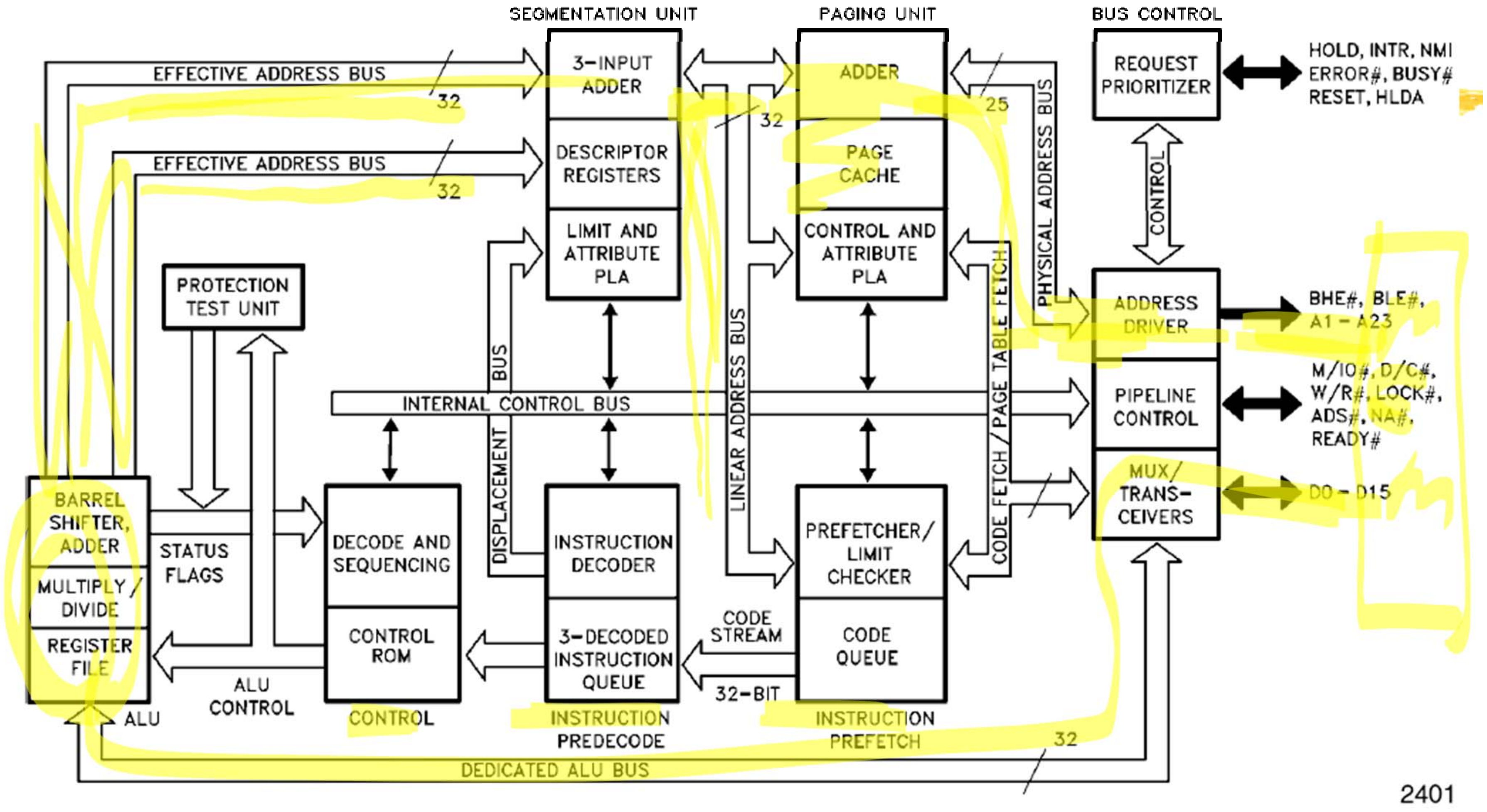
Memory Size and Address





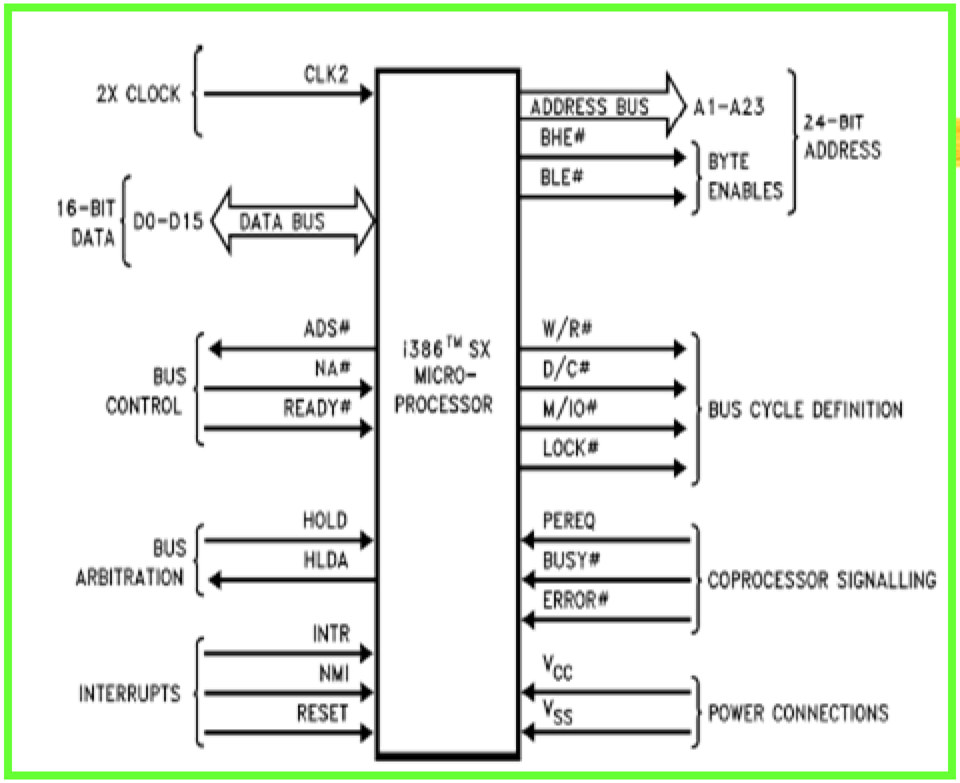


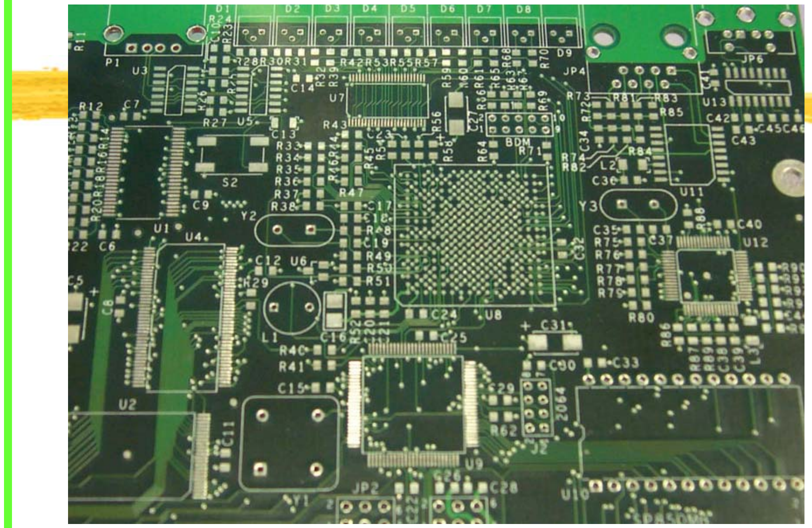
386 Micro-Architecture

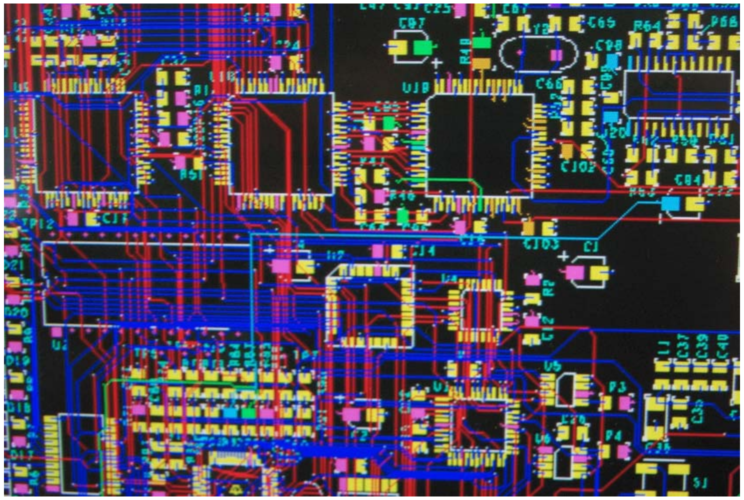


Intel386™ SX Pipelined 32-Bit Microarchitecture

Connecting with Memory, I/O, and Peripherals

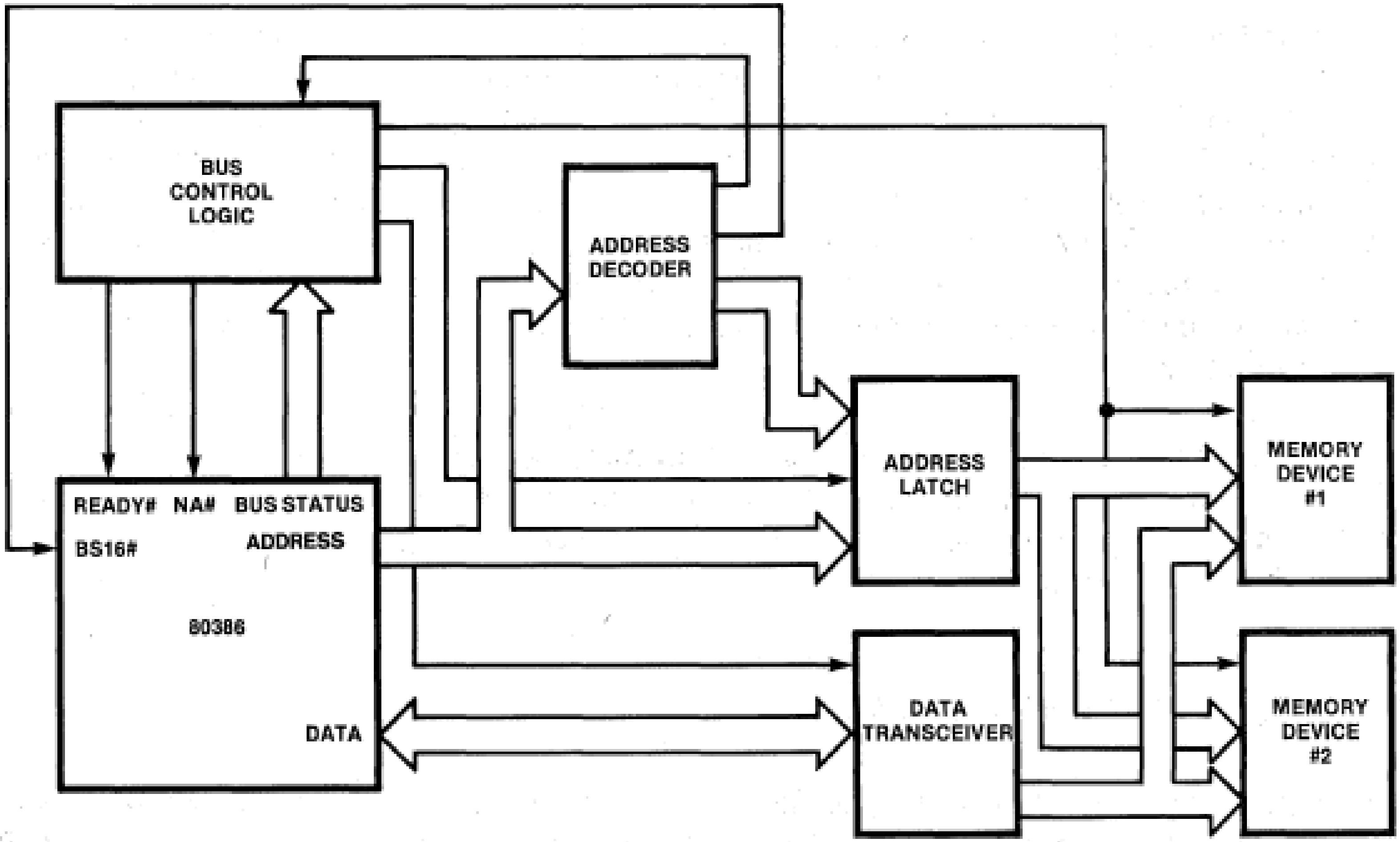




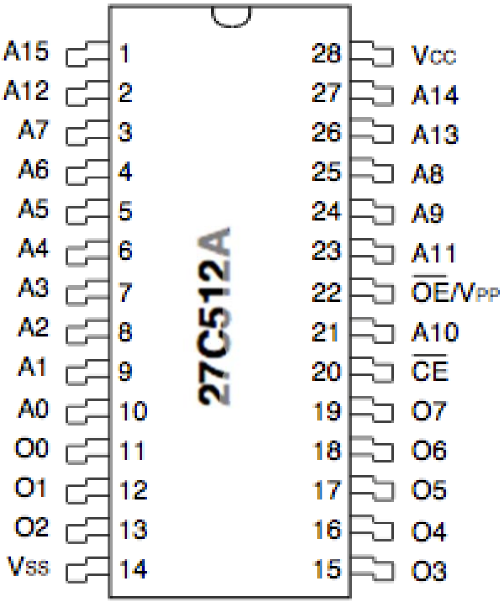


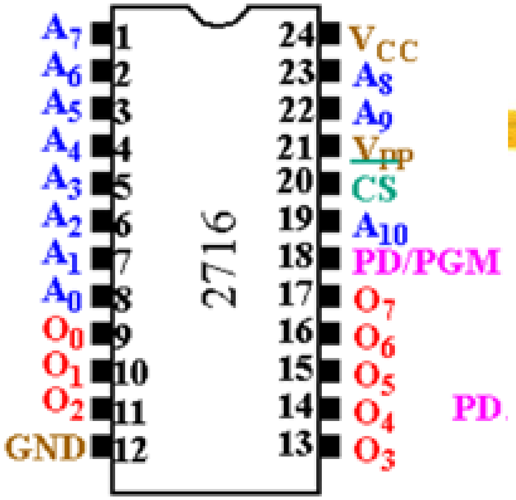
- Single Board Computers
- Processor Boards
- Kits

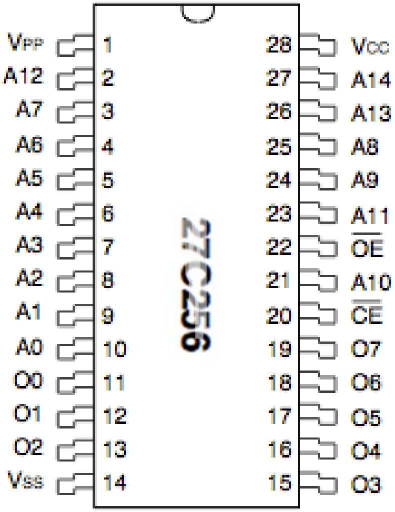
Memory Interface

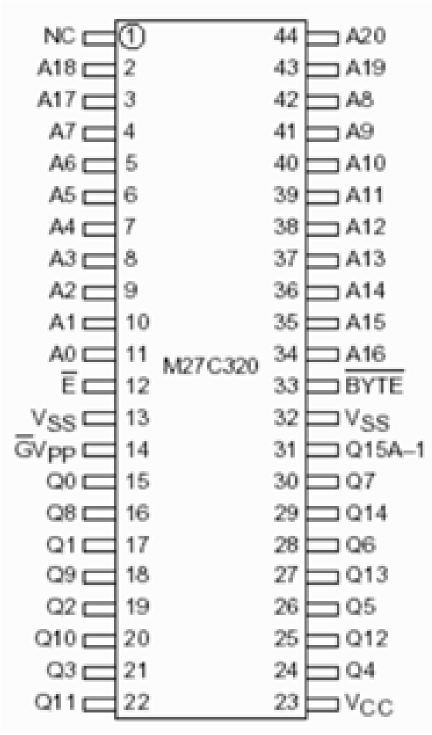


Memory Size and Address 2







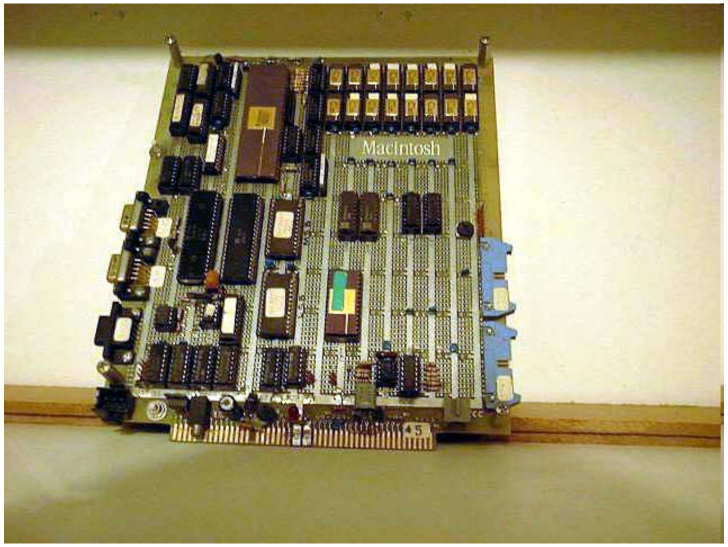


Memory Interface

- Interface between a processor and a (pair) of memory (of smaller than the maximum memory space)
- Where do we place the memory in the memory space? → "MEMORY DECODING"
- How to access two MEMs at the same time (for 16-bit Data bus)?
  - MEM --- Byte Access (8 bits)
  - UDS and LDS --- Motorola
  - BLE and BHE --- Intel

Apple Macintosh

- CPU: 8MHz Motorola 68000
- Introduced in 1984
- Memory: 128KB (512KB in later version) RAM, 64KB ROM
- 3.5” 400KB Floppy Disk
- Application: MacWrite and MacPaint
- Mouse
- 9” B&W Monitor
- Keyboard
- Serial Port (DB-9)
- Printer Port
- Addressing: 24-bit



Apple Macintosh Circuit Diagram

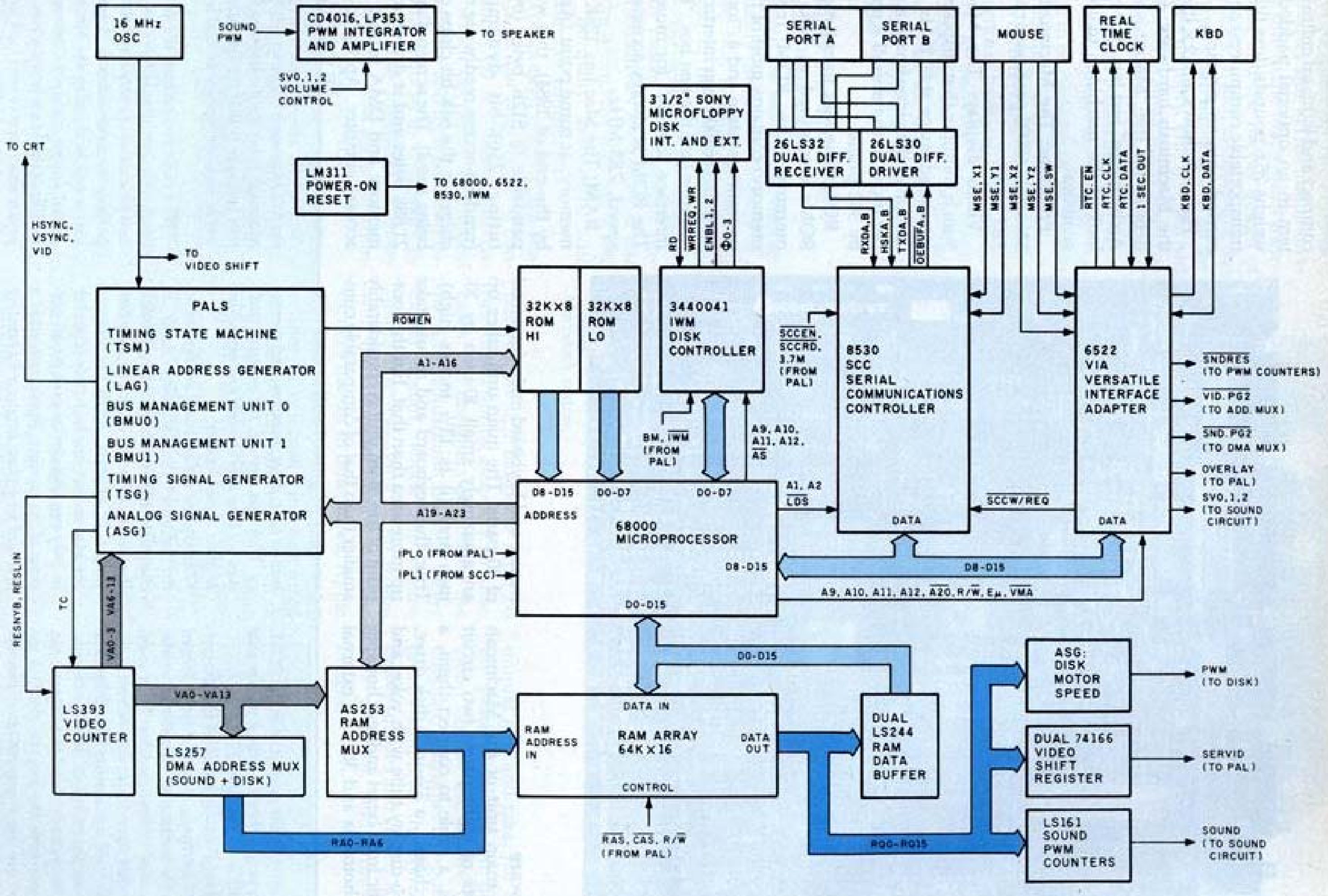


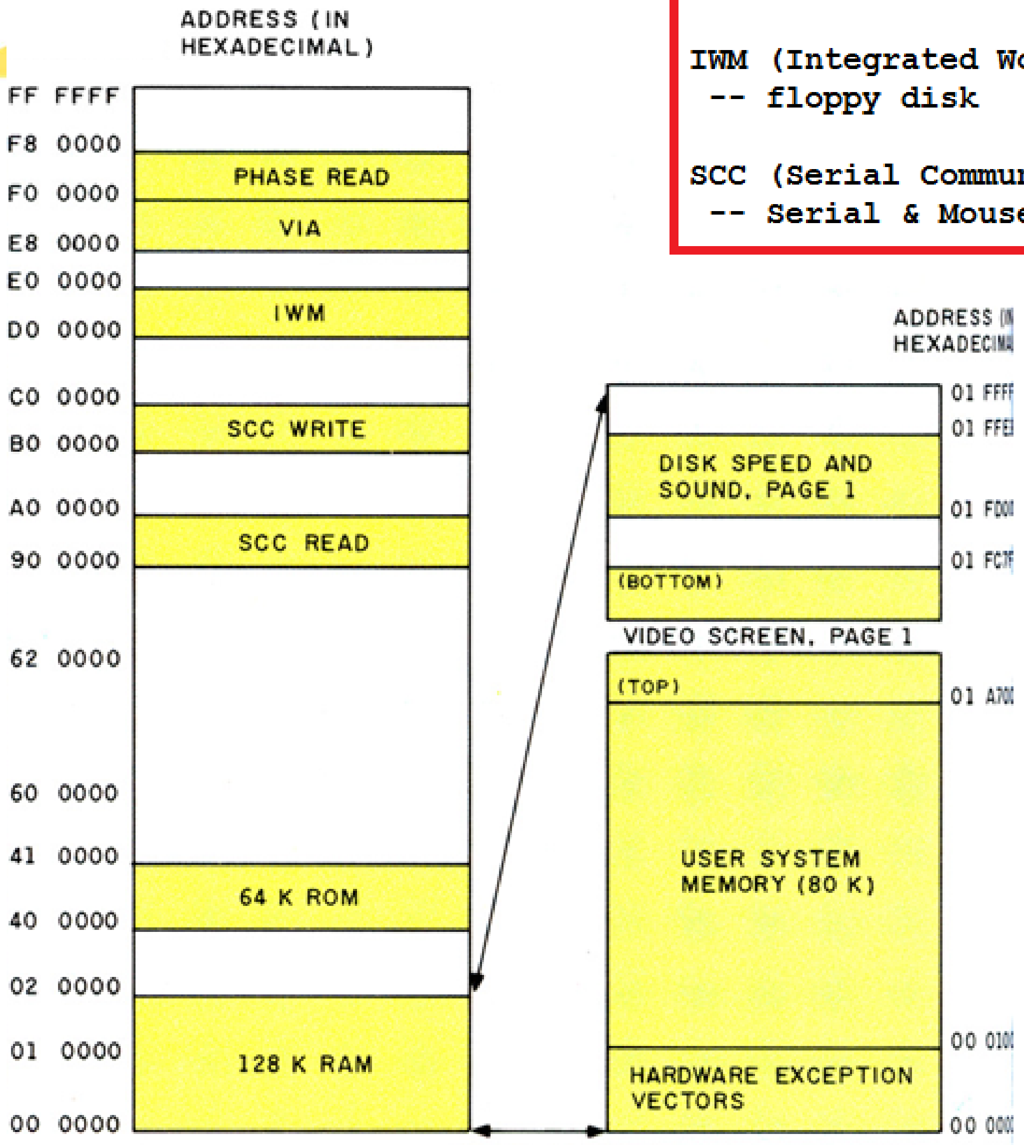
Figure 2: *A block diagram of the Macintosh hardware. For more details, see the "Macintosh System Architecture" text box.*

Memory Map (for Apple Macintosh)

VIA(Versatile Interface Adapter)
---general I/O

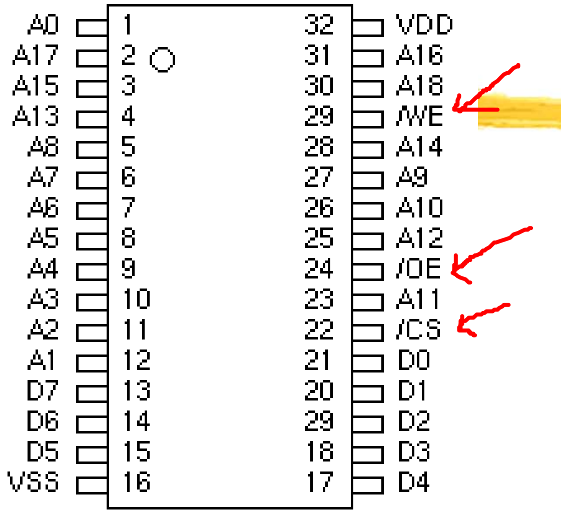
IWM (Integrated Woz Machine)
-- floppy disk

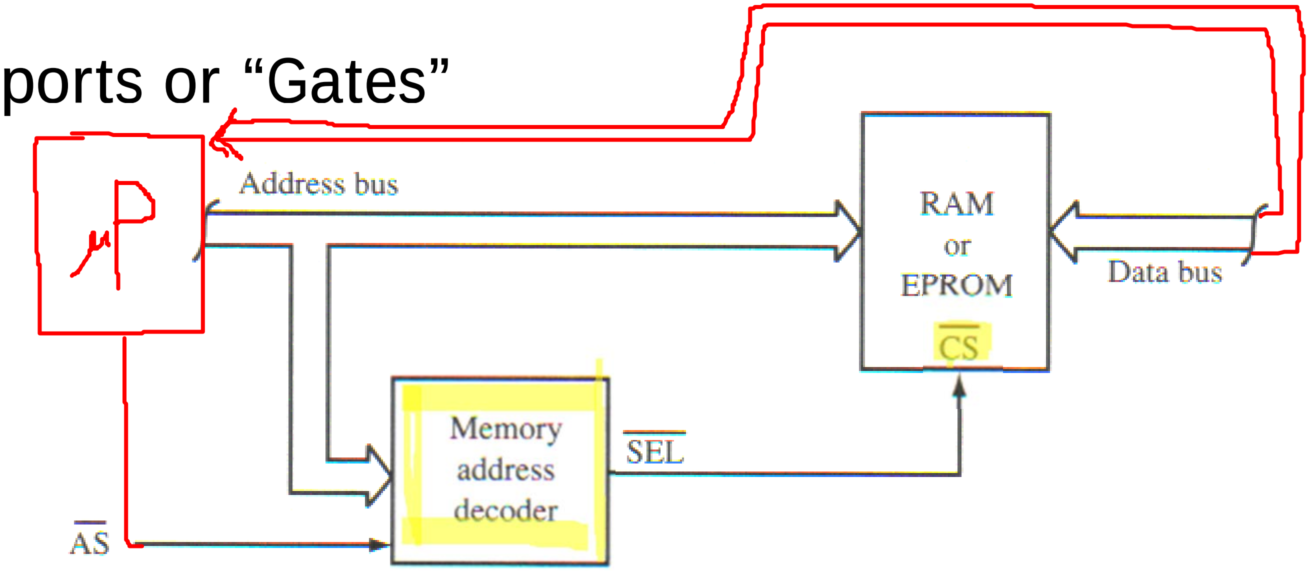
SCC (Serial Communications Controller)
-- Serial & Mouse



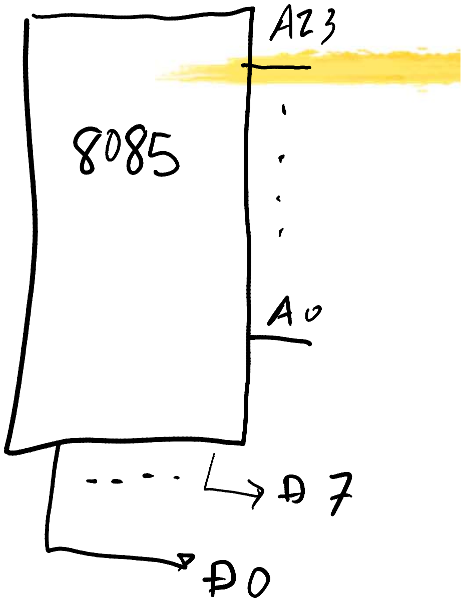
Memory Address Decoding

- How Much Memory?
  - How Many Address Lines?
    - 1K → 10 lines
    - 32K → 15 lines
    - 23 ADDR lines → 8MB?
- MEMORY PINOUTS
  - Address Lines
  - Data Lines
  - “Defense” ports or “Gates”
    - /CS
    - /CE
    - /OE
    - /WE

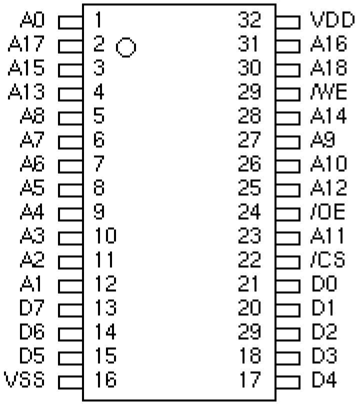




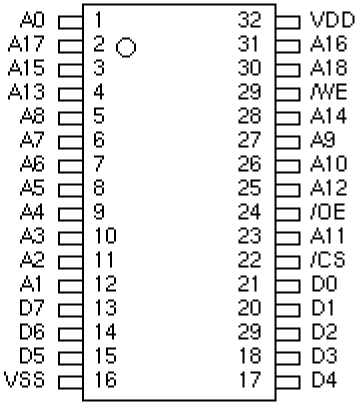
uP + MEM



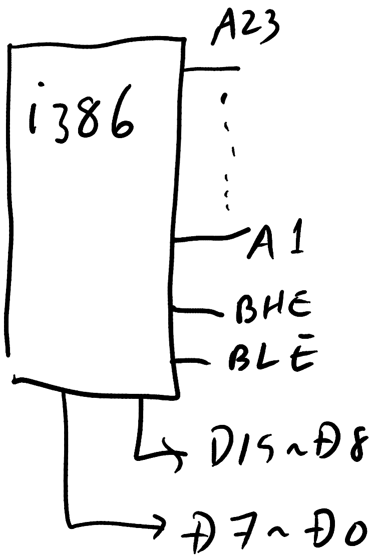
8085
A23
A0
D7
D0



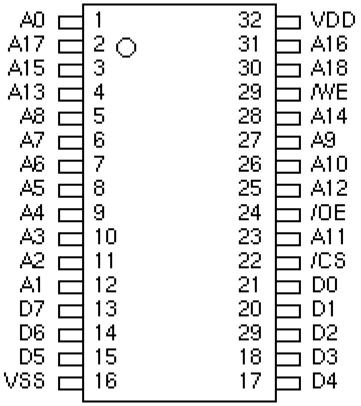
A0 1
A17 2
A15 3
A13 4
A8 5
A7 6
A6 7
A5 8
A4 9
A3 10
A2 11
A1 12
D7 13
D6 14
D5 15
VSS 16
32 VDD
31 A16
30 A18
29 /WE
28 A14
27 A9
26 A10
25 A12
24 /OE
23 A11
22 /CS
21 D0
20 D1
29 D2
18 D3
17 D4



A0 1
A17 2
A15 3
A13 4
A8 5
A7 6
A6 7
A5 8
A4 9
A3 10
A2 11
A1 12
D7 13
D6 14
D5 15
VSS 16
32 VDD
31 A16
30 A18
29 /WE
28 A14
27 A9
26 A10
25 A12
24 /OE
23 A11
22 /CS
21 D0
20 D1
29 D2
18 D3
17 D4



i386
A23
A1
BHE
BLE
D15~D8
D7~D0



A0 1
A17 2
A15 3
A13 4
A8 5
A7 6
A6 7
A5 8
A4 9
A3 10
A2 11
A1 12
D7 13
D6 14
D5 15
VSS 16
32 VDD
31 A16
30 A18
29 /WE
28 A14
27 A9
26 A10
25 A12
24 /OE
23 A11
22 /CS
21 D0
20 D1
29 D2
18 D3
17 D4

8-bit uP + MEM

- uP has 2^{24}=24MB=16MB memory space: 000000 - FFFFFF
- MEM has 2^{19}=0.5MB
- Let's place the MEM between \$00000 - \$7FFFF
- Up Addr ←→MEM Addr: A18 – A0
- The left-over Addr lines in the uP: A23 – A19
  - This condition is used to open the MEM gate (namely, /CS)

2 0 0 0 0 0

A 23 22 21 20 19 | 18 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0

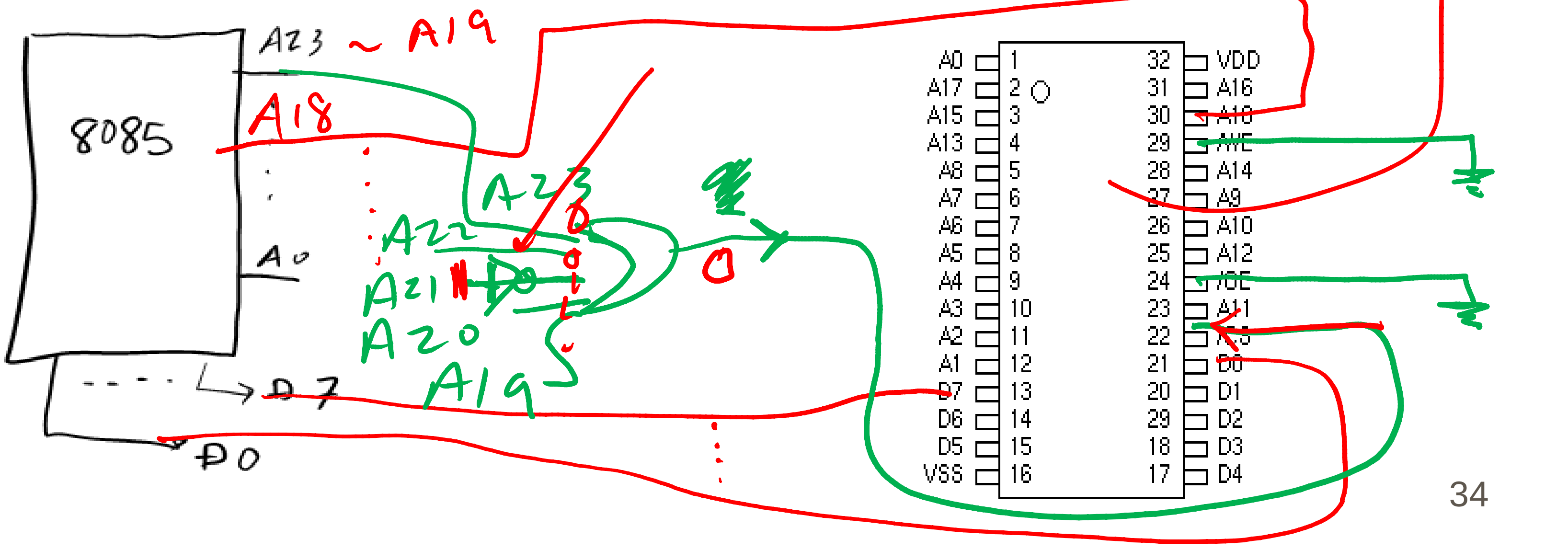
0 0 1 0 0

0 0 0 0 0 0

0 7 F F F F

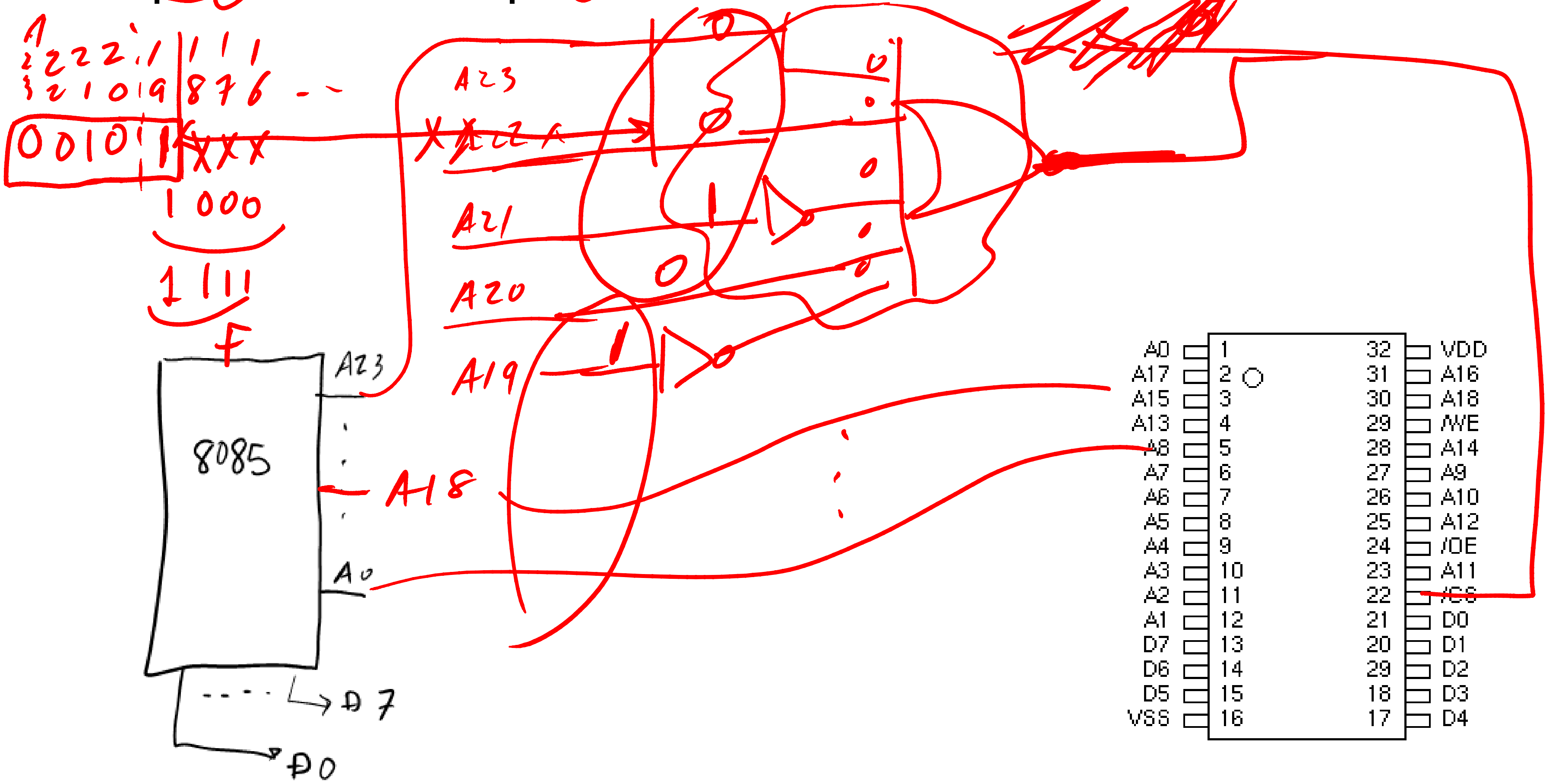
0 7FFFF

00 0000



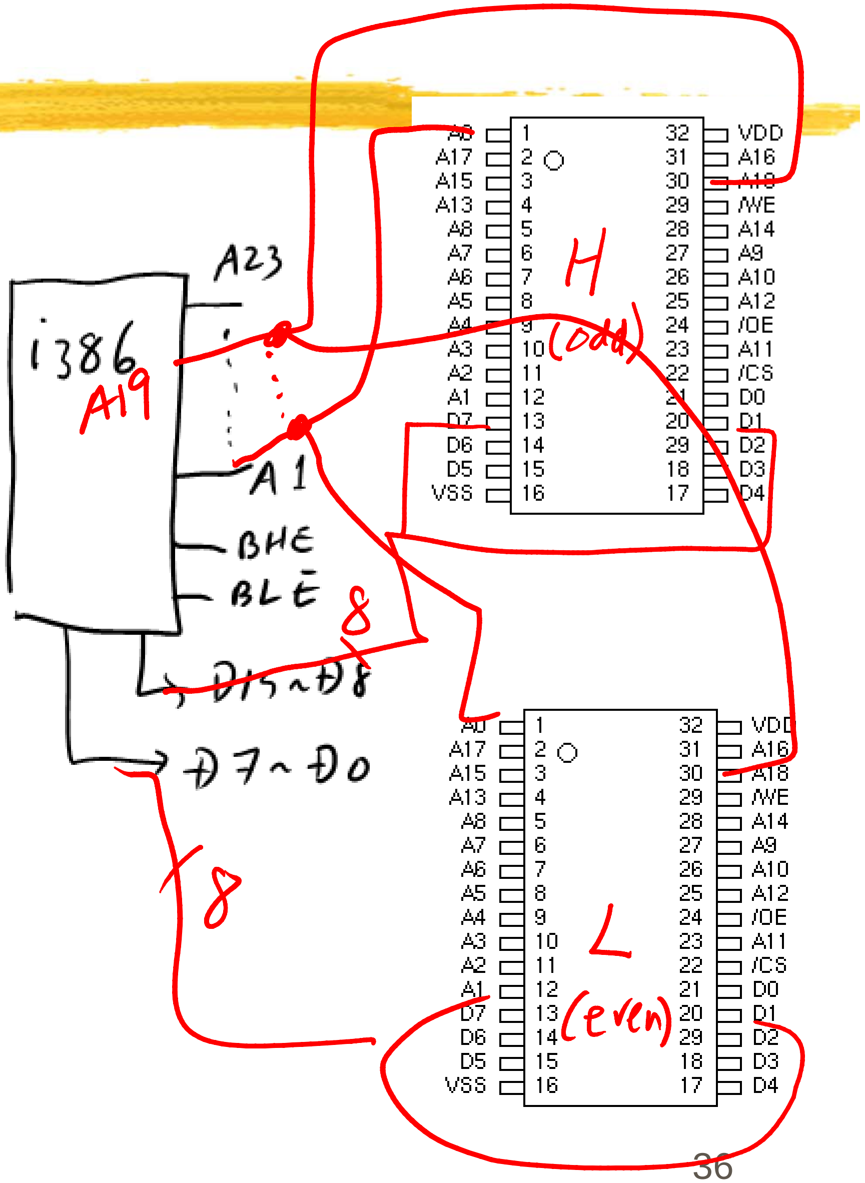
8-bit uP + MEM

- Now, place 0.5MB size MEM between $280000 - $2FFFFF



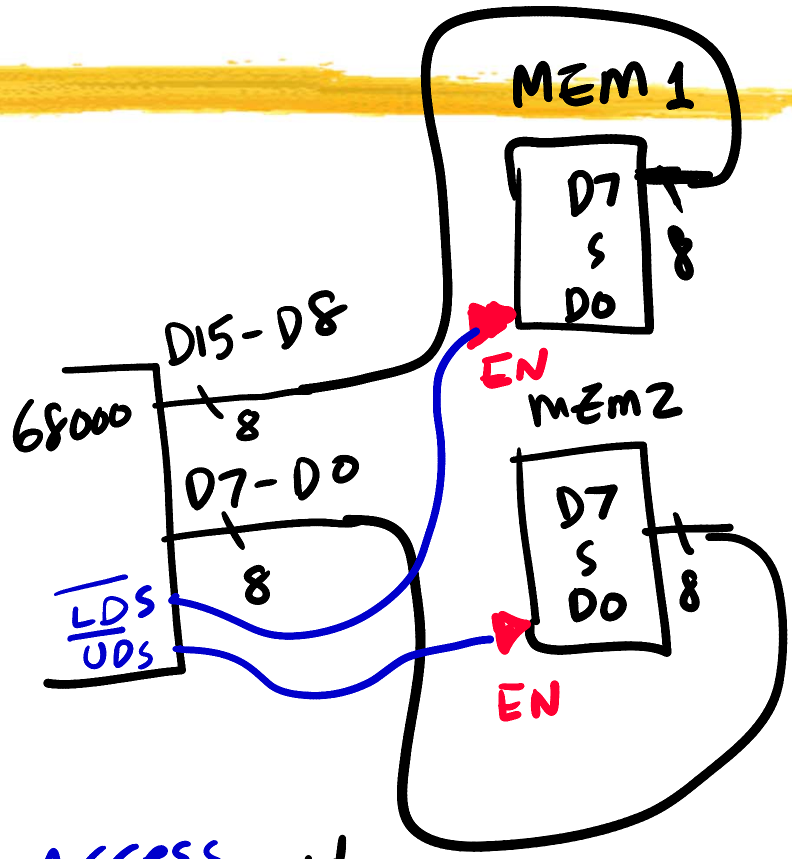
16-bit uP + MEM

- uP does not have A0
  - BHE (UDS) and BLE (LDS), instead.
- uP can access 2 MEMs
- uP Addr ←→ MEM Addr
  - A19 – A1 (uP): A18 – A0 (MEM)
  - Left-Over Addr (uP): A23-A20
  - BHE and BLE controls which MEM (or ADDRESS LOCATION) to access
    - BHE LOW: Upper MEM (upper or odd address location)
    - BLE LOW: Lower MEM (lower or even address location)
    - Both BHE amd BLE low: both address locations



BHE/UDS and BLE/LDS (for 386/68000)

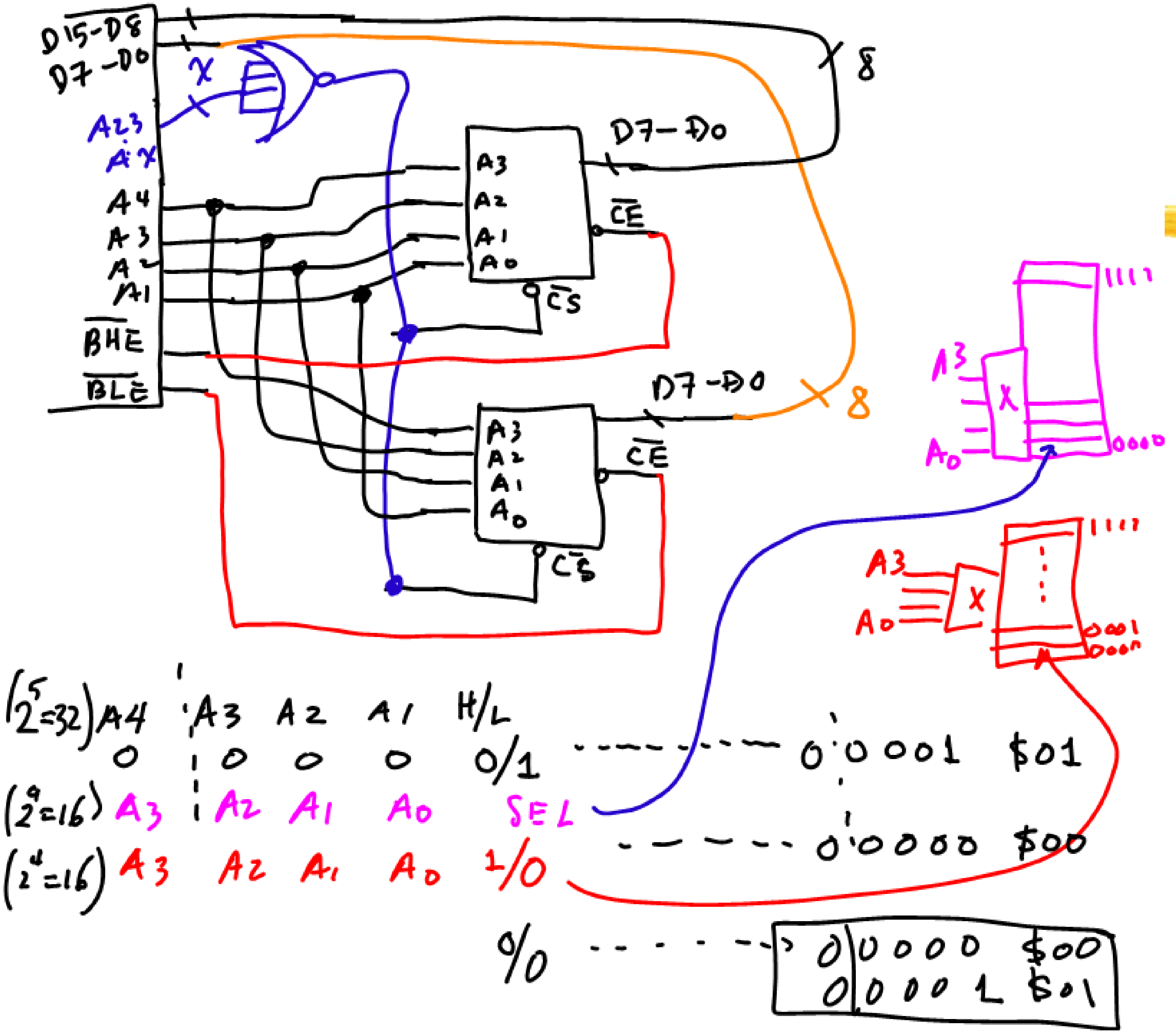
- Upper Data Strobe
  - For accessing upper byte of memory
  - D15 – D8 part of CPU
  - D7 – D0 part of memory
- Lower Data Strobe
  - For accessing lower byte of memory
  - D7-D0 part of CPU
  - D7 – D0 part of Memory
- Both together works as A0 line



$\overline{LDS}$	$\overline{UDS}$	
0	0	D15 ~ D0
0	1	D15 ~ D8
1	0	D7 ~ D0

Access

odd	EVEN
5	4
3	2
1	0



$(2^5=32)$ A4 A3 A2 A1 H/L

0 0 0 0 0/1 - - - - - - - - 0 0 0 0 1 \$01

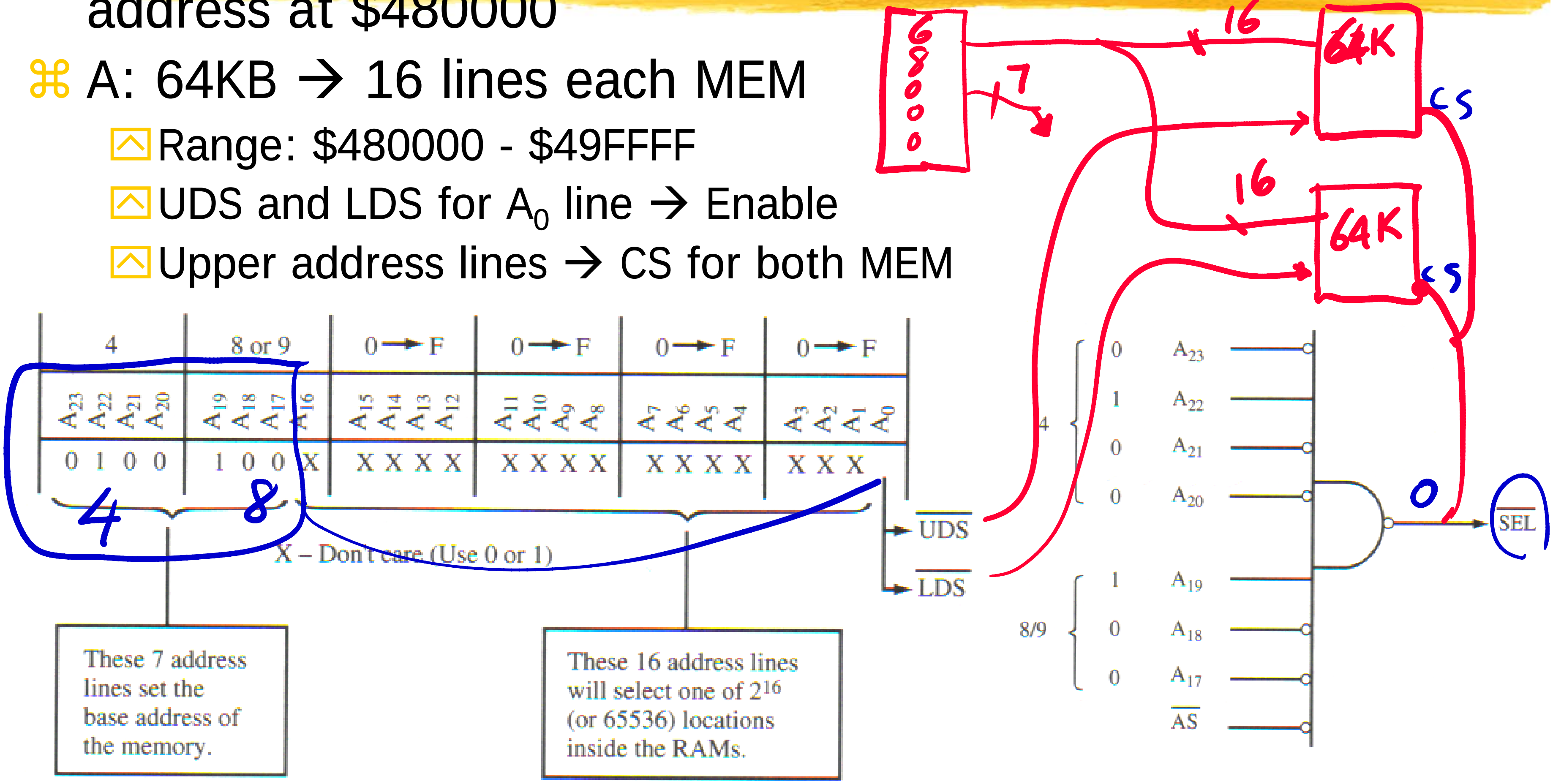
$(2^4=16)$ A3 A2 A1 A0 SEL

$(2^4=16)$ A3 A2 A1 A0 1/0 - - - - - - 0 0 0 0 0 \$00

%0 - - - - - - -> 0 0 0 0 0 \$00
0 0 0 0 1 \$01

Memory Decoding

- Q: 64K Word (or 128 KB) of RAM, with it's starting address at $480000
- A: 64KB → 16 lines each MEM
  - Range: $480000 - $49FFFF
  - UDS and LDS for A_0 line → Enable
  - Upper address lines → CS for both MEM



4	8 or 9	0 → F	0 → F	0 → F	0 → F
A_{23} A_{22} A_{21} A_{20}	A_{19} A_{18} A_{17} A_{16}	A_{15} A_{14} A_{13} A_{12}	A_{11} A_{10} A_9 A_8	A_7 A_6 A_5 A_4	A_3 A_2 A_1 A_0
0 1 0 0	1 0 0 X	X X X X	X X X X	X X X X	X X X

X – Don't care (Use 0 or 1)

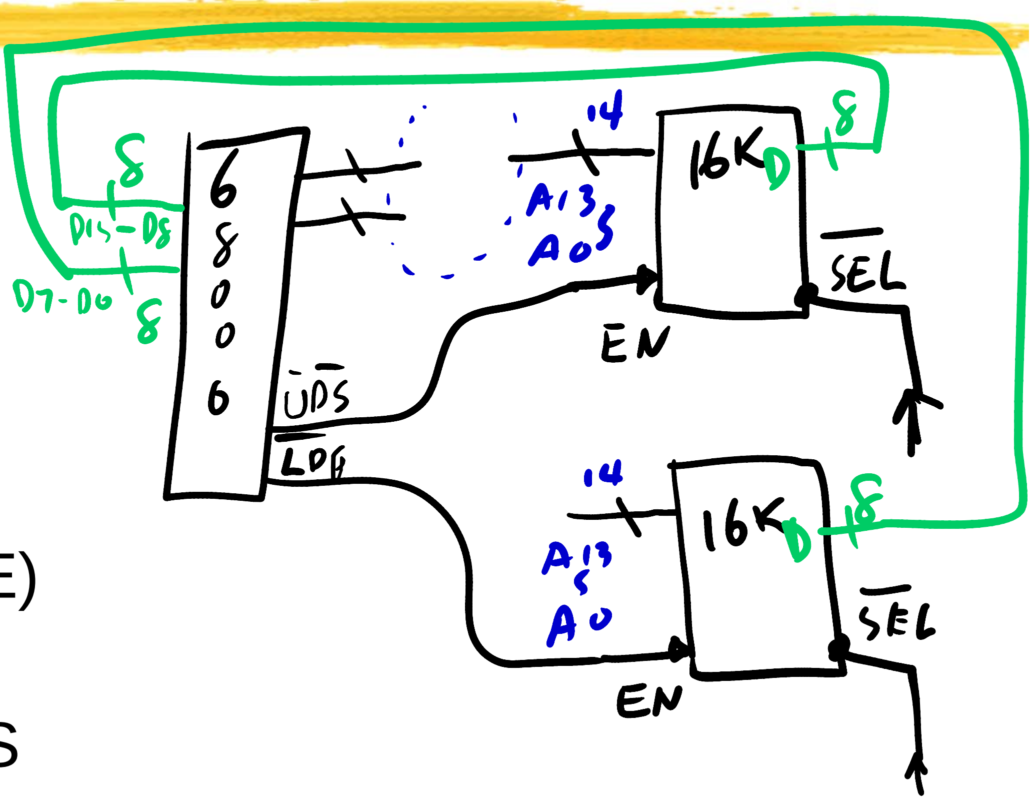
These 7 address lines set the base address of the memory.

These 16 address lines will select one of 2^{16} (or 65536) locations inside the RAMs.

Memory Decoding Example - 1

2 · 16KByte ROMs

- Q: 16K Word ROM with starting address at $300000.
- A: 16KB → 14 lines each MEM
  - $300000 - $307FFF
  - UDS (BHE)/LDS (BLE) → /CE
  - Upper Address → /CS



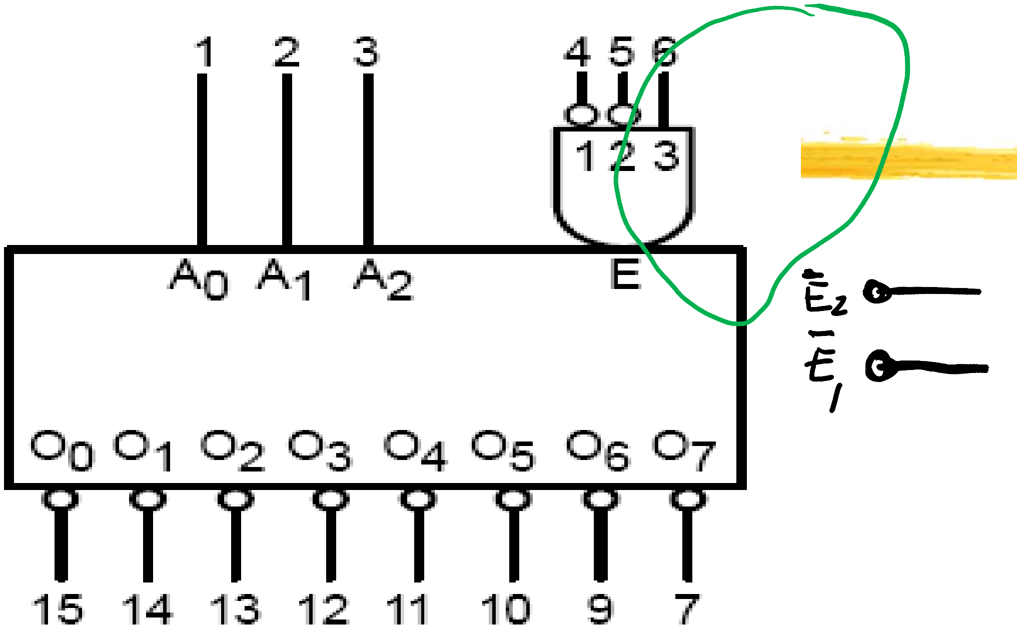
Decoder/Multiplexer

⌘74138

MOTOROLA

1-OF-8 DECODER/ DEMULTIPLEXER

LOGIC SYMBOL

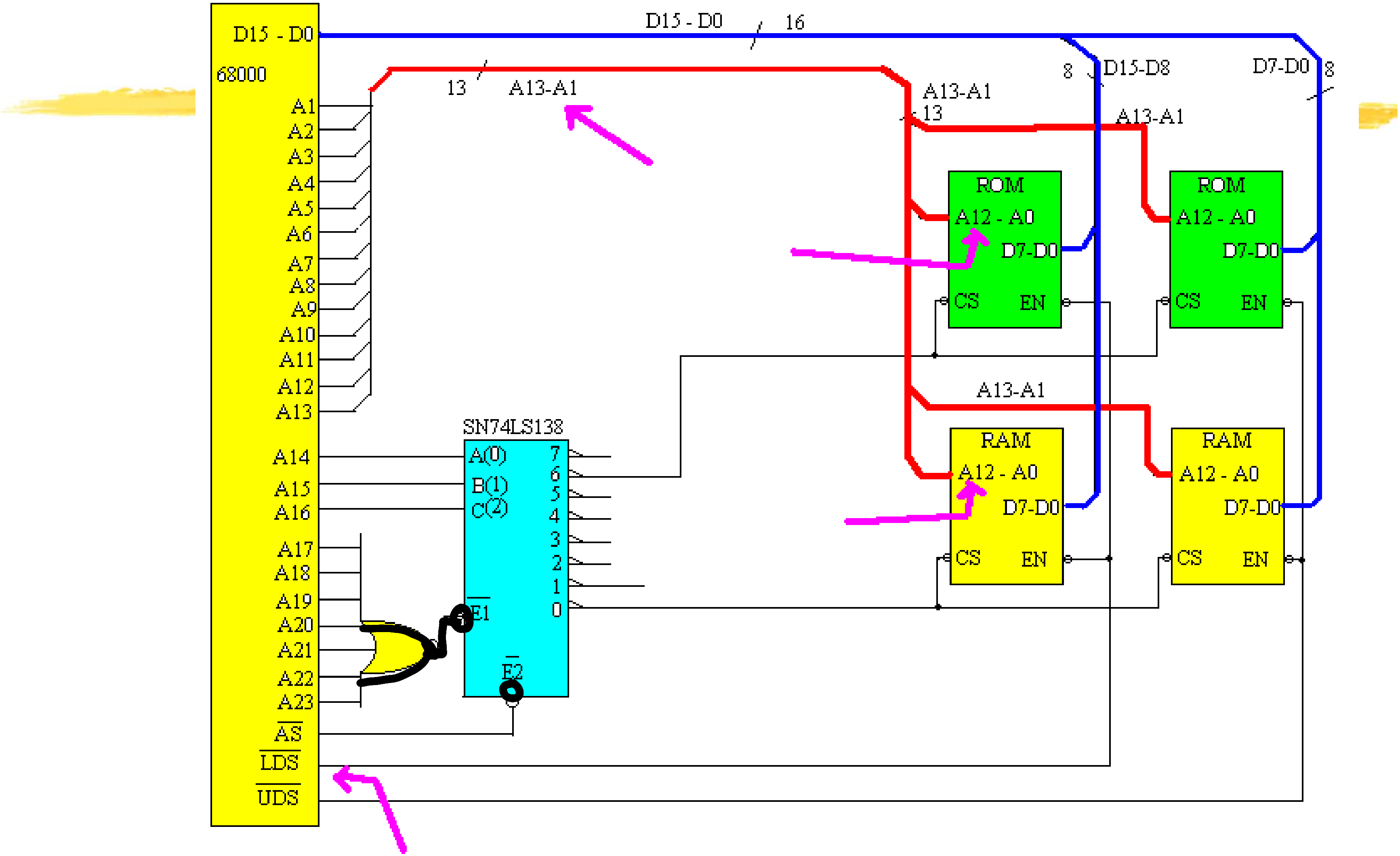


$\overline{E_1}$	$\overline{E_2}$
L	L

TRUTH TABLE

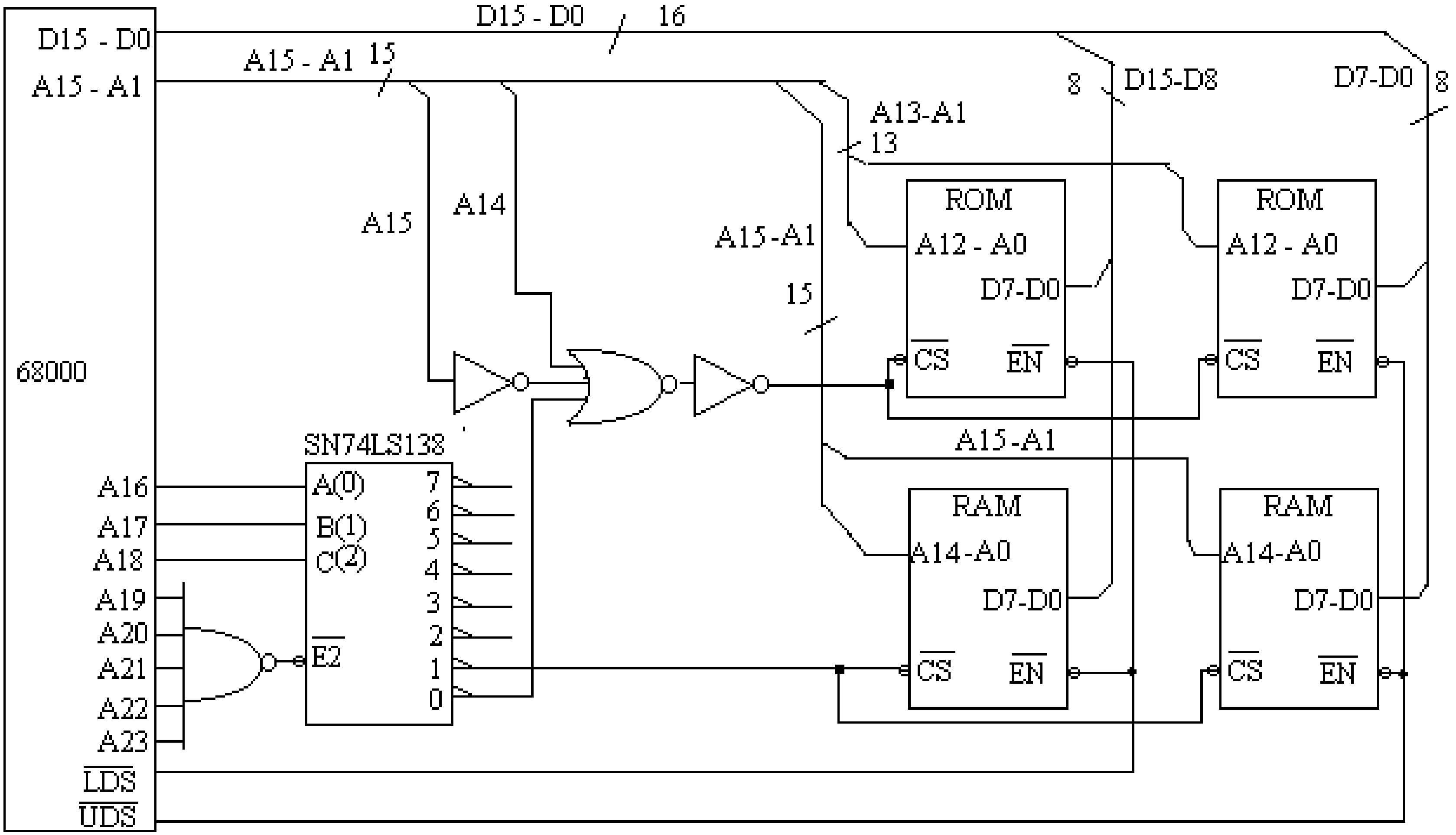
	2^0	2^1	2^2								
	INPUTS			OUTPUTS							
	A_0	A_1	A_2	$\overline{O_0}$	$\overline{O_1}$	$\overline{O_2}$	$\overline{O_3}$	$\overline{O_4}$	$\overline{O_5}$	$\overline{O_6}$	$\overline{O_7}$
	X	X	X	H	H	H	H	H	H	H	H
	X	X	X	H	H	H	H	H	H	H	H
	X	X	X	H	H	H	H	H	H	H	H
0	L	L	L	L	H	H	H	H	H	H	H
1	H	L	L	H	L	H	H	H	H	H	H
2	L	H	L	H	H	L	H	H	H	H	H
3	H	H	L	H	H	H	L	H	H	H	H
	L	L	H	H	H	H	H	L	H	H	H
	H	L	H	H	H	H	H	H	L	H	H
6	L	H	H	H	H	H	H	H	H	L	H
7	H	H	H	H	H	H	H	H	H	H	L

Memory Decoding with Byte/Word Access – 2 HW

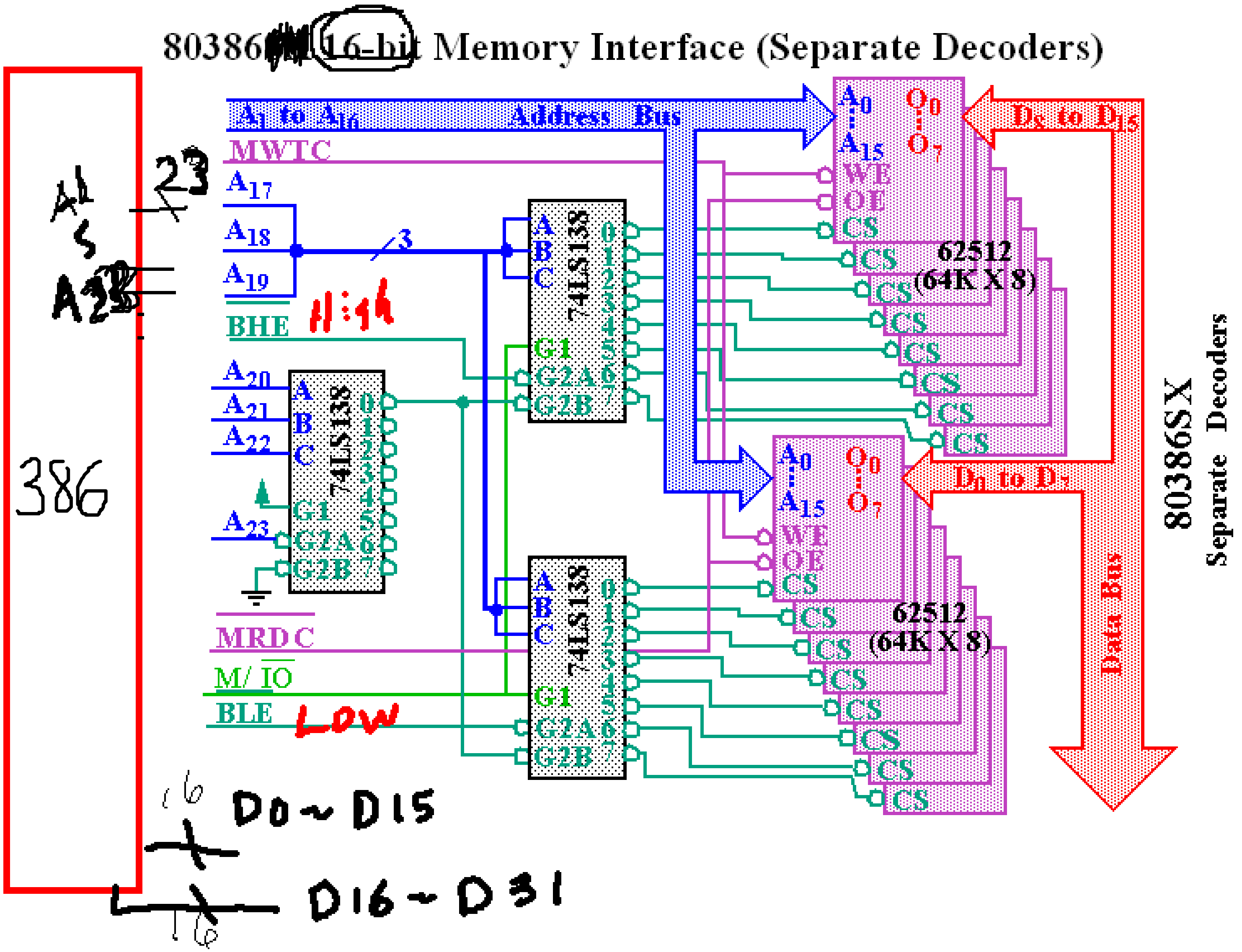


- Questions:
  - 1. Size of ROM
  - 2. Size of RAM
  - 3. Memory Map

Can You Draw a Memory Map of this? - 3



Intel 80386 Memory – Decoding – Class Activity

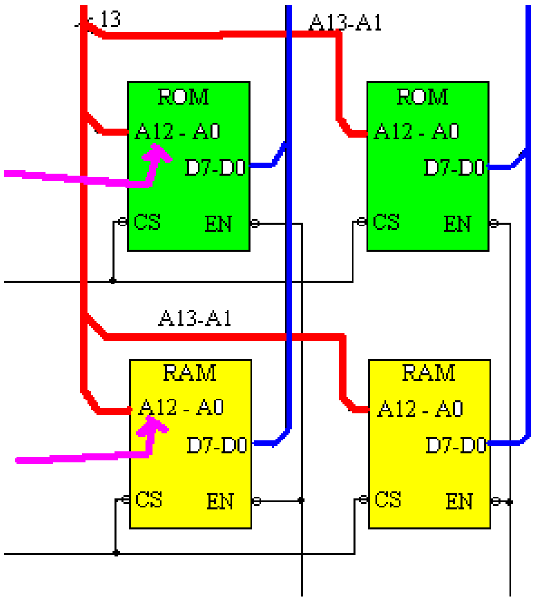


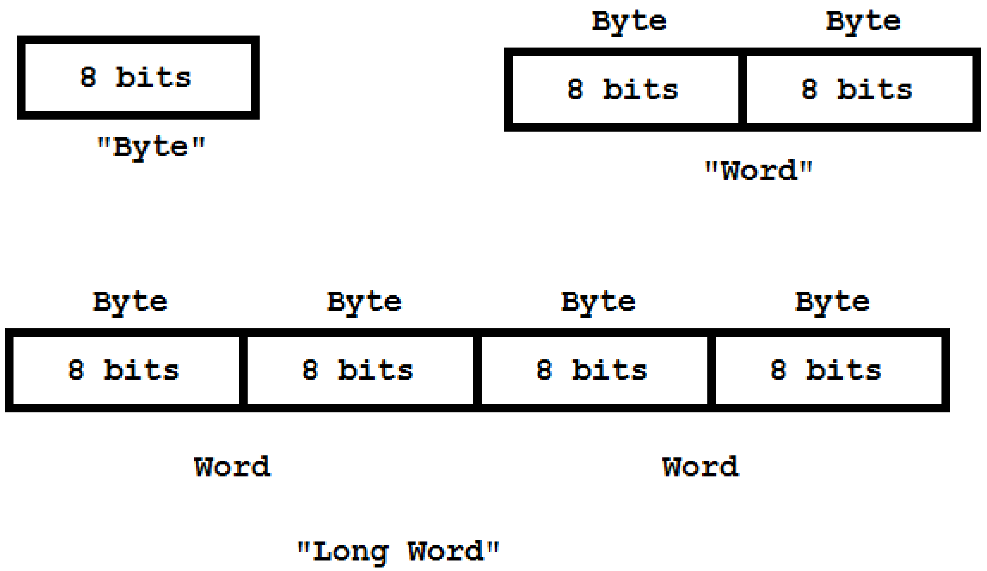
80386SX 16-bit Memory Interface (Separate Decoders)

- Find the address range of the **second** chips of the memory pair.
- Individual Work – Class Activity
- Name and Date

Multiple Address Access Issues

- 8-bit processor
  - Access one address with a byte data
- 16-bit processor
  - Can access two address spaces (Even and Odd)at a single execution with 2-byte (or “Word”) data
  - Where do we store each of the 2 bytes to each of the 2 address spaces?




8 bits

"Byte"

Byte | Byte

8 bits | 8 bits

"Word"

Byte | Byte | Byte | Byte

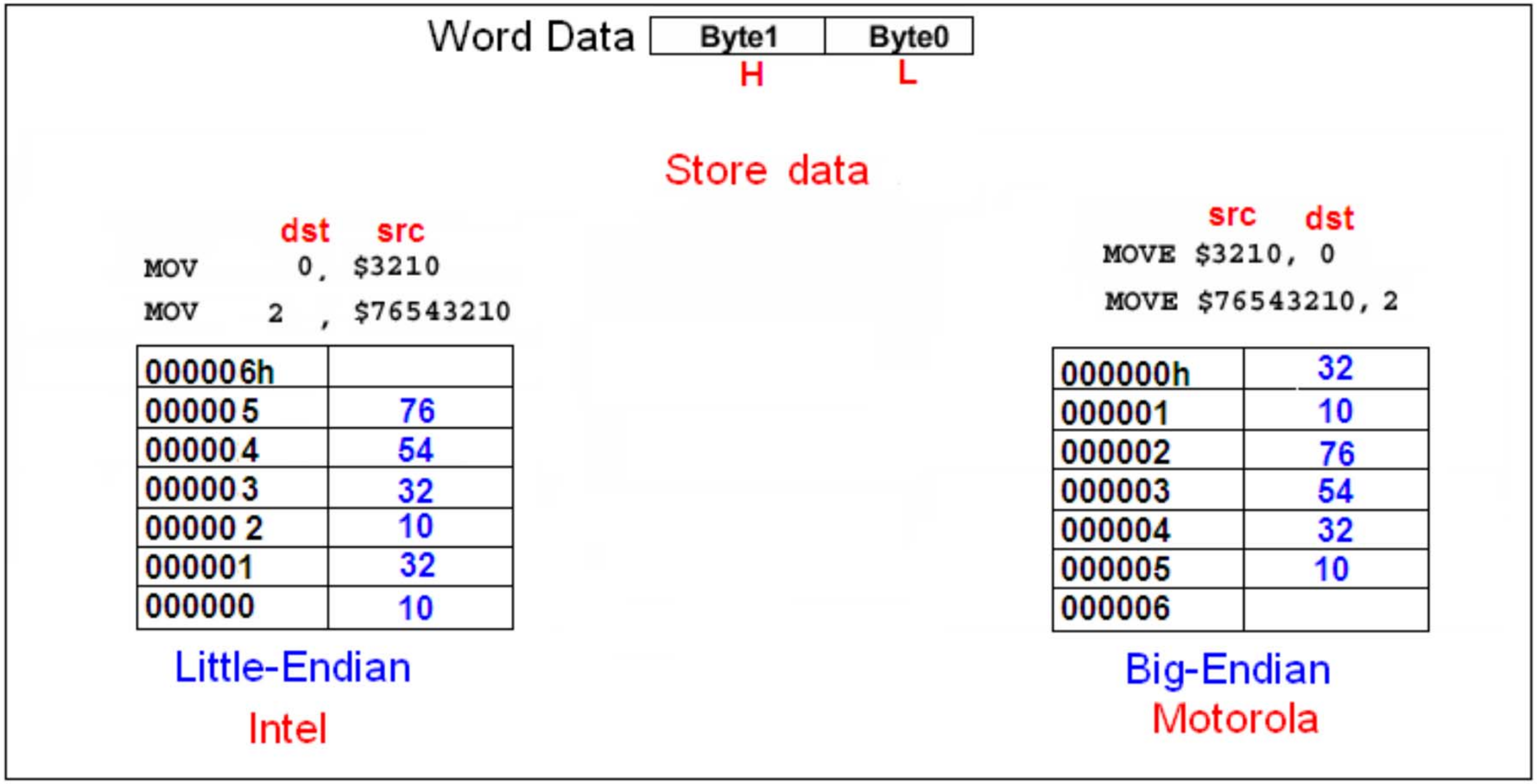
8 bits | 8 bits | 8 bits | 8 bits

Word | Word

"Long Word"

Big-Endian vs. Little-Endian

- Big-Endian: Words are stored with the lower 8- bits in the higher of the two storage locations: **Motorola**
  - “Big guy ends at lower address”
- Little- Endian: Lower-order byte stored in the lowest address) processors: **Intel** 80x86 family
  - Little guys ends at lower address”



Word Data

Byte1	Byte0
H	L

Store data

```
         dst   src
MOV        0,  $3210
MOV      2  ,  $76543210
```

000006h	
000005	76
000004	54
000003	32
00000 2	10
000001	32
000000	10

Little-Endian

Intel

```
     src     dst
MOVE $3210,  0
MOVE $76543210, 2
```

000000h	32
000001	10
000002	76
000003	54
000004	32
000005	10
000006	

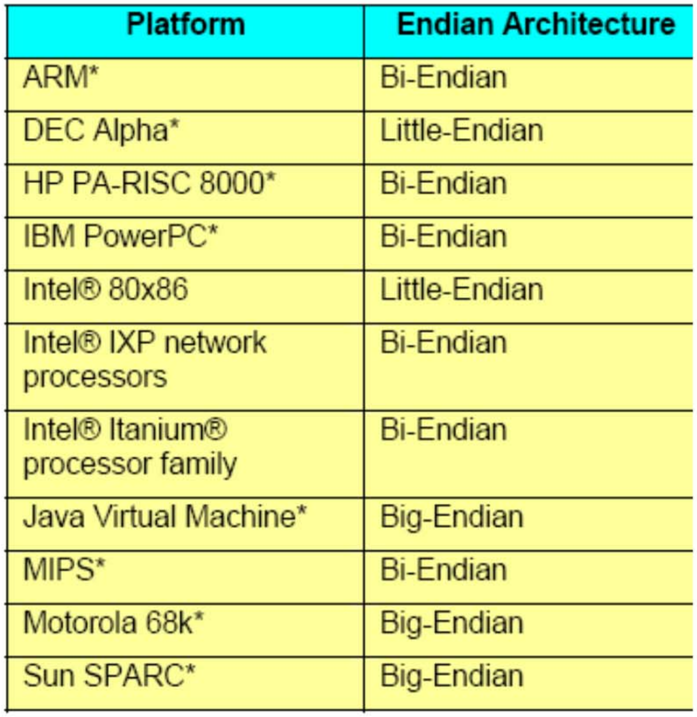
Big-Endian

Motorola

“Endianness”

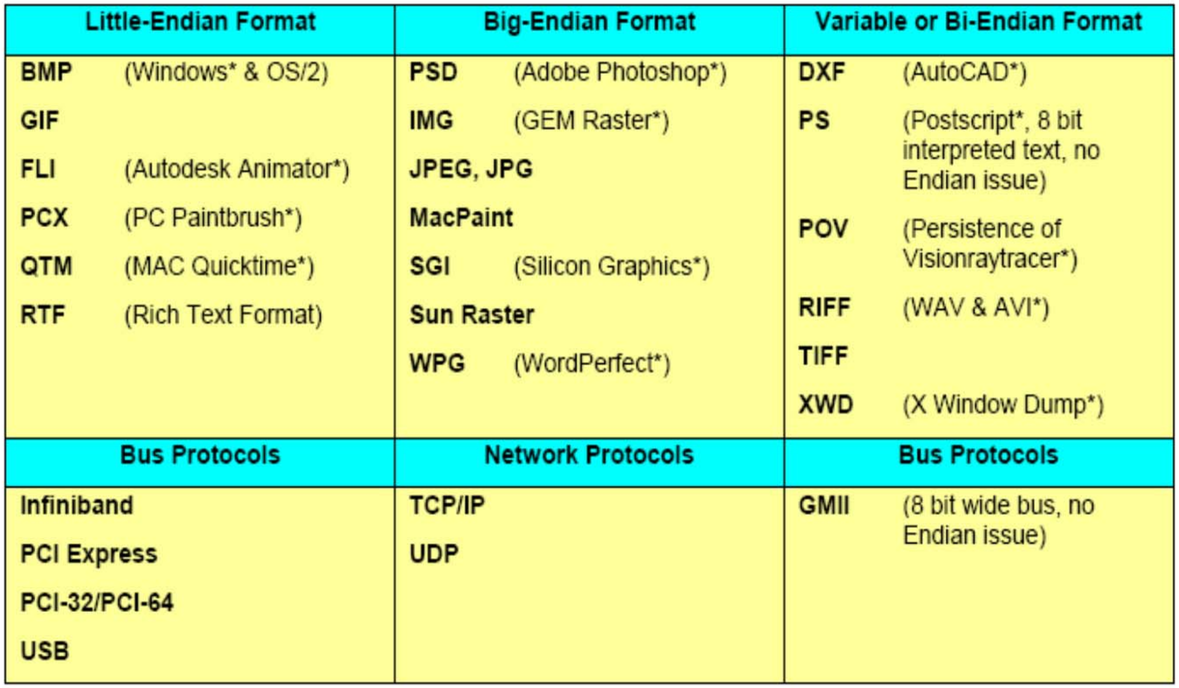
- Endian or Endian-Architecture
  - how multi-byte data is represented by a computer system and is dictated by the CPU architecture of the system
  - Not all computer systems are designed with the same endian architecture
  - Issues with software and interface

Computer System Endianness



Platform	Endian Architecture
ARM*	Bi-Endian
DEC Alpha*	Little-Endian
HP PA-RISC 8000*	Bi-Endian
IBM PowerPC*	Bi-Endian
Intel® 80x86	Little-Endian
Intel® IXP network processors	Bi-Endian
Intel® Itanium® processor family	Bi-Endian
Java Virtual Machine*	Big-Endian
MIPS*	Bi-Endian
Motorola 68k*	Big-Endian
Sun SPARC*	Big-Endian

Common file formats



Little-Endian Format	Big-Endian Format	Variable or Bi-Endian Format
BMP (Windows* & OS/2) **GIF** **FLI** (Autodesk Animator*) **PCX** (PC Paintbrush*) **QTM** (MAC Quicktime*) **RTF** (Rich Text Format)	**PSD** (Adobe Photoshop*) **IMG** (GEM Raster*) **JPEG, JPG** **MacPaint** **SGI** (Silicon Graphics*) **Sun Raster** **WPG** (WordPerfect*)	**DXF** (AutoCAD*) **PS** (Postscript*, 8 bit interpreted text, no Endian issue) **POV** (Persistence of Visionraytracer*) **RIFF** (WAV & AVI*) **TIFF** **XWD** (X Window Dump*)
Bus Protocols	**Network Protocols**	**Bus Protocols**
Infiniband **PCI Express** **PCI-32/PCI-64** **USB**	**TCP/IP** **UDP**	**GMII** (8 bit wide bus, no Endian issue)

Endian-Neutral Approaches

- Conversion
  - Byte Swap
  - Network I/O Macro
- “Endian Neutral”: allowing the code to be ported easily between processors of different Endian-architectures, and without rewriting any code. Endian-neutral software is developed by identifying system memory and external data interfaces, and using Endian-neutral coding practices to implement the interfaces.
- **HOMEWORK #2**
  - Technical Report on “Endian-Neutral Approaches”
  - What? Why? How?
  - 2 - 3 pages; 1” margin all sides; 11 pt; Times New Roman; No cover page (Title and your name at the top, and start in the first page); single space; single column; again **the importance of the first paragraph**. No figure, no photo, text only.
  - Submission: Hardcopy only by 5:00pm Thursday 10/10/2012.