SYSC 3601, Winter 2012, Review for Final Exam

Prof. V. Aitken, Ph.D., P.Eng.

These slides are taken mostly from the lecture notes and focus on material that is important for the final exam. Additional review material, not included in this slide deck, will be presented during the remaining lectures of this term. Be there!

Students who need additional assistance to prepare for the final exam must contact me ASAP via e-mail to arrange for meeting dates/times.

History of Intel x86 μP

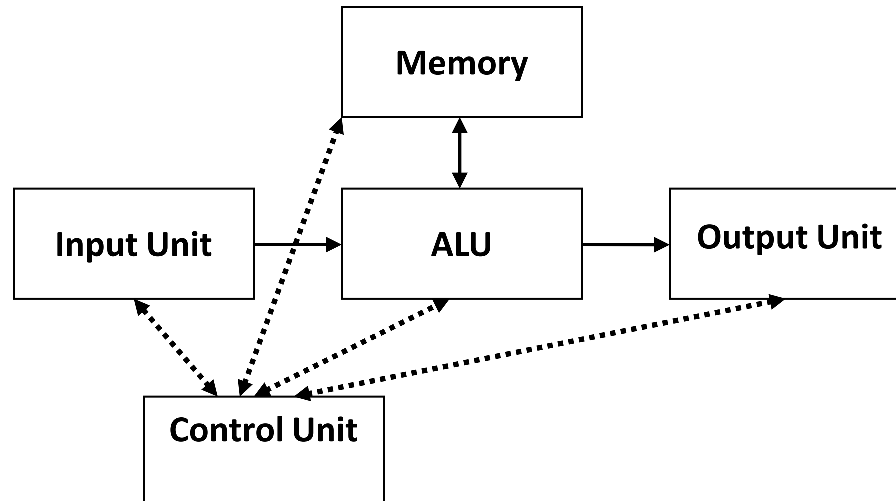
- 4004 the first microprocessor (4-bit) 16K RAM
- 8008 (8-bit)
- 8080 (8-bit) 64K RAM, 2Mhz clock
- 8088 (8 bit)
- 8086 (16-bit) 1M RAM, 5MHz clock
- 80286 (16-bit) 16M RAM, 16MHz clock
- 80386, 32-Bit, 4G RAM, 33 MHz clock
- 80486, 4G RAM, 66 MHz clock
- Pentium, 4G RAM, 66 MHz clock
- Pentium Pro, 64G RAM, 133 MHz clock
- Pentium II, 64G RAM, 233 MHz clock
- Pentium III, 64G RAM, 500 MHz clock
- Pentium 4, 64G RAM, 1.5 GHz clock

History of Motorola 680X0 μP

- **6800** - 1974, 8-bit.
- **68000** - 1979, 16-bit data, 24-bit address.
- **68008** - 8 bit data bus, 20 bit address bus.
- **68010** - 1982. Added virtual memory support.
- **68020** - 1984. Fully 32 bit. 3 stage pipeline.
  - 256 byte cache. More addressing modes!
- **68030** - 1987. Integrated MMU into chip.
- **68040** - 1991. Harvard architecture with two 4-k caches. FP on chip. 6 stage pipeline.
- **68060** - 1994. Superscalar version . 10-stage pipeline. 2 integer, 1 fp unit. 8k caches.

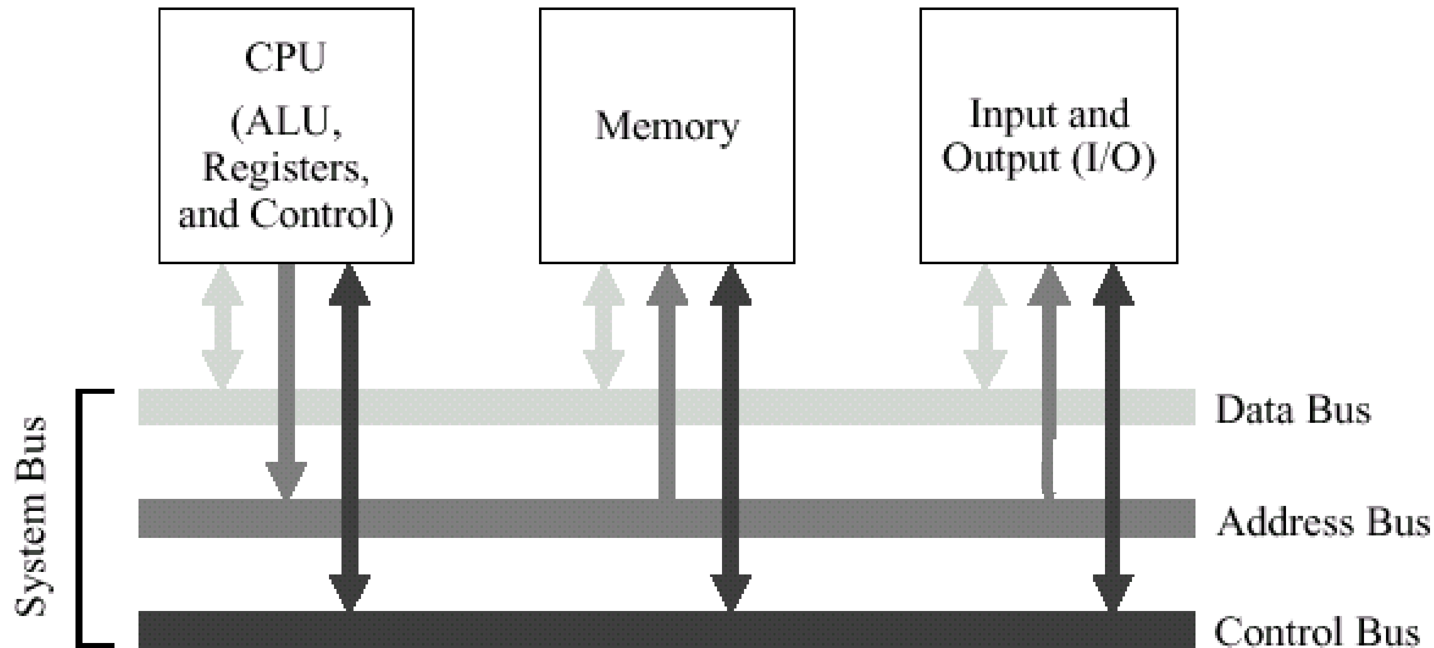
Von Neumann Model

- Consists of **5 major components**:
  - **Arithmetic and Logic Unit (ALU)**: Performs mathematical and logical operations on its operands
  - **Control Unit**: Produces control signals to orchestrate functioning of all other units (the boss!)
  - **Memory Unit**: Holds both data and program (in a stored program computer)
  - **Input Unit**: Obtains data from external sources
  - **Output Unit**: Provides data to external sources



System Bus Model1

- Refinement of the von Neumann Model
  - Same 5 components, but CPU (Central Processing Unit) or microprocessor now contains both ALU and Control Unit.
- All components are attached to a shared communication pathway called the **system bus**.



Memory

- Each addressable location is typically 1 byte of binary data
  - Each memory element (byte) has an address, usually specified in hexadecimal notation.
- Memory size chart:

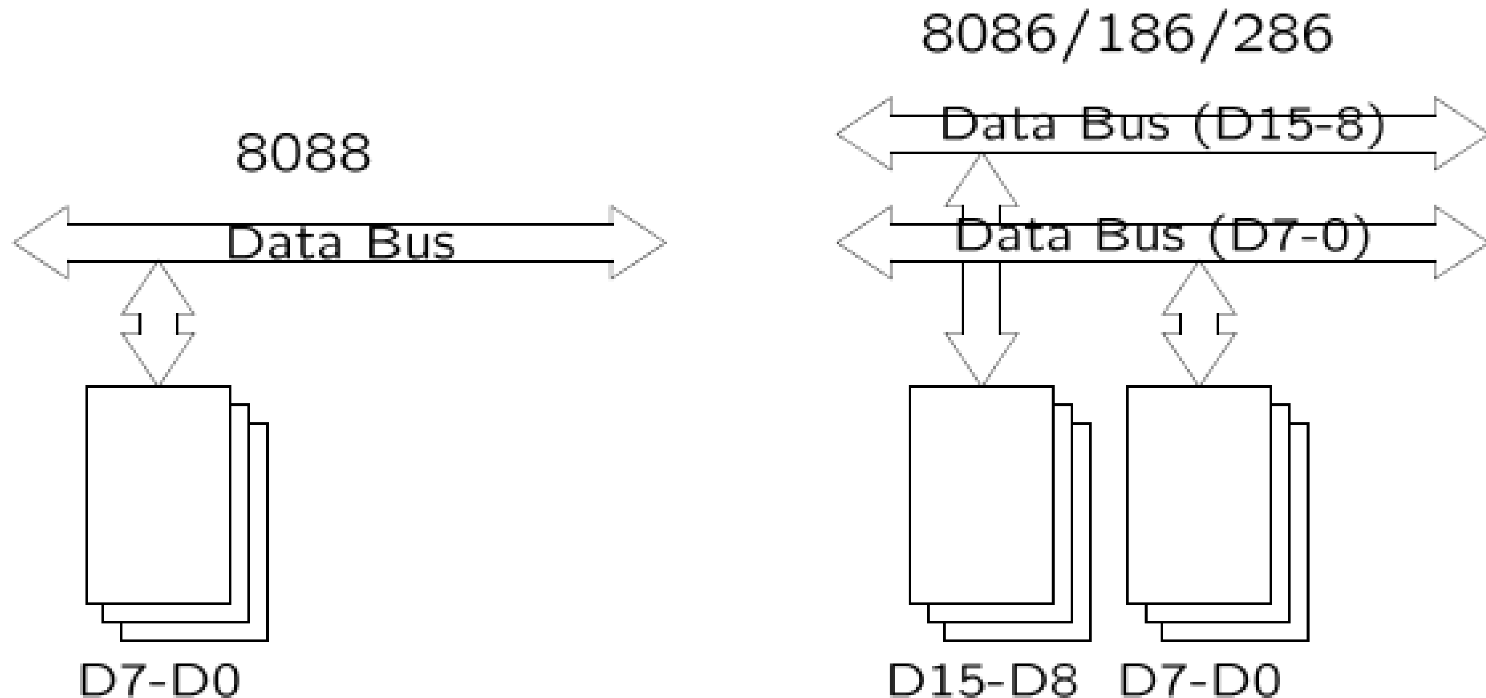
1KB	2^{10} bytes	1,024 bytes
1MB	2^{20} bytes	1,048,576 bytes
1GB	2^{30} bytes	1,073,741,824 bytes

- Ex: 64KB = 64 x 2^{10} bytes = 65536 bytes

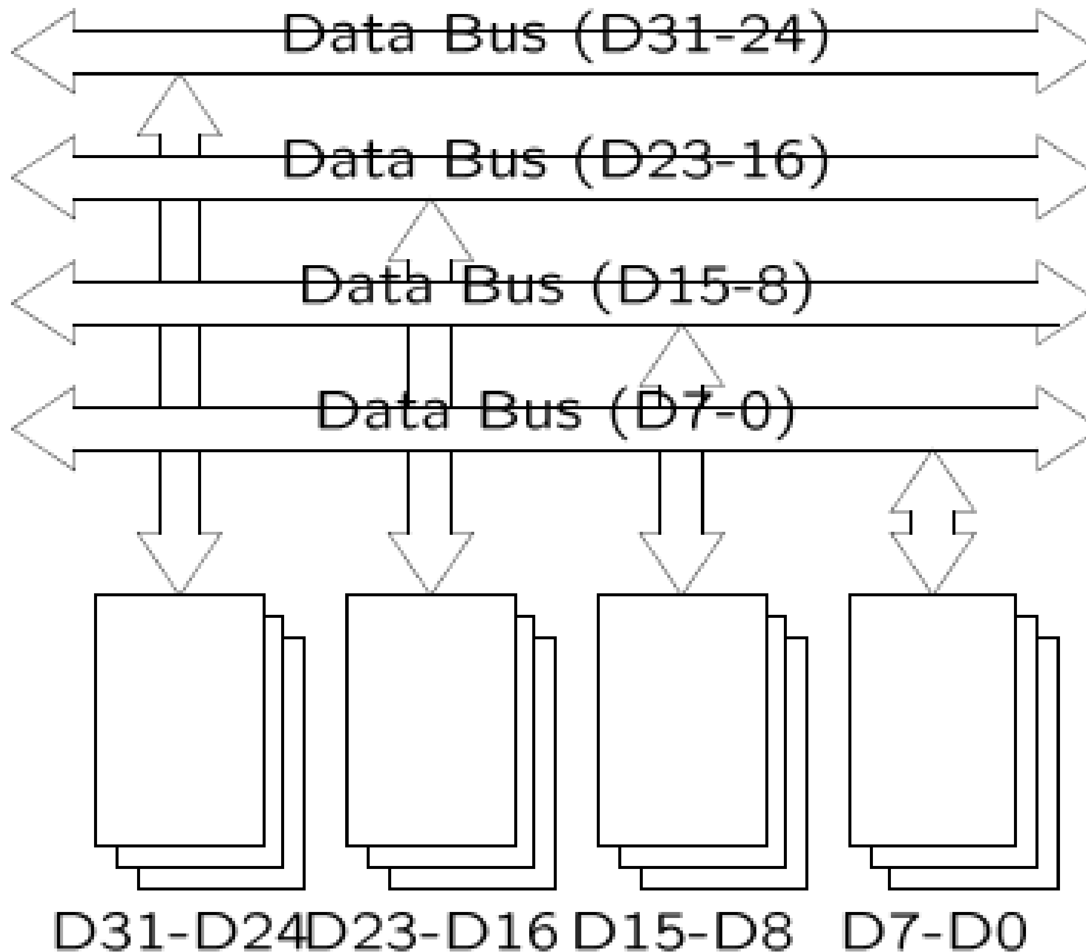
  64K = 2^{16} : need 16 address lines.

Memory Organization

- Memory devices are arranged in bytes of 8-bits (modulo parity/ECC)
- μP may have 8, 16, 32, or 64 data lines...more?
- Each memory chip returns a single byte
  - Therefore, multiple banks of memory chips are used.
- Each bank requires a 'bank enable' signal



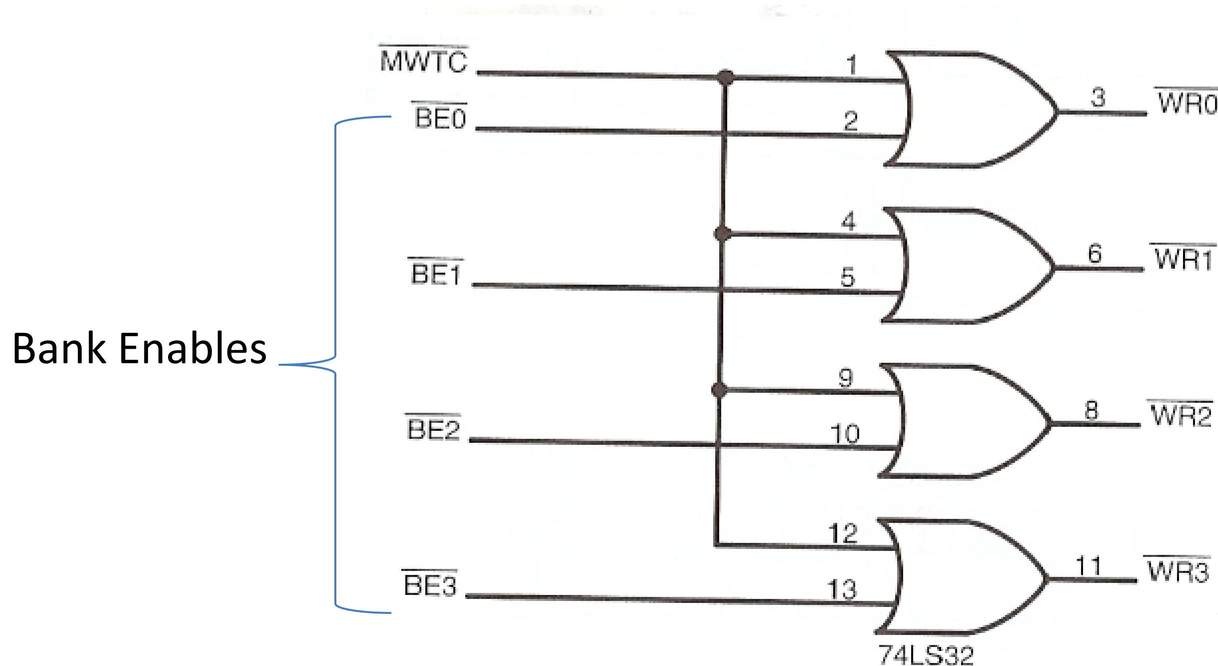
Memory Organization – 32 bit data bus



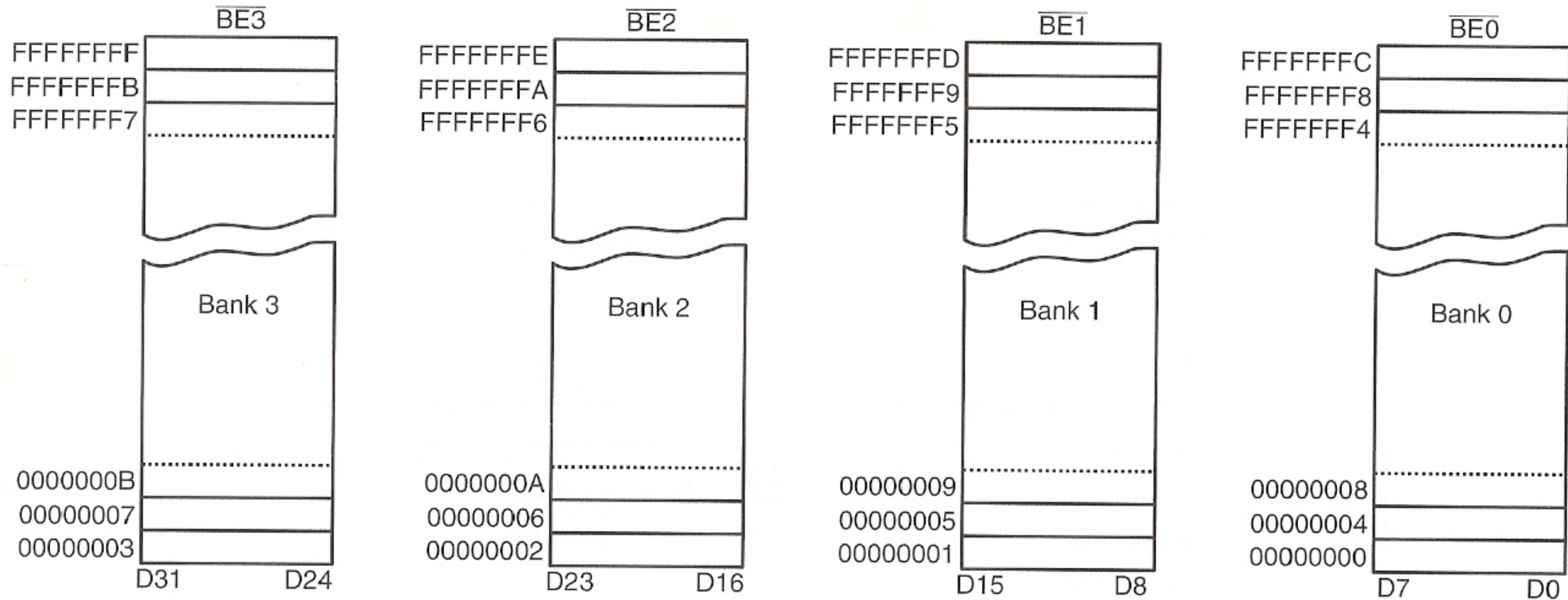
- What do we do for a 64 bit data bus? 128?
- Addresses depend on Little vs Big Endien

32-bit Wide Memory

- Requires 4 banks, each 8-bits wide to generate (up to) 32-bits per read/write
- Bank ID is system address ‘mod 4’
- No A0, or A1 address pins (Why?)
- Requires 4 bank enable signals for writes:



32-bit Data and 32-bit Address Intel Memory (‘386DX)



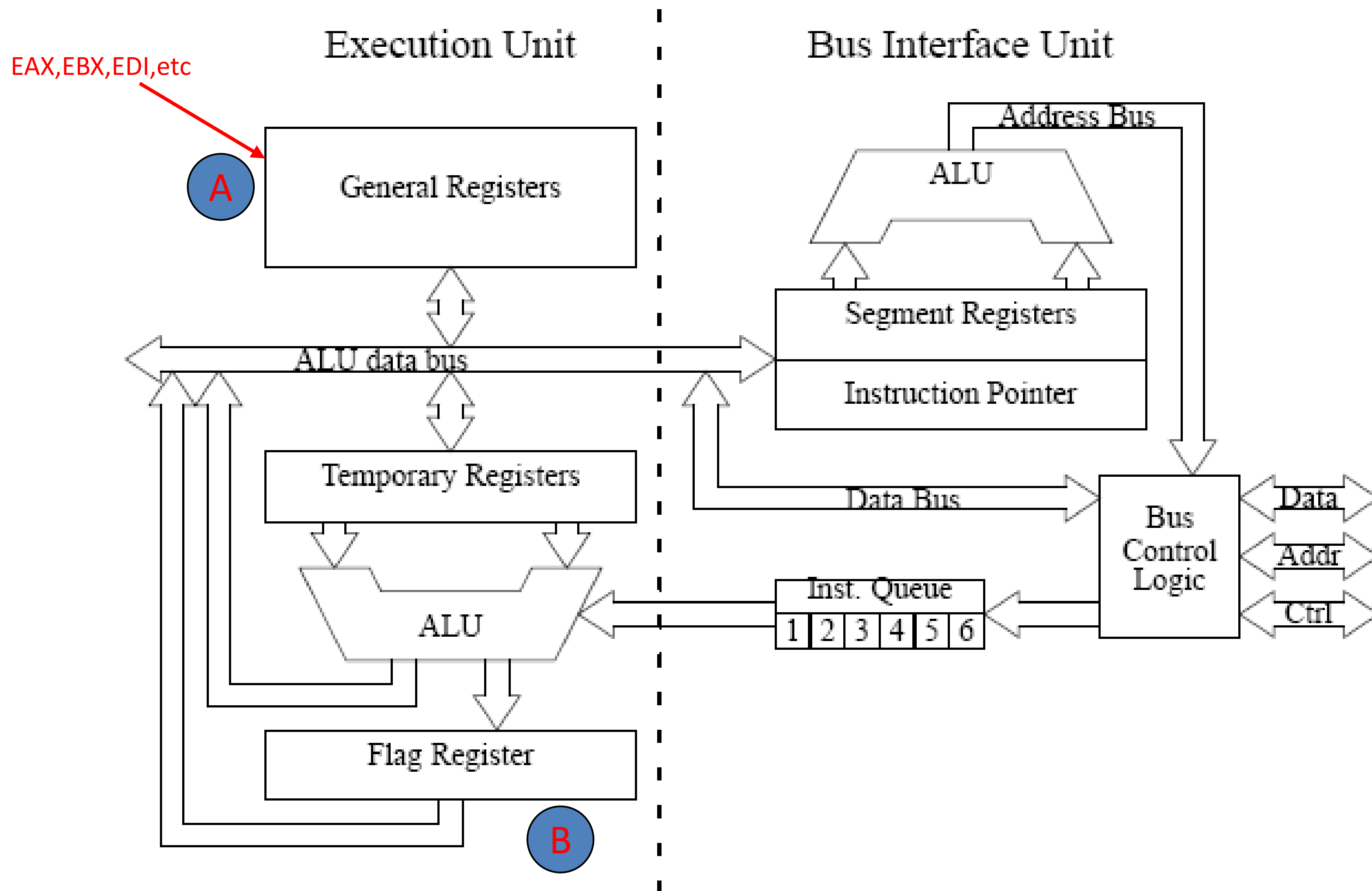
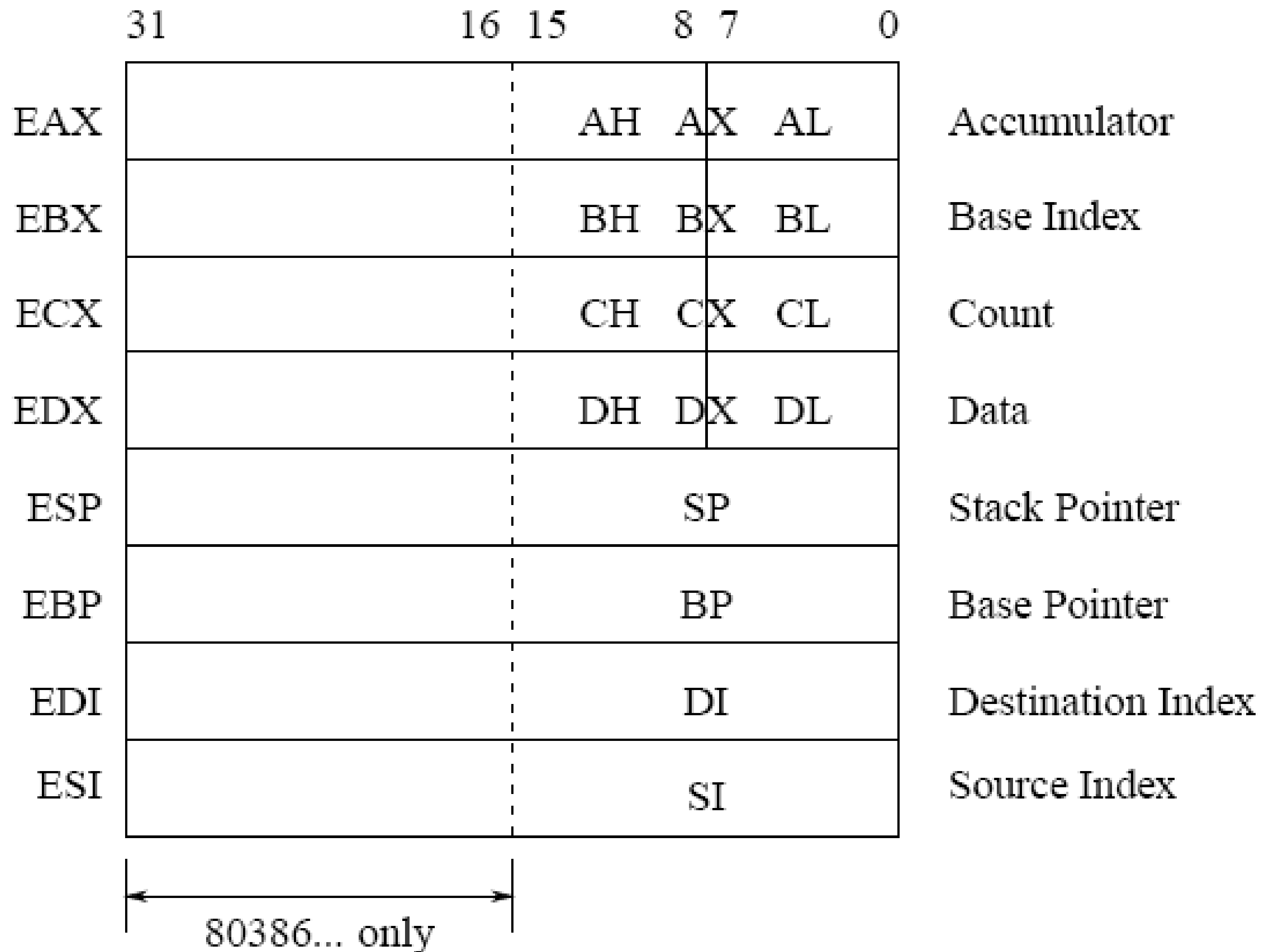
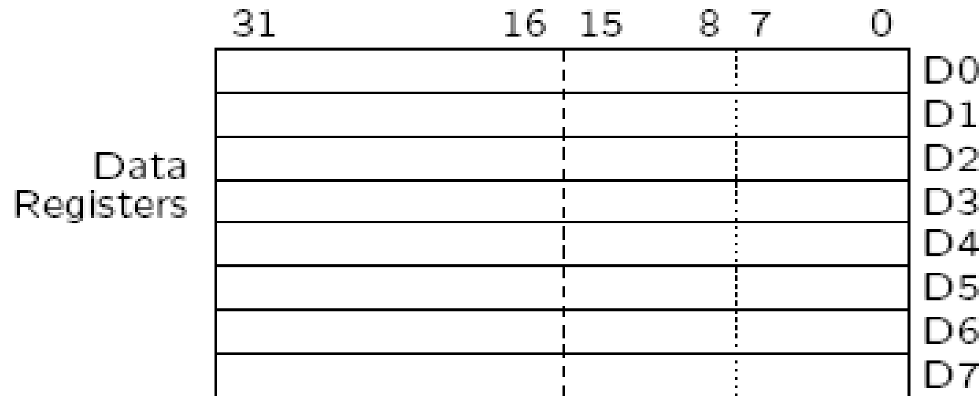


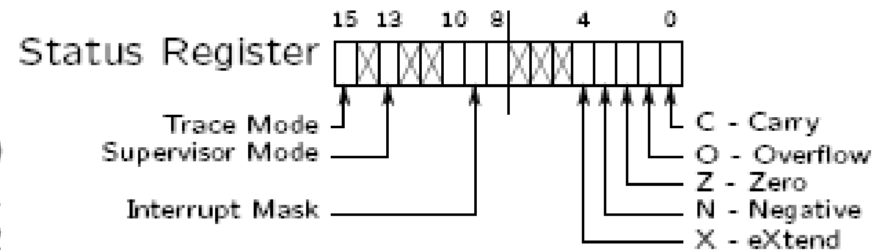
Figure 1: Internal Architecture of the 8086/88

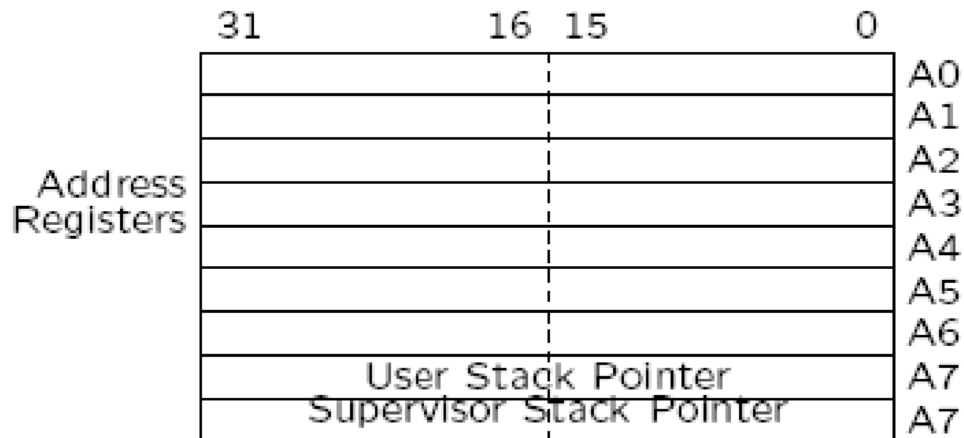
Intel Execution Unit – Programming Model



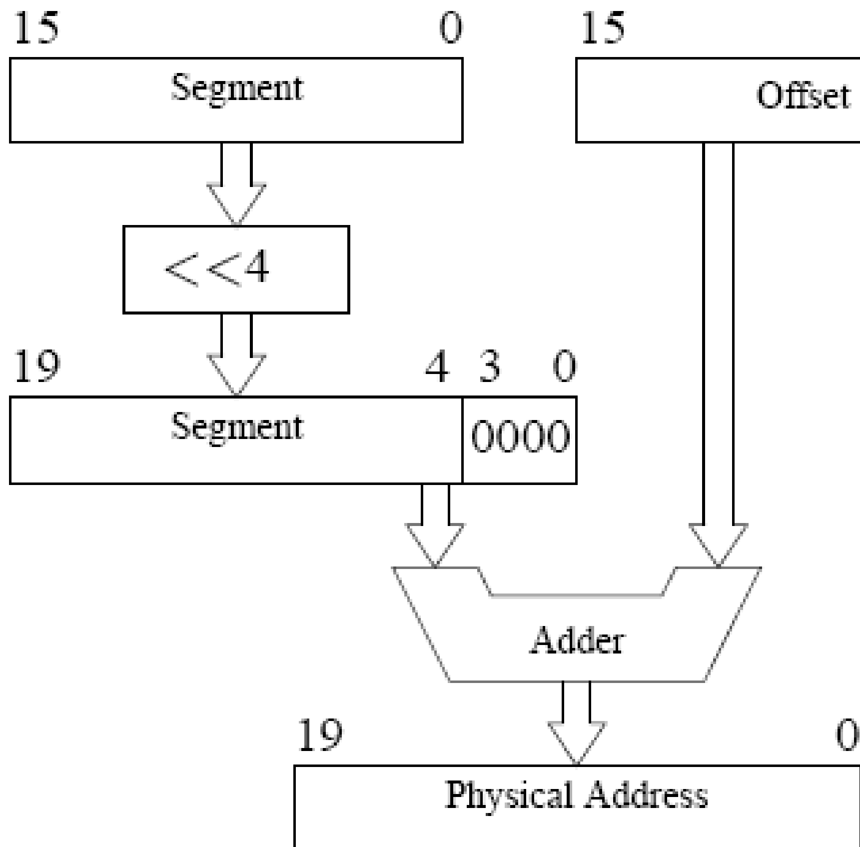
Motorola 68000 μP – Programming Model







Intel Real Mode Address Generation



Ex. If IP=1200H and CS=1400H
then next instruction will be fetched from:

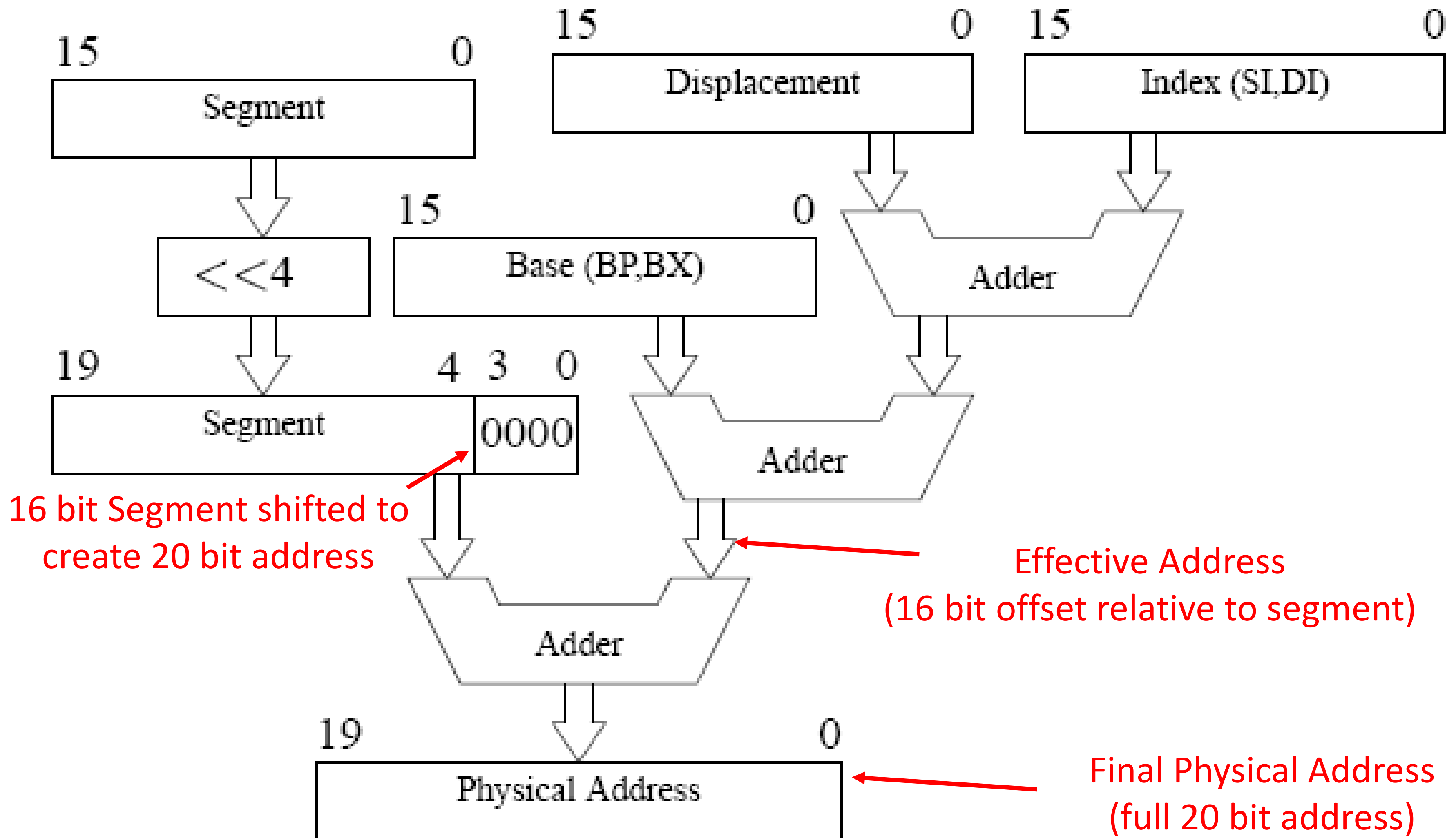
```
1400:1200

       or

 14000H
 +1200H
 ------
 15200H
```

Not done in Motorola...use lower 24 bits of internal 32-bit address registers

Intel Addressing Modes - Effective Address (EA)



Motorola 68000 µP – Addressing Modes

- There are 14 different addressing modes (more with the 68020!)

Mode	Syntax
Data reg direct	d_n, n = 0..7
Addr reg direct	a_n, n = 0..7
Addr reg indirect	(a_n)
with Postincrement	$(a_n)+$
with Predecrement	$-(a_n)$
with Displacement	$d_{16}(a_n)$
with Index	$d_8(a_n,X_m)$ (X_m is any a_m or d_m)
Relative with offset	d_{16}(PC)
Relative with index and offset	$d_8(PC,X_n)$
Absolute short	< … > (16-bits sign-extended to 32) (for 000000-007FFF or FF8000-FFFFFF)
Absolute long	< … > (32-bits)
Immediate	#< … >
Quick immediate	#< … > (1 byte, sign-extend to 32)
Implied	Register specified as part of mnemonic

Motorola 68000 μP – Addressing Mode Examples

- Example: A sample assembler subroutine for the 68000:

Total: Find the sum of 16-bytes stored in memory.

```
        org       $8000          ;load program counter
total   clr.w     d0             ;clear D0.
        move.b    #16,d1         ;initialize counter
        movea.l   #data,a0       ;init pointer to data
loop    add.b     (a0)+,d0       ;add byte, increment address
        subq.b    #1,d1          ;decrement counter
        bne       loop           ;test for zero, branch not equal.
        movea.l   #sum,a1        ;load address to store result
        move.w    d0,(a1)        ;store sum at sum
        rts                      ;return from subroutine.

sum     dc.w      0              ;save room for result.
data    ds.b      16             ;save room for 16 data bytes.
        end
```

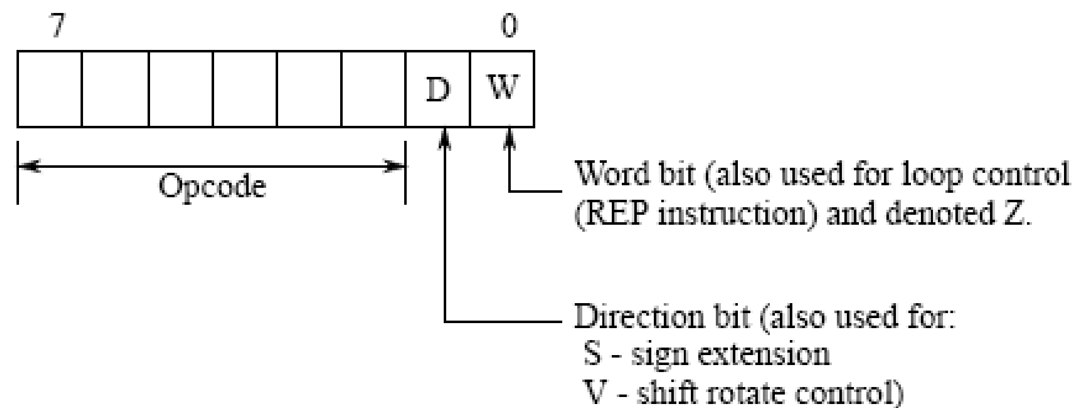
- Note:
  - `dc.w` - define a constant word, operand specifies the value to be written.
  - `ds.b` - define storage byte, operand specifies number of bytes, but not the contents

Intel Assembly and Machine Language

- 16 bit mode instructions take the form:

Opcode++ 1-2 bytes	MOD-REG-R/M 0-1 byte	Displacement 0-2 bytes	Immediate 0-2 bytes

- OPCODE^{++}
  - Typically 1 byte, but not always!
  - Selects the operation (MOV, ADD, JMP)



Opcode++ 1-2 bytes	MOD-REG-R/M 0-1 byte	Displacement 0-2 bytes	Immediate 0-2 bytes

"data" (Displacement + Immediate)

Direction:

D=0	SRC=REG
D=1	DST=REG

Word:

W=0	8-Bit
W=1	16-Bit

"reg" Field Bit Assignments:

16-Bit (w=1)	8-Bit (w=0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

"r/m" Field Bit Assignments:

r/m	Operand Address
000	(BX) + (SI) + DISP
001	(BX) + (DI) + DISP
010	(BP) + (SI) + DISP
011	(BP) + (DI) + DISP
100	(SI) + DISP
101	(DI) + DISP
110	(BP) + DISP
111	(BX) + DISP

"mod" Field Bit Assignments:

mod	Displacement
00	DISP = 0*, disp-low and disp-high are absent
01	DISP = disp-low sign-extended to 16-bits, disp-high is absent
10	DISP = disp-high:disp-low
11	r/m is treated as "reg" field

*except if mod = 00 and r/m = 110 then EA = disp-high:disp-low

Effective Address Calculation Time

EA Components		Clocks
Displacement Only		6
Base or Index Only	(BX,BP,SI,DI)	5
Disp + Base or Index	(BX,BP,SI,DI) + DISP	9
Base + Index	BP + DI, BX + SI	7
	BP + SI, BX + DI	8
Disp + Base + Index	BP + DI + DISP BX + SI + DISP	11
	BP + SI + DISP BX + DI + DISP	12

MOV – MOVE (BYTE OR WORD)

Operation (DEST)←(SRC)

Flags Affected None

Description MOV *destination, source*

MOVE transfers a byte or word from the source to the destination operand.

Encoding

Memory or Register Operand to/from Register Operand

1 0 0 0 1 0 d w	mod reg r/m	"data"

If d = 1 then SRC = EA, DEST = REG, else SRC = REG, DEST = EA.

Immediate Operand to Memory or Register Operand

1 1 0 0 0 1 1 w	mod 0 0 0 r/m	"data"

SRC = data, DEST = EA.

Immediate Operand to Register

1 0 1 1 w reg	"data"

SRC = data, DEST = REG.

Memory Operand to Accumulator

1 0 1 0 0 0 0 w	addr-low	addr-high

If w = 0 then SRC = addr, DEST = AL, else SRC = addr, DEST = AX.

Accumulator to Memory Operand

1 0 1 0 0 0 1 w	addr-low	addr-high

If w = 0 then SRC = AL, DEST = addr, else SRC = AX, DEST = addr.

Memory or Register Operand to/from Segment Register

1 0 0 0 1 1 d 0	mod 0 reg r/m	"data"

d must be set such that segment register is encoded by 2-bit reg field. CS cannot be DST.

MOV Operands	Clocks	Transfers	Bytes	MOV Coding Example
memory, accumulator	10	1	3	MOV ARRAY, AL
accumulator, memory	10	1	3	MOV AX, TEMP_RESULT
register, register	2	-	2	MOV AX, CX
register, memory	8 + EA	1	2-4	MOV BP, STACK_TOP
memory, register	9 + EA	1	2-4	MOV COUNT[DI], CX
register, immediate	4	-	2-3	MOV CL, 2
memory, immediate	10 + EA	1	3-6	MOV MASK[BX][SI], 2CH
seg-reg, reg16	2	-	2	MOV ES, CX
seg-reg, mem16	8 + EA	1	2-4	MOV DS, SEGMENT_BASE
reg16, seg-reg	2	-	2	MOV BP, SS
memory, seg-reg	9 + EA	1	2-4	MOV [BX]SEG_SAVE,CS

ADD – ADDITION

Operation (DEST)←(LSRC) + (RSRC)

Flags Affected AF, CF, OF, PF, SF, ZF

Description ADD *destination, source*

The sum of the two operands, which may be bytes or words, replaces the destination operand. Both operands may be signed or unsigned binary numbers (see AAA and DAA). ADD updates AF, CF, OF, PF, SF and ZF.

Encoding

Memory or Register Operand with Register Operand

0 0 0 0 0 0 d w	mod reg r/m	*"data"*

If d=1 then LSRC = REG, RSRC = EA, DEST = REG,
else LSRC = EA, RSRC = REG, DEST = EA.

Immediate Operand to Memory or Register Operand

1 0 0 0 0 0 s w	mod 0 0 0 r/m	*"data"*

LSRC = EA, RSRC = data, DEST = EA.

Immediate Operand to Accumulator

0 0 0 0 0 1 0 w	"data"

If w=0 then LSRC = AL, RSRC = data, DEST = AL,
else LSRC = AX, RSRC = data, DEST = AX.

ADD Operands	Clocks	Transfers	Bytes	ADD Coding Example
register,register	3	-	2	`ADD CX, DX`
register,memory	9 + EA	1	2-4	`ADD DI, [BX]ALPHA`
memory,register	16 + EA	2	2-4	`ADD TEMP, CL`
register,immediate	4	-	3-4	`ADD CL,2`
memory,immediate	17 + EA	2	3-6	`ADD ALPHA,2`
accumulator,immediate	4	-	2-3	`ADD AX, 200`

DEC – DECREMENT

Operation (DEST)←(DEST) – 1

Flags Affected AF, OF, PF, SF, ZF

Description DEC (Decrement) subtracts one from the destination operand. The operand may be a byte or a word and is treated as an unsigned binary number (see AA and DAA). DEC updates AF, OF, PF, SF and ZF; it does not affect CF.

Encoding

8-bit Register or Memory

1 1 1 1 1 1 1 w	mod 0 0 1 r/m

DEST = EA.

16-bit Register Operand

0 1 0 0 1 reg

DEST = REG.

DEC Operands	Clocks	Transfers	Bytes	DEC Coding Example
reg16	3	-	1	`DEC AX`
reg8	3	-	2	`DEC AL`
memory	15 + EA	2	2-4	`DEC ARRAY[SI]`

JNE – JUMP ON NOT EQUAL / JNZ – JUMP ON NOT ZERO

Operation If (ZF) = 0 then (IP)←(IP) + disp (sign extended to 16-bits)

Flags Affected none

Description JNE (Jump on Not Equal to)/JNZ (Jump on Not Zero) transfers control to the target operand (IP + displacement) if the condition tested (ZF = 0) is true.

Encoding

0 1 1 1 0 1 0 1	Disp

JNE/JNZ Operands	Clocks	Transfers	Bytes	DEC Coding Example
Short-label	16 or 4	-	2	`JNE NOT_EQUAL`

Assembly and Machine Language

- **Example**: Base relative + index (memory) to register

```
MOV AX,[BX+DI+1234H]
```

100010	D	W	MOD	REG	R/M	Displacement

Opcode:	100010	
D:	1	*Must be 1, dest AX specified by REG*
W:	1	*16 bit transfer*
MOD:	10	*16-bit displacement*
REG:	000	*AX*
R/M:	001	

Machine instruction is: **What?**

Example: Decode the following 8086 instructions:

- There are two **MOV** statements and one **Add** statement
- Give **Starting Address** of each statement and full **ASM** code

Addr	Data D15-D8	Data D7 – D0	Addr
ABCD:1013	...	...	ABCD:1012
ABCD:1011	...	FB	ABCD:1010
ABCD:100F	EB	12	ABCD:100E
ABCD:100D	34	81	ABCD:100C
ABCD:100B	8B	12	ABCD:100A
ABCD:1009	34	C3	ABCD:1008
ABCD:1007	81	12	ABCD:1006
ABCD:1005	34	B8	ABCD:1004
ABCD:1003	FB	EB	ABCD:1002
ABCD:1001	07	...	ABCD:1000

8086/8088 Pin Assignments & Functions

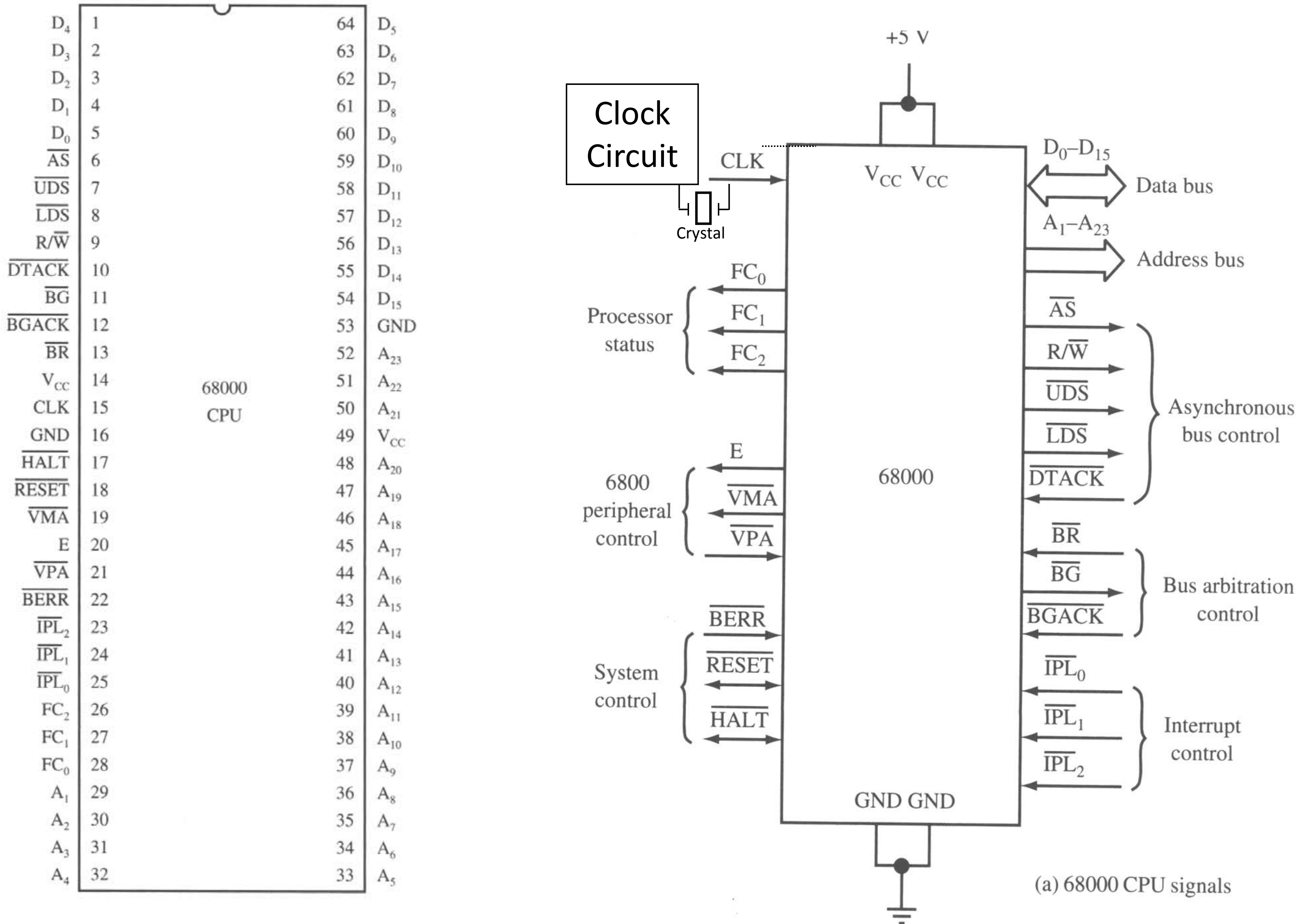
8086 CPU

Signal	Pin	Pin	Signal	Max mode
GND	1	40	Vcc	
AD14	2	39	AD15	
AD13	3	38	A16/S3	
AD12	4	37	A17/S4	
AD11	5	36	A18/S5	
AD10	6	35	A19/S6	
AD9	7	34	$\overline{BHE}$/S7	
AD8	8	33	MN/$\overline{MX}$	
AD7	9	32	$\overline{RD}$	
AD6	10	31	HOLD	($\overline{RQ}$/$\overline{GT0}$)
AD5	11	30	HLDA	($\overline{RQ}$/$\overline{GT1}$)
AD4	12	29	$\overline{WR}$	($\overline{LOCK}$)
AD3	13	28	M/$\overline{IO}$	($\overline{S2}$)
AD2	14	27	DT/$\overline{R}$	($\overline{S1}$)
AD1	15	26	$\overline{DEN}$	($\overline{S0}$)
AD0	16	25	ALE	(QS0)
NMI	17	24	$\overline{INTA}$	(QS1)
INTR	18	23	$\overline{TEST}$	
CLK	19	22	READY	
GND	20	21	RESET	

8088 CPU

Signal	Pin	Pin	Signal	Max mode
GND	1	40	Vcc	
A14	2	39	A15	
A13	3	38	A16/S3	
A12	4	37	A17/S4	
A11	5	36	A18/S5	
A10	6	35	A19/S6	
A9	7	34	$\overline{SS0}$	
A8	8	33	MN/$\overline{MX}$	
AD7	9	32	$\overline{RD}$	
AD6	10	31	HOLD	($\overline{RQ}$/$\overline{GT0}$)
AD5	11	30	HLDA	($\overline{RQ}$/$\overline{GT1}$)
AD4	12	29	$\overline{WR}$	($\overline{LOCK}$)
AD3	13	28	IO/$\overline{M}$	($\overline{S2}$)
AD2	14	27	DT/$\overline{R}$	($\overline{S1}$)
AD1	15	26	$\overline{DEN}$	($\overline{S0}$)
AD0	16	25	ALE	(QS0)
NMI	17	24	$\overline{INTA}$	(QS1)
INTR	18	23	$\overline{TEST}$	
CLK	19	22	READY	
GND	20	21	RESET	

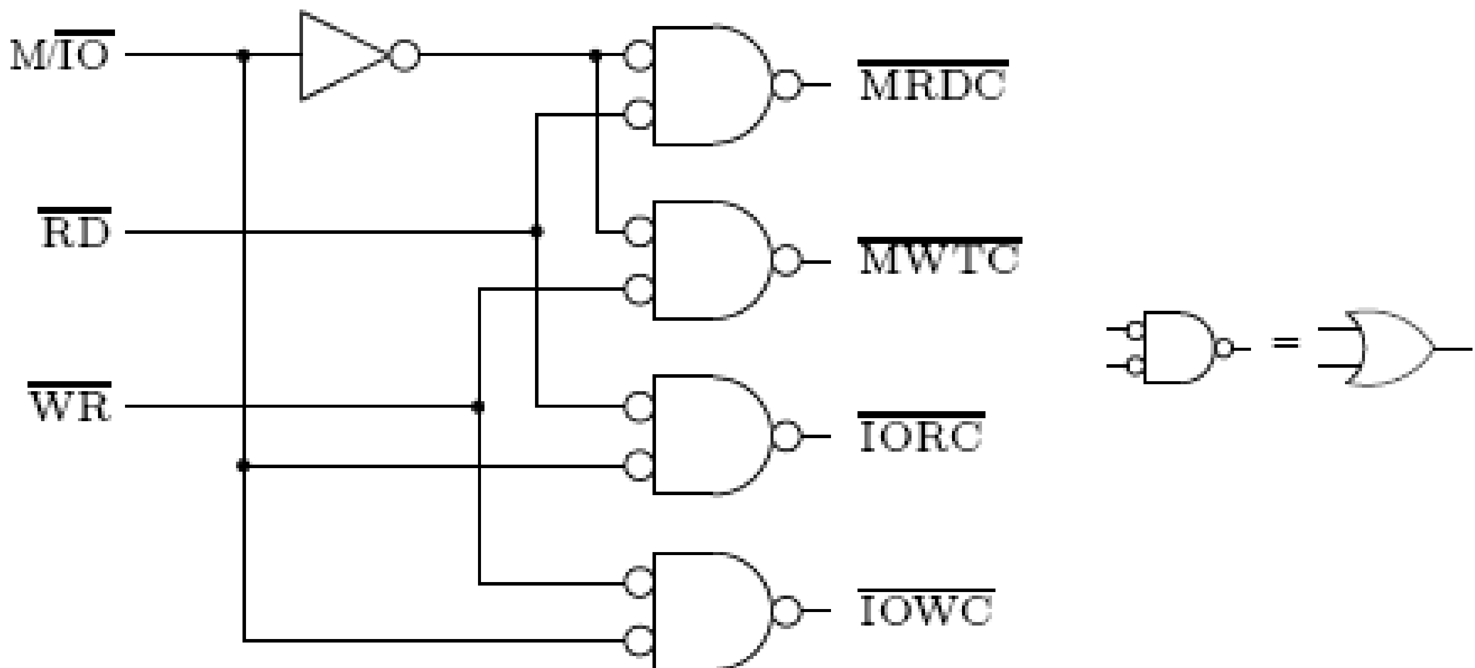
Motorola 68000 μP – Hardware



(a) 68000 CPU signals

Intel Decoding Bus Control Signals

- In *“max mode”* use 8288 bus controller to generate MRDC,MWTC, IORC, IOWC.
- In *“min mode”* (and for other processors) it is sometimes better to decode the available signals.



Motorola 68000 μP – Asynchronous bus control

No Separate IO and Memory!

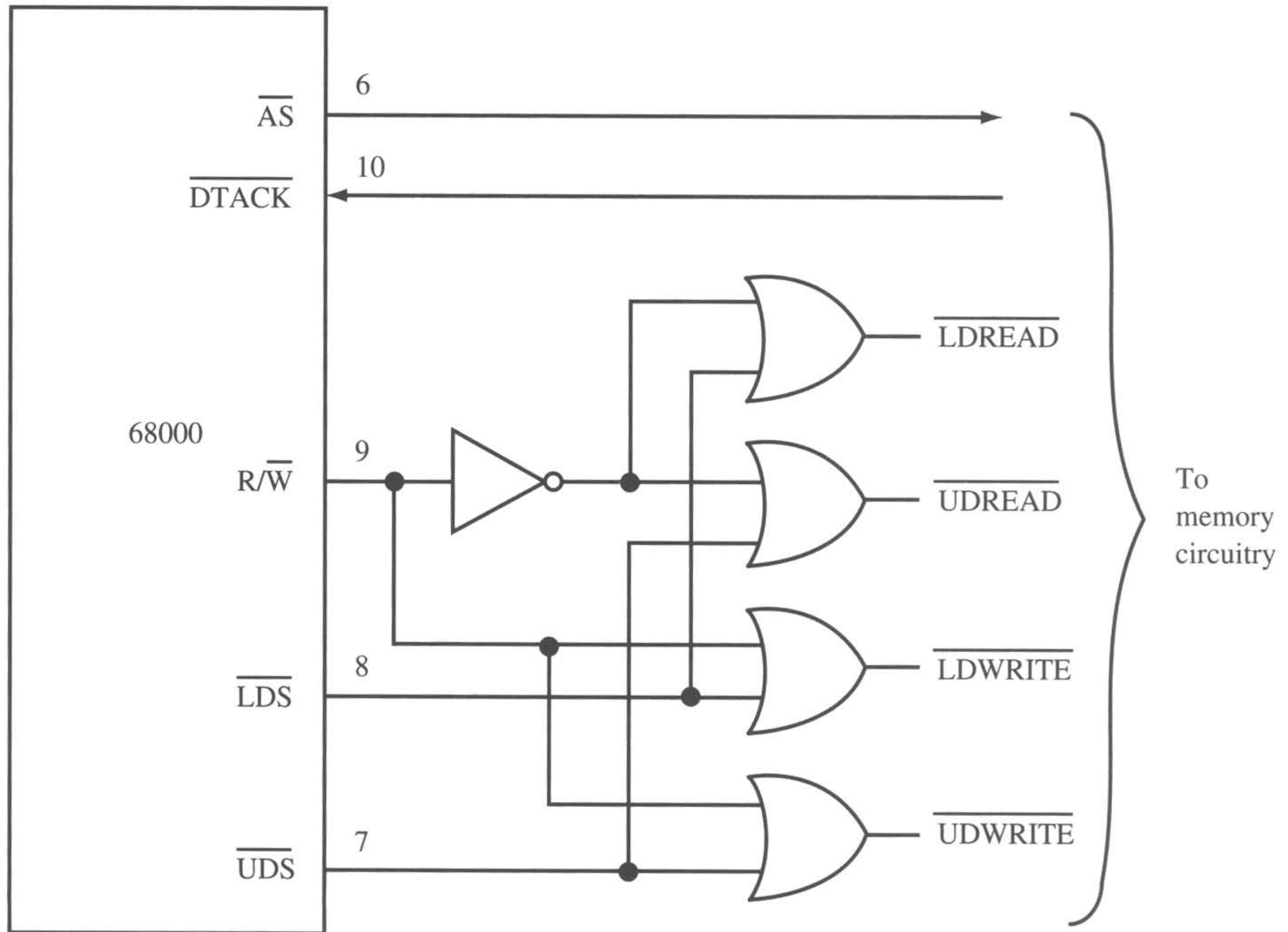
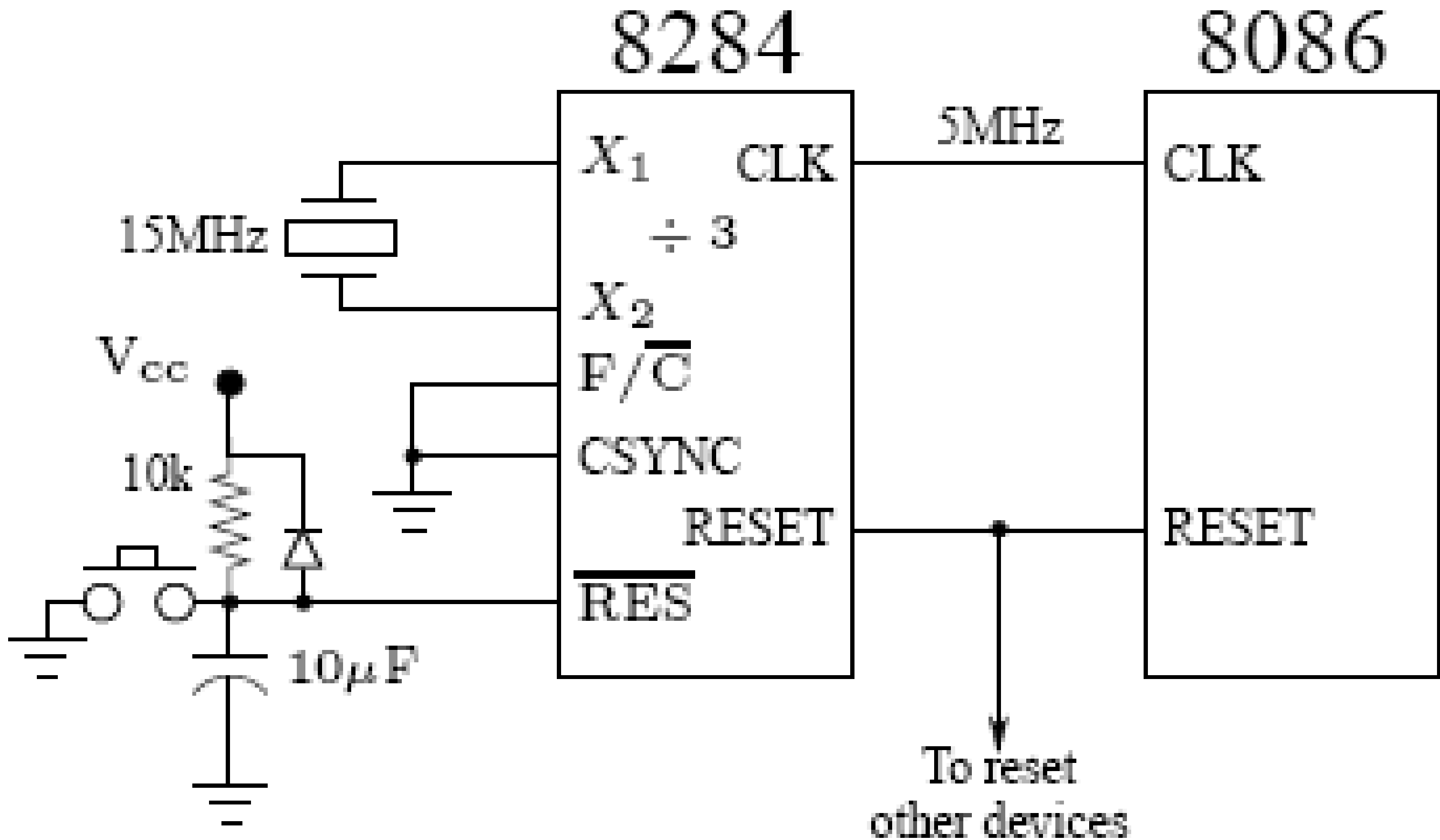


FIGURE 7.12 Decoding memory read/write signals

8284A Clock Generator

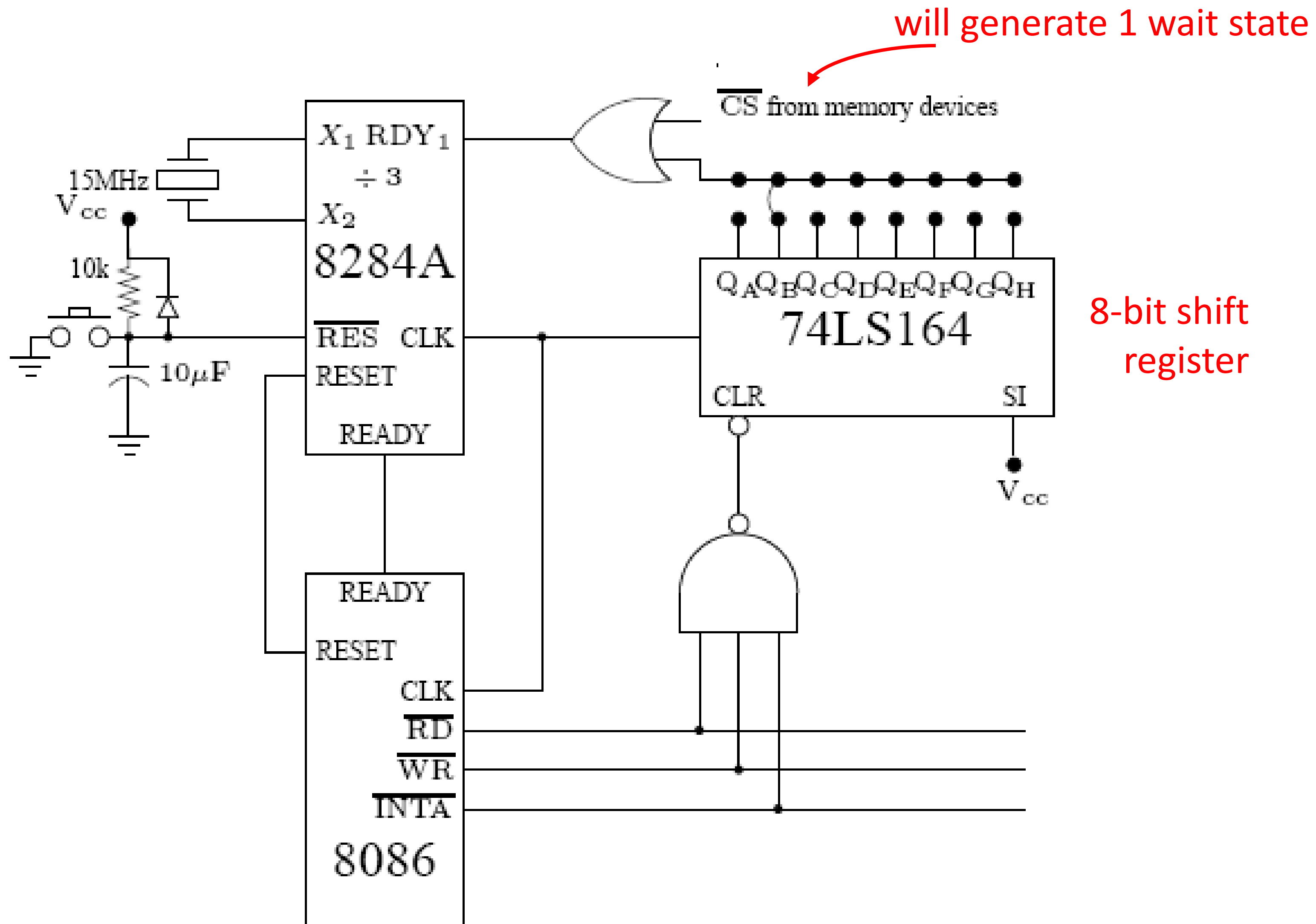


10K pullup? 0.5mA sink. (debouncing!)

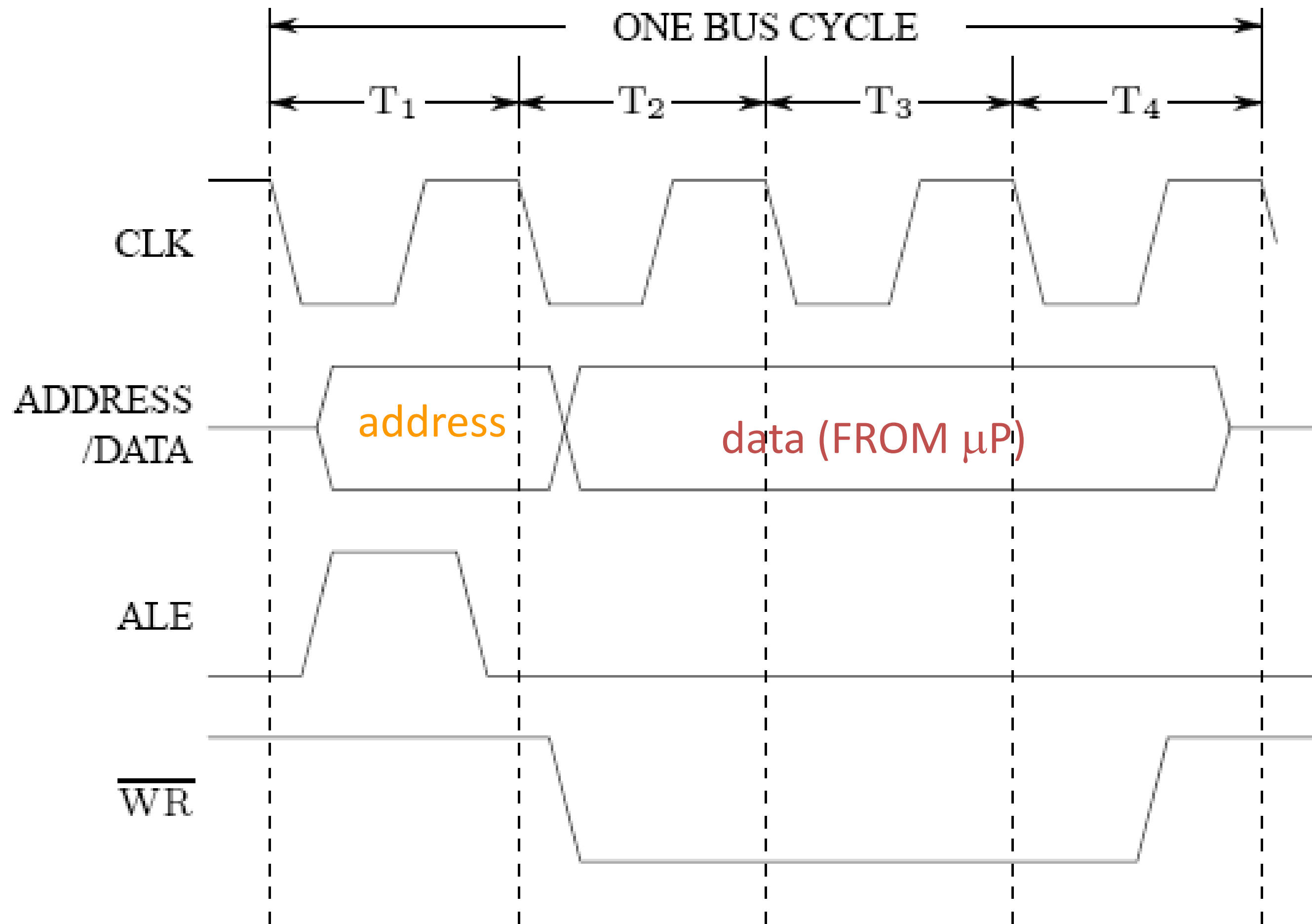
Bus Transfer Synchronization

- Synchronous busses (eg. Motorola 6800/11/12)
  - Transfer times and synchronization are tied to the system clock.
  - No facility for varying bus timing.
  - Clock generators could be used to vary bus speed (for slower memory).
- Semi-synchronous busses
  - provide for “wait states” to be inserted into bus timing (eg. 8086).
  - Allows more flexibility in interfacing to slower memory or I/O.
- Asynchronous busses (eg. Motorola 68000).
  - Requires extra bus signals for bus arbitration.
  - Requires *“acknowlegement”* ($\overline{DTACK}$) signal from devices.
  - Requires bus time-out (watchdog).
  - Easier multiprocessor memory management.

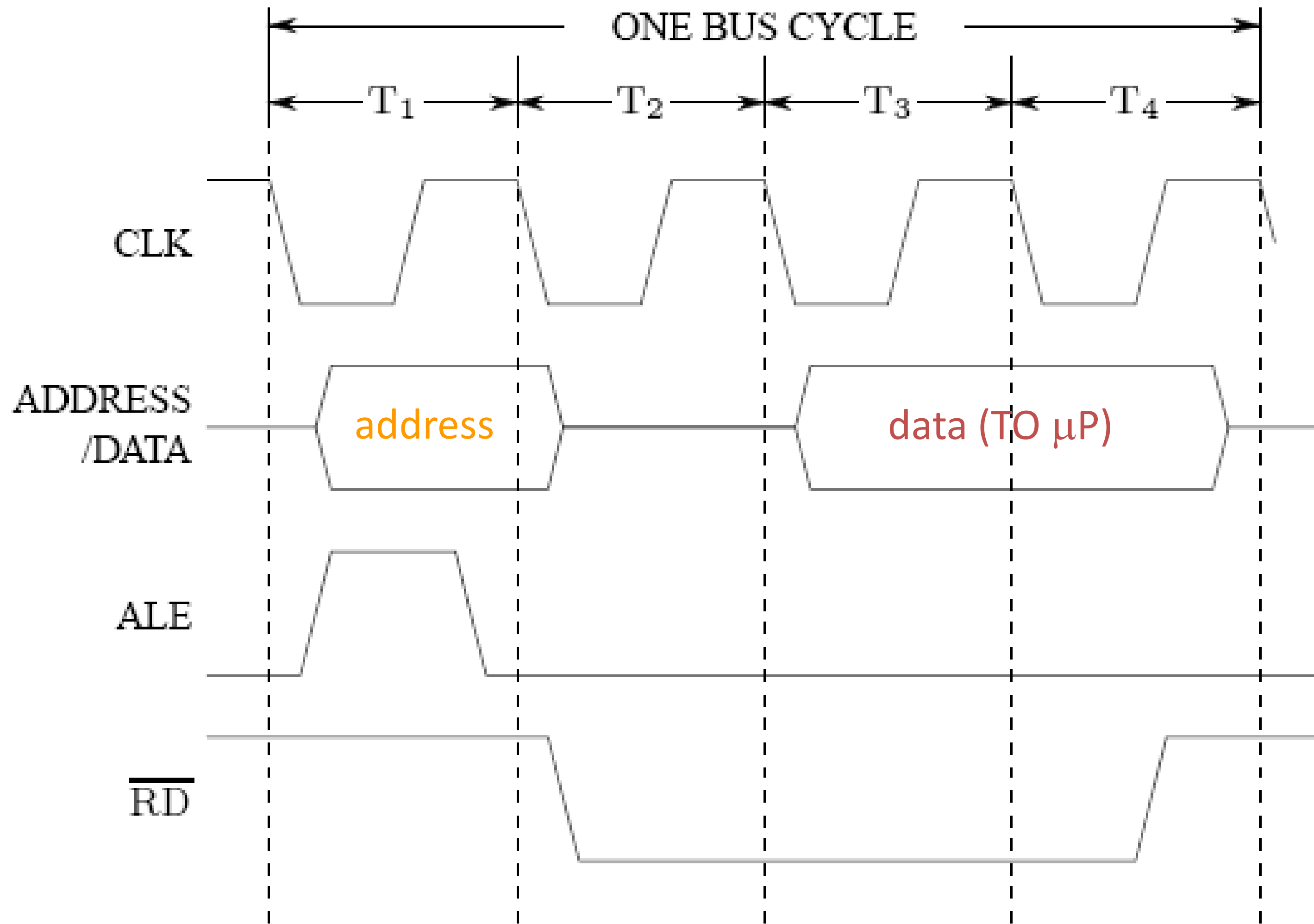
Wait State Generation using 8284A



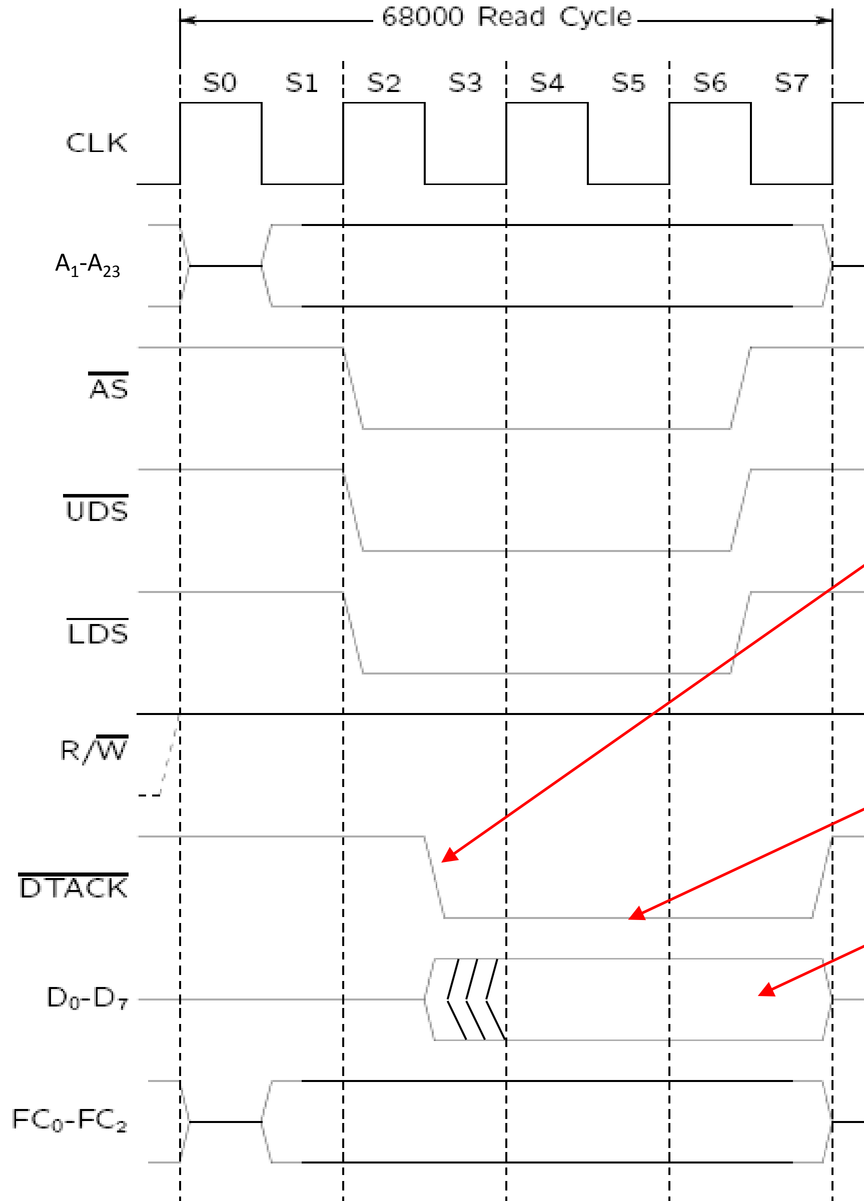
Intel Write Cycle



Intel Read Cycle



Motorola 68000 μP – Read Cycle



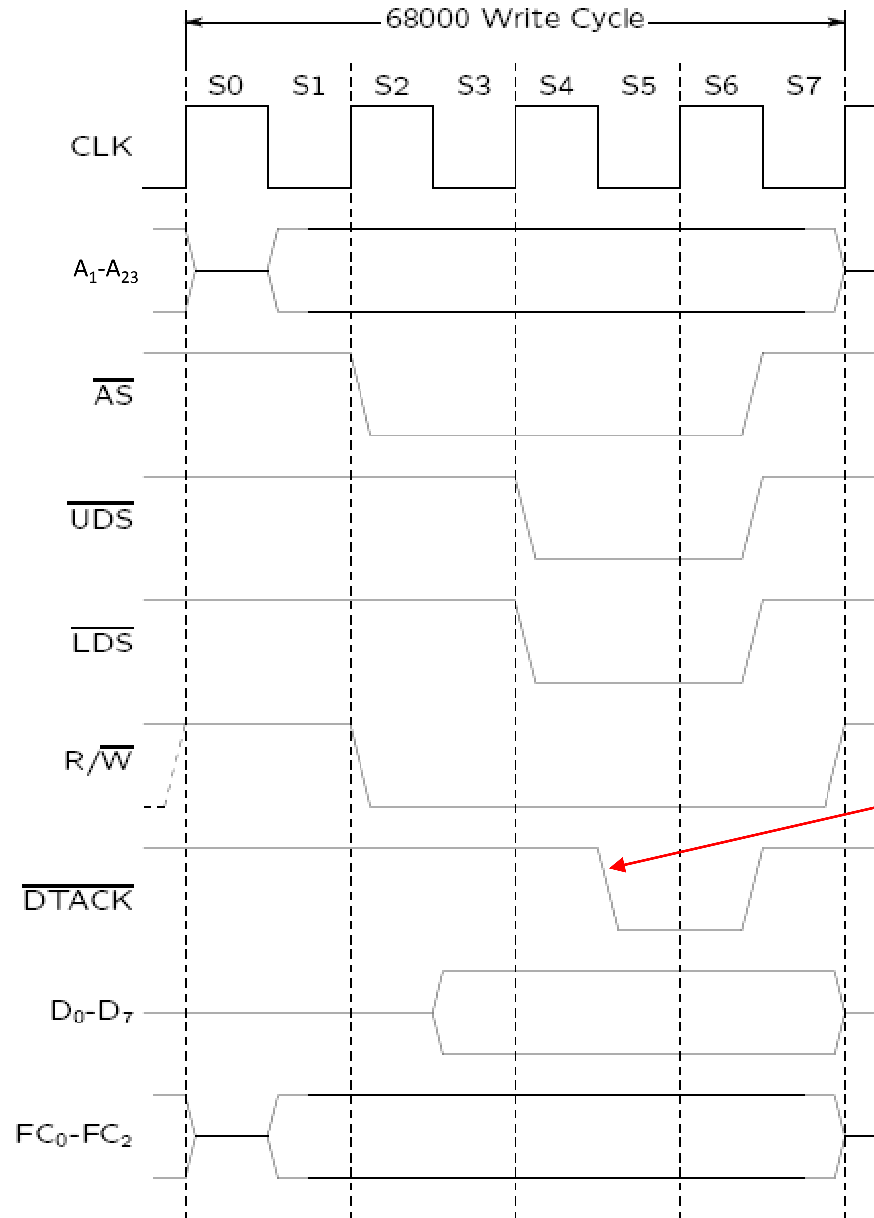
ASYNCHRONOUS!
From memory device.

CHECK $\overline{DTACK}$
(S5)

READ DATA
(end of S6)

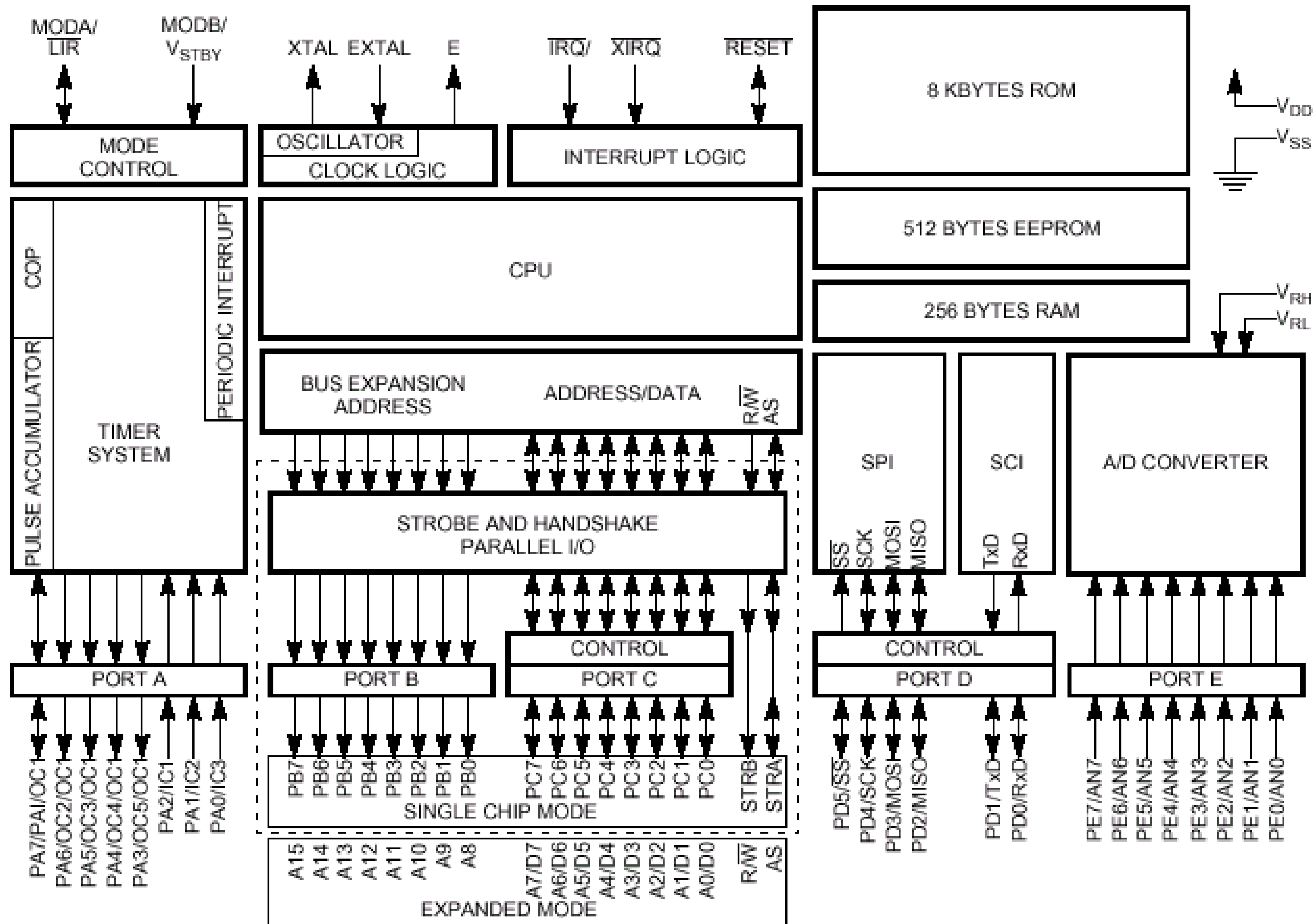
Data may not be
avail when DTACK drops
Only guaranteeing that it
WILL be ready in time
(end of S6)

Motorola 68000 µP – Write Cycle

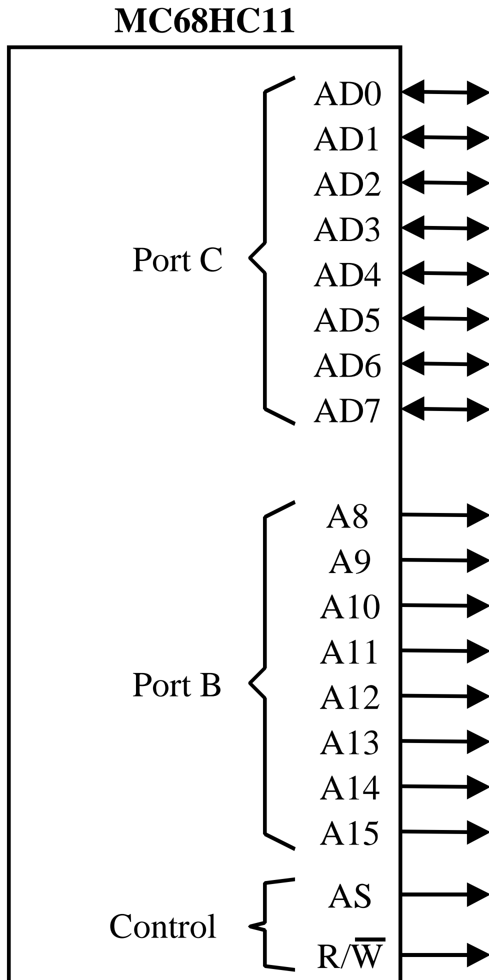


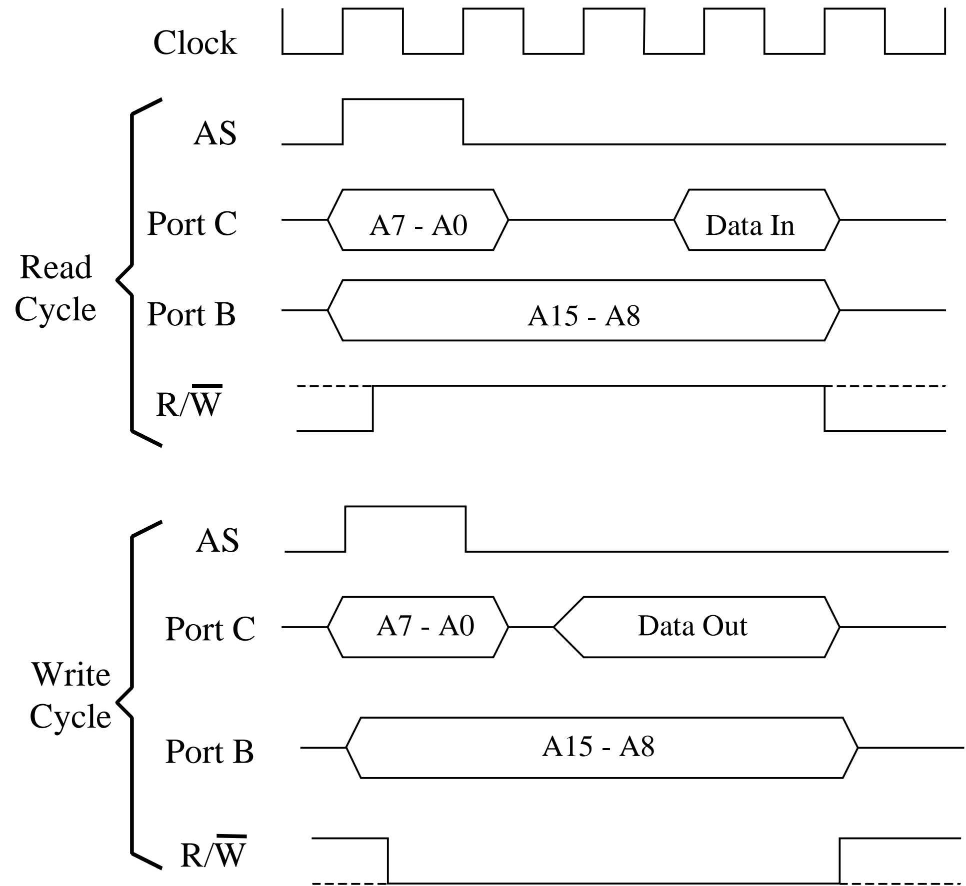
ASYNCHRONOUS! From **memory, can arrive any time before S5 w**ithout causing wait states

Microcontrollers – MC68HC11



Motorola HC11 Expanded Mode Read/Write Bus Cycles





Program Timing

- See Text Appendix B (or handout) for timing
  - Note: the times provided assume that the instructions have already been fetched and are waiting in the queue.
- Max 8086 clock:
  - 5MHz (200ns or 0.2μs per cycle)
  - 2.5MHz (400ns or 0.4μs per cycle)
- instruction times are given in clock cycles.
- Ex: Estimate the time for a 5MHz, zero wait state, 8086 to execute the following code segment:

```
        MOV DI,00FFH
AGAIN:  ADD [1234H+DI],AL
        DEC DI
        JNZ AGAIN
```

Program Timing

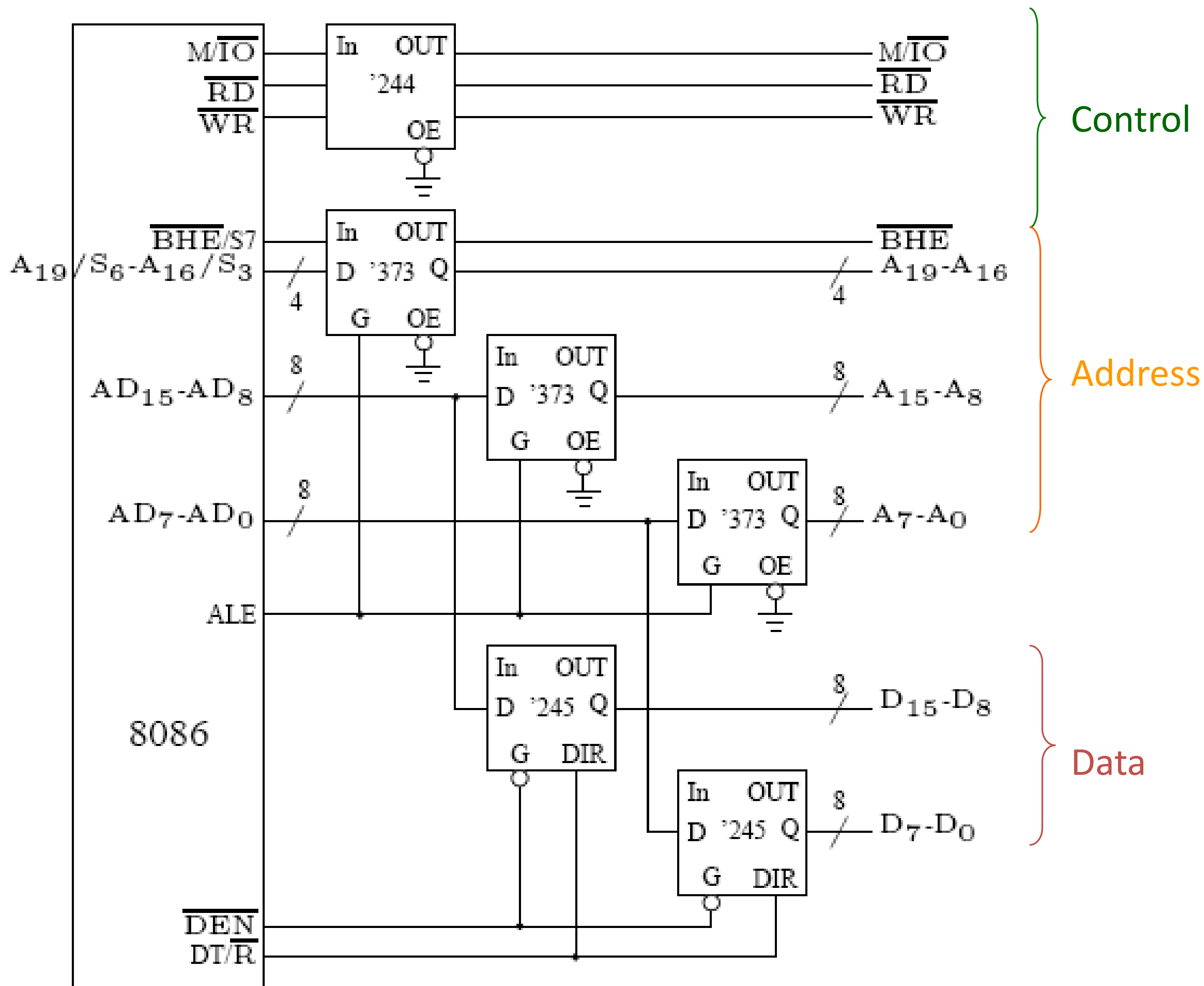
- Note: Loop is executed **254** times with a jump to again, and once with no jump.

Instruction	Add.Mode	T-states	Times	Total
`MOV DI,00FFH`	(reg,imm)	4	1	4
`ADD [1234H+DI],AL`	(mem,reg) EA=9	16+EA=25	255	6375
`DEC DI`	(reg 16)	3	255	765
`JNZ AGAIN`	T	16	254	4064
	F	4	1	4
TOTAL				11212

Total time is: 11212 x 200ns = 2.24ms

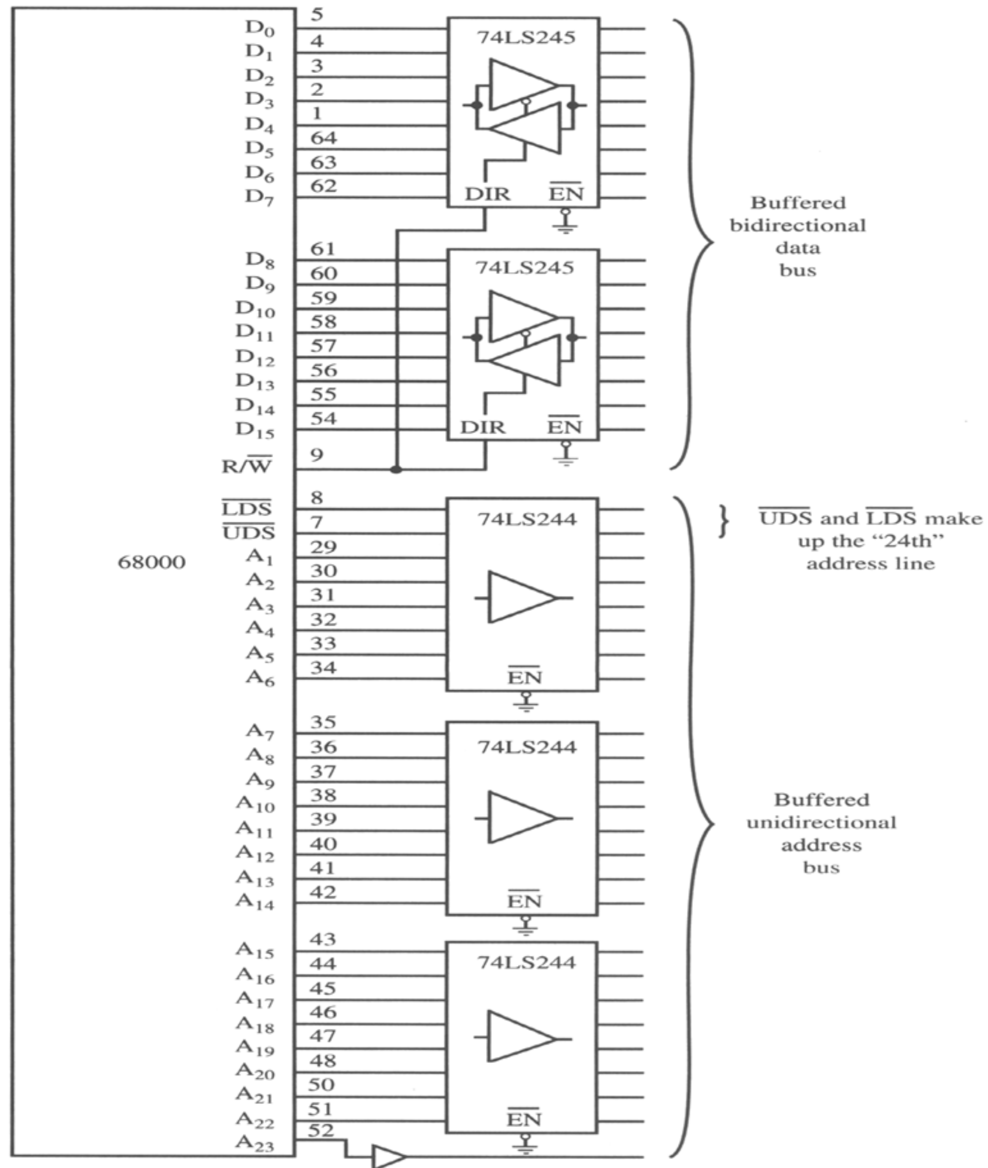
Note: Timing is complicated by 1) Wait States and 2) Unaligned Transfers.

Buffered and Demultiplexed 8086



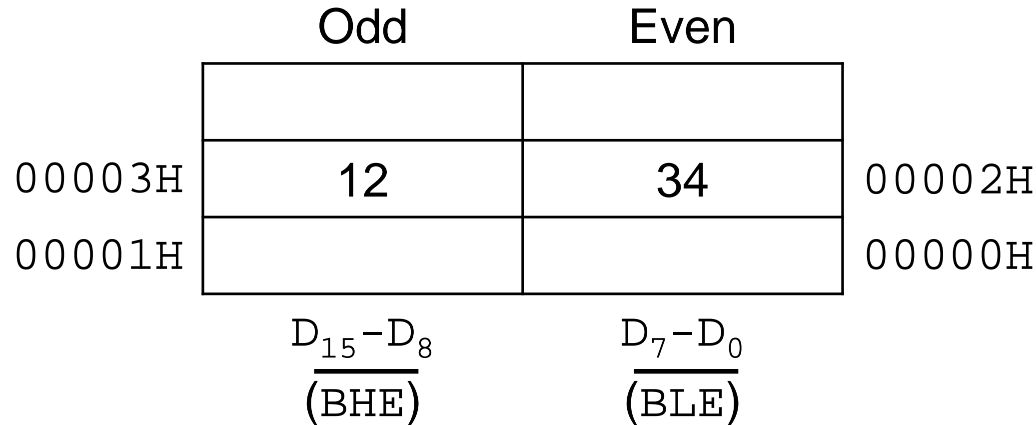
Buffered Motorola 68000, No Demultiplexing!

FIGURE 7.13 Buffering the address and data buses

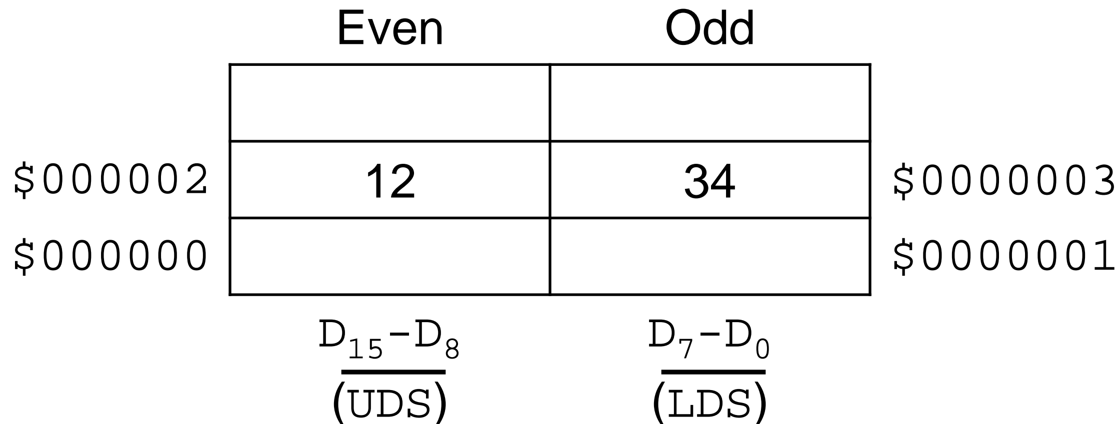


1234 Hex, Little Endian (Intel) vs Big Endian (Motorola)

- 8086 memory drawn with *odd bank (*addresses) on left, and *even* bank on right.

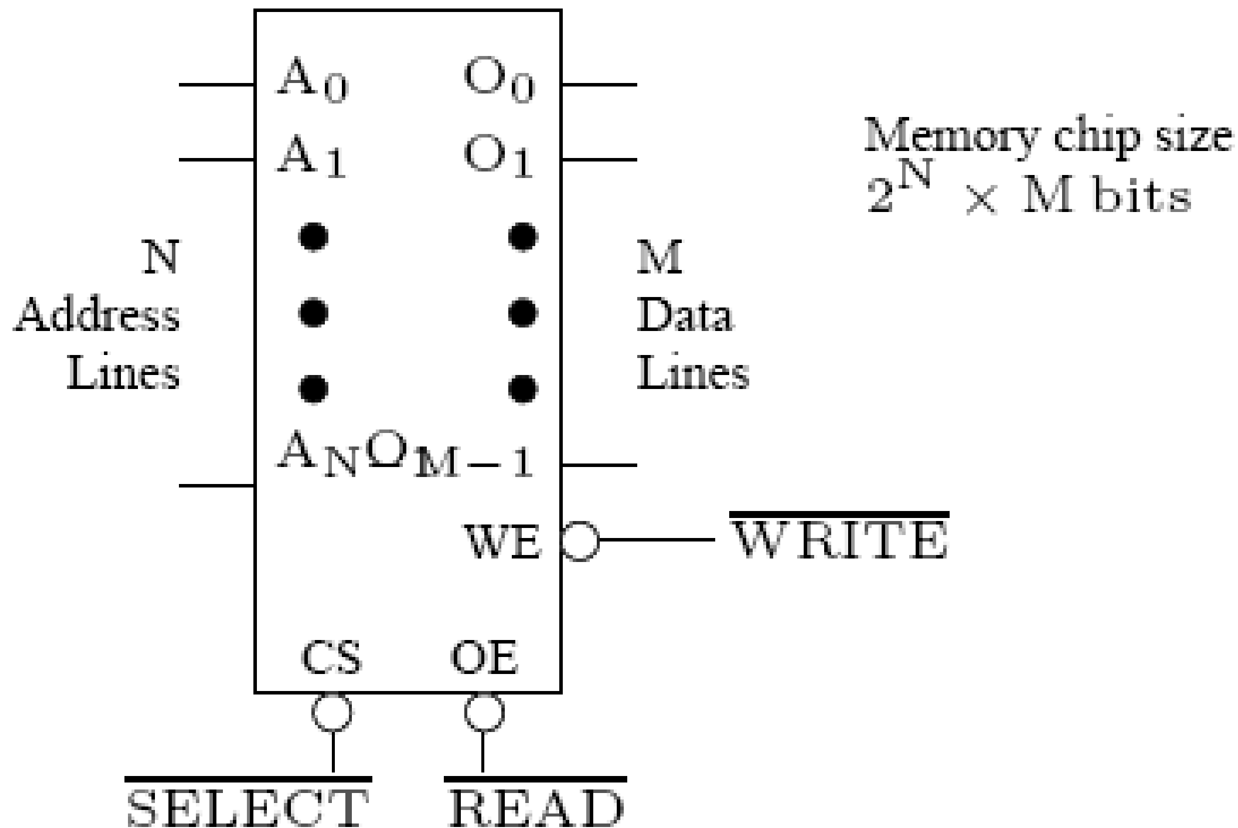


- 68000 memory is usually drawn with *even Addr.* bank on left, and *odd* bank on right.



Memory & I/O Interfacing

- General steps for memory and I/O interfacing
  - Generic memory device:



Memory & I/O Interfacing

- **Steps to success**:

  1) **Architectural questions**:
     - How many chips are required?
     - How many address lines go to each chip?
     - How will chips be organized into banks and which parts of the address bus will be used?

  2) **Determine address range**:
     - Typically problem is to place devices within memory map
     - Determine START, SIZE, LO (=START), HI (=LO+SIZE-1)
     - Determine CONST, SEL, and MEM address lines

  3) Generate overall **chip select** signal ($\overline{\text{MSEL}}$) from CONST portion of address range and M/$\overline{\text{IO}}$

  4) Generate bank-specific **write signals** if required

  5) Complete **interface** design! (often using decoders)
     - Be sure to connect address bus, data bus, and control bus ($\overline{\text{RD}}$, $\overline{\text{WR}}$)

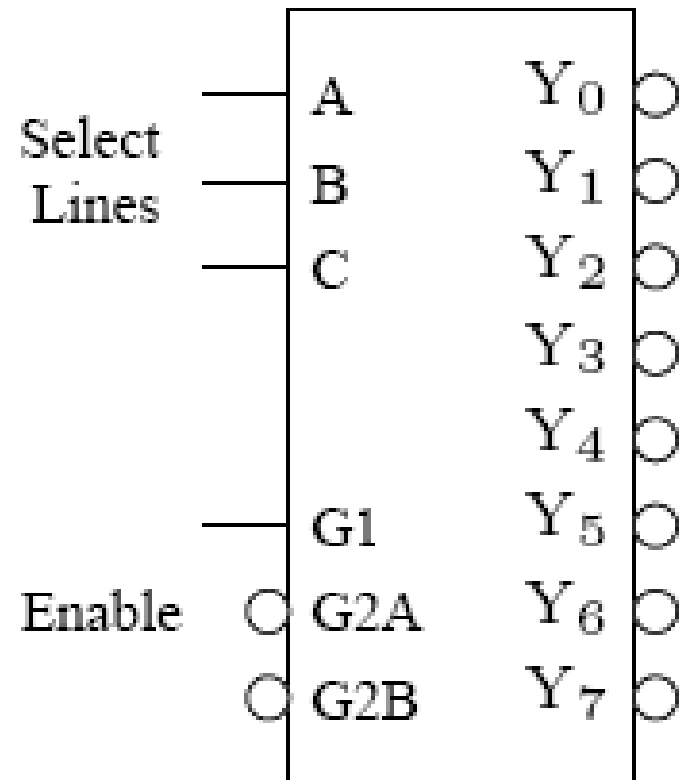
Address Decoding

- **The 74LS138 3-to-8 decoder**

  - Y_n goes low whenever:
    1. G1='1' and G2A = G2B = '0'.
    2. $n = C \times 4 + B \times 2 + A \times 1$

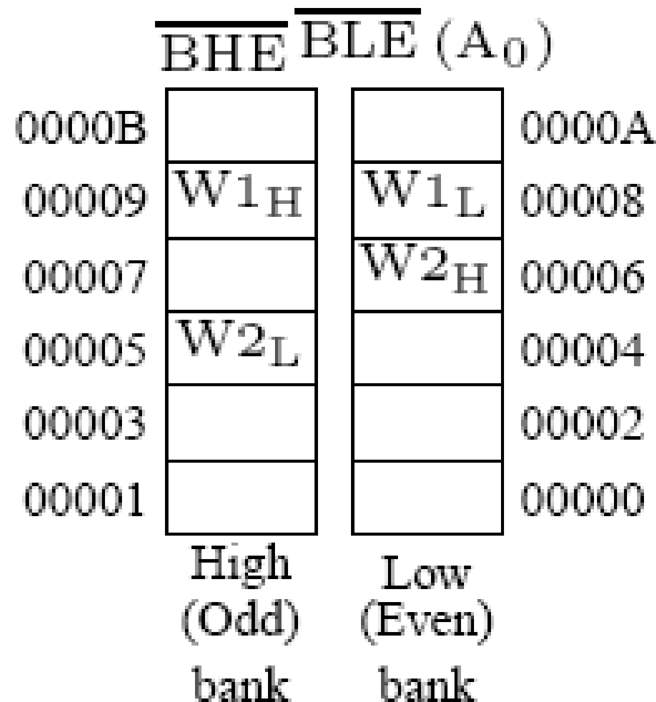
	C	B	A	n
ex:	0	0	1	1
	1	0	1	5

  - Typical time delay is 12nS.



8086 Memory Interface

- The 8086 has a 16-bit data bus.
- Memory is arranged in two 8-bit banks
  - low bank: contains all even addresses.
  - high bank: contains all odd addresses.



8086 Memory Interface

- Aligned/unaligned words
  - W1 is stored on an even (aligned) address.
    - It can be access in a single read cycle.
  - W2 is stored at an odd (unaligned) address.
    - It will require two read cycles (8 T-cycles).

      (a) During first read, $W2_L$ (odd address) will appear on the high byte of the data bus.

      (b) During the second read, $W2_H$ (even address) will appear on the low byte of data bus.
- During a read operation, both banks may (and often are) activated.
- The μP will read 16-bits for read operations, or will only read the correct half of the data bus for byte operations.
  - Note that AL may receive data from the high half of the data bus when reading a byte from an odd address!

Memory Interface

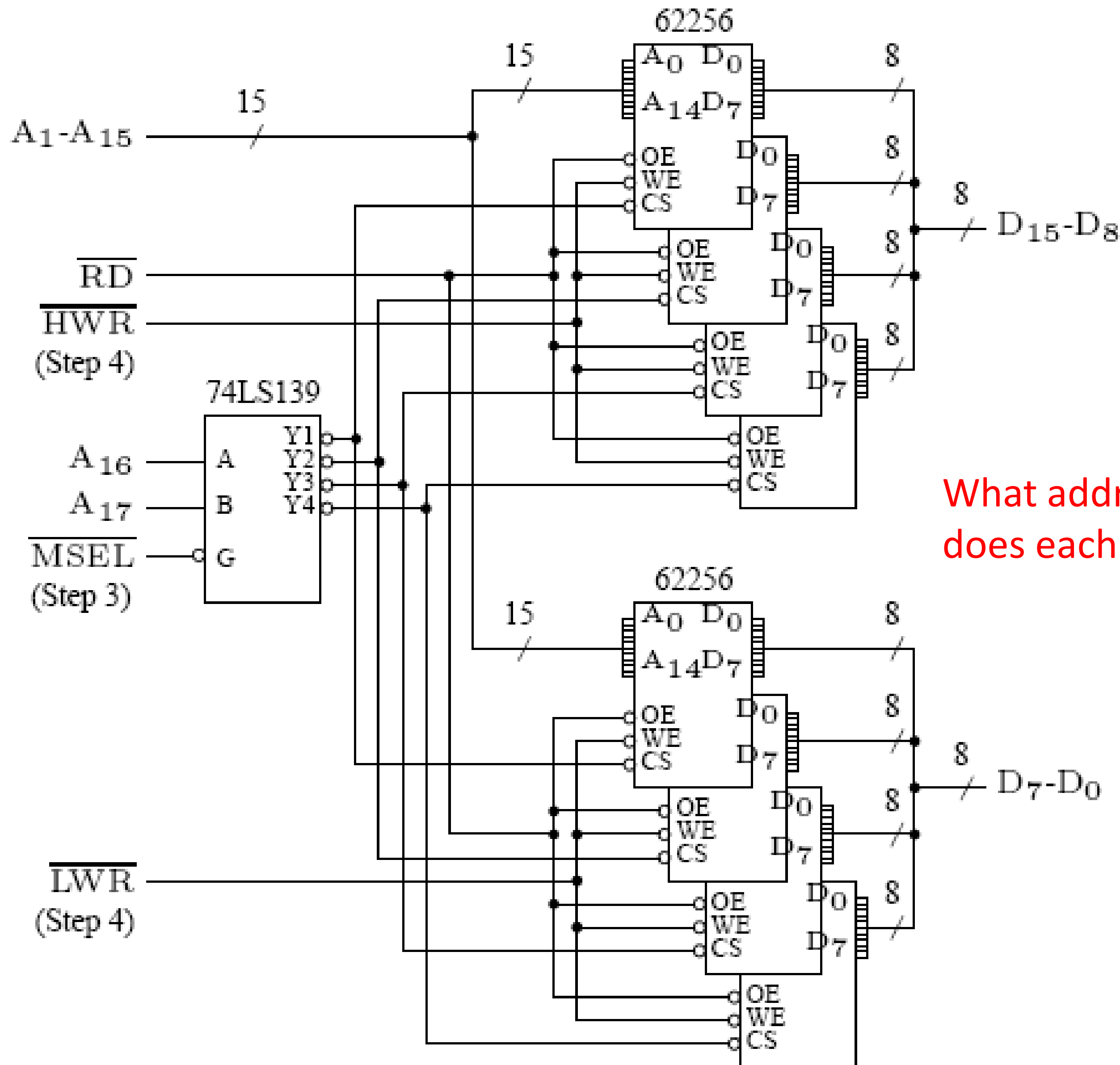
- Write cycles must activate the correct bank(s) based on $\overline{BHE}$ and $\overline{BLE}$ (A_0).
- $\overline{BHE}$ is supplied by μP (multiplexed with S_7)
- A_0 is used as $\overline{BLE}$
  - i.e. A_0=0 for an even address and A_0=1 for an odd address
  - (A_0 is not even a pin on the '386 and up)

$\overline{BHE}$	$\overline{BLE}$	Function
0	0	Both banks (16 bits)
0	1	High bank (8 bits)
1	0	Low bank (8 bits)
1	1	No banks enabled

16-bit Intel Memory Interfacing Example

- Design a memory interface for the 8086 which will provide 256k bytes of SRAM, organized as 128k x 16bits, starting at address ?????H and using 62256 SRAM chips (32k x 8bit).
  - Assume that 8086 address, data, status, and control busses are already demultiplexed and buffered.

1



What address range does each chip respond to?

Motorola 68000 μP – Memory Interfacing

- Almost identical to the 8086 except:
  1. Switch even and odd banks
  2. Must generate $\overline{\text{DTACK}}$
  3. Must use AS, R/$\overline{\text{W}}$, $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ for control.
- During a byte-read operation, the μP will select the correct half of the data bus depending on whether it's an even or odd address (similar to 8086).
- Separate write strobes are required for even and odd banks so that data is not written to the wrong memory bank.

16-bit Motorola Memory Interfacing Example

- Design a memory interface for the 68000 which will provide 256k bytes of SRAM, organized as 128k x 16bits, starting at address $?????? and using 62256 SRAM chips (32k x 8bit).
  - Assume that the 68000 address, data, status, and control busses are already buffered.

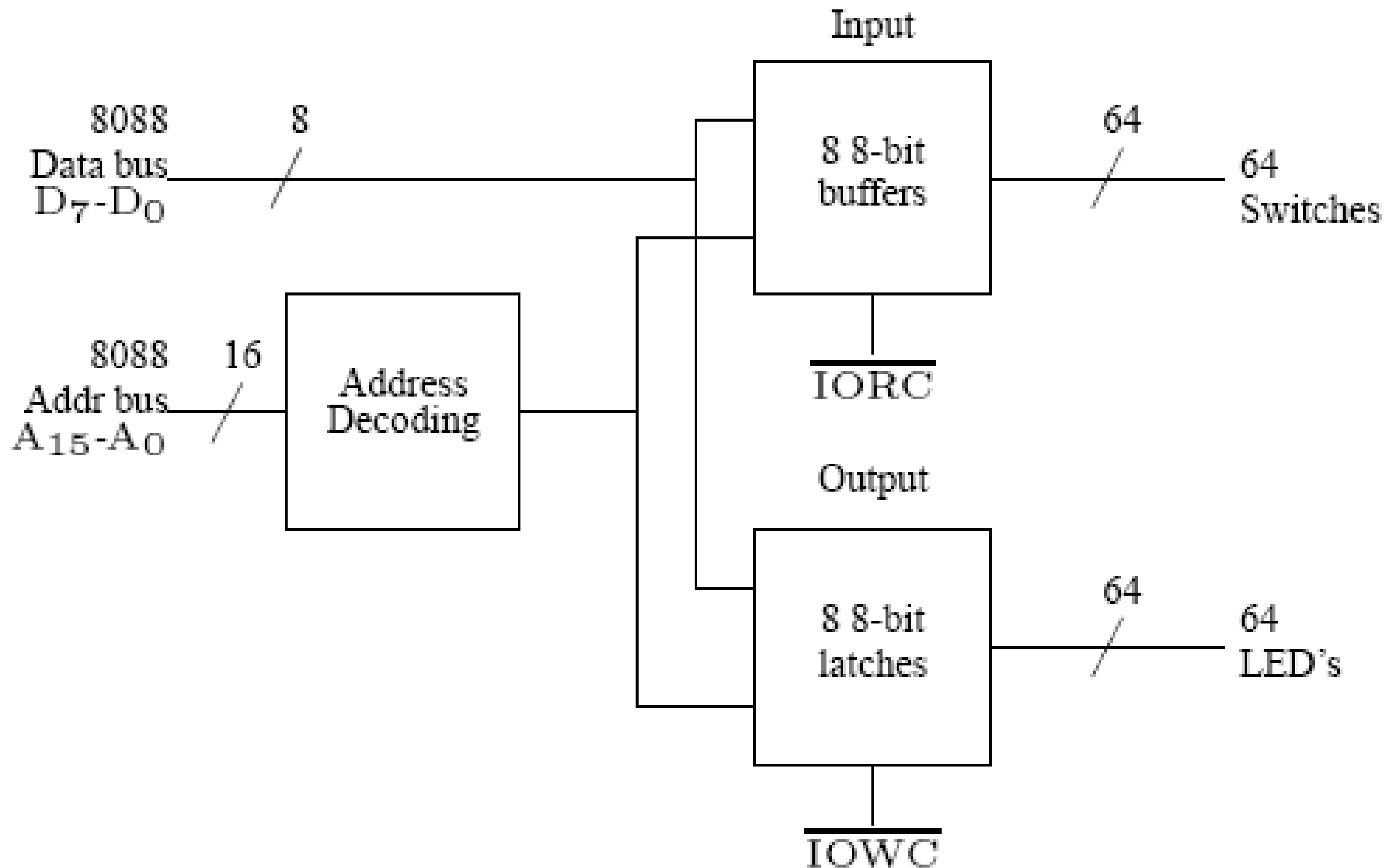
Intel I/O Mapping Options

- Two methods are available for Intel:
  1. I/O mapped I/O (isolated I/O, Intel)
     - I/O Ports are isolated from memory in a separate I/O address space.
     - Memory can be expanded to full size
     - Data transfer from/to I/O is restricted to IN and OUT instructions.
     - Separate control signals using $M/\overline{IO}$, $\overline{WR}$, $\overline{RD}$ enable I/O ports.
     - Intel-based PC's use isolated I/O

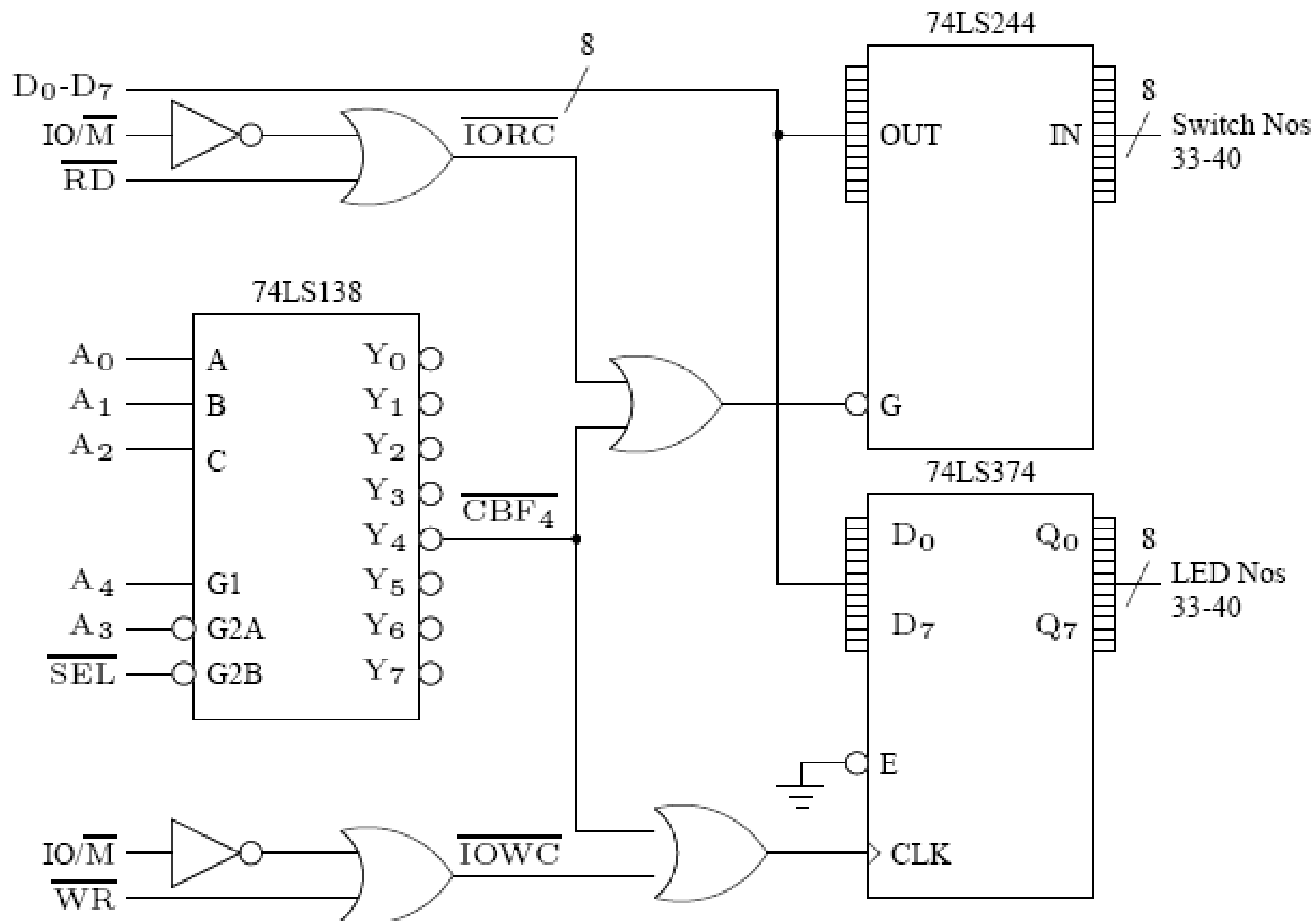
I/O Mapping Options

2. Memory Mapped I/O (Intel and Motorola)
   - I/O device is treated as a memory location.
   - Any memory transfer instruction can used to access the device.
   - Reduces amount of system memory available to applications.
   - Reserves fixed portion(s) of the memory map for I/O.
   - 6800, 68000 uses memory-mapped I/O.

I/O Interfacing



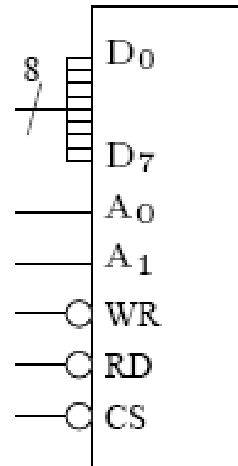
I/O Interfacing



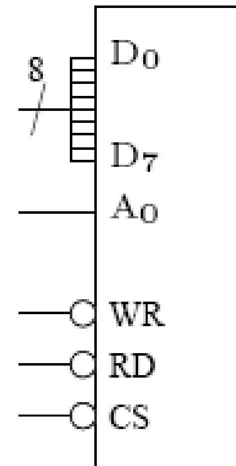
[a]Only one input and one output bank at address CBF4H is shown.

Peripheral Device Interfacing for I/O

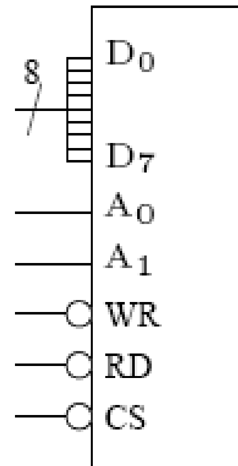
8255 (PPI)



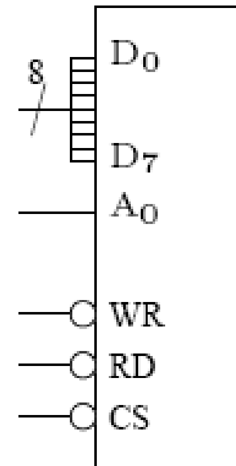
8259 (PIC)



8253 (PIT)

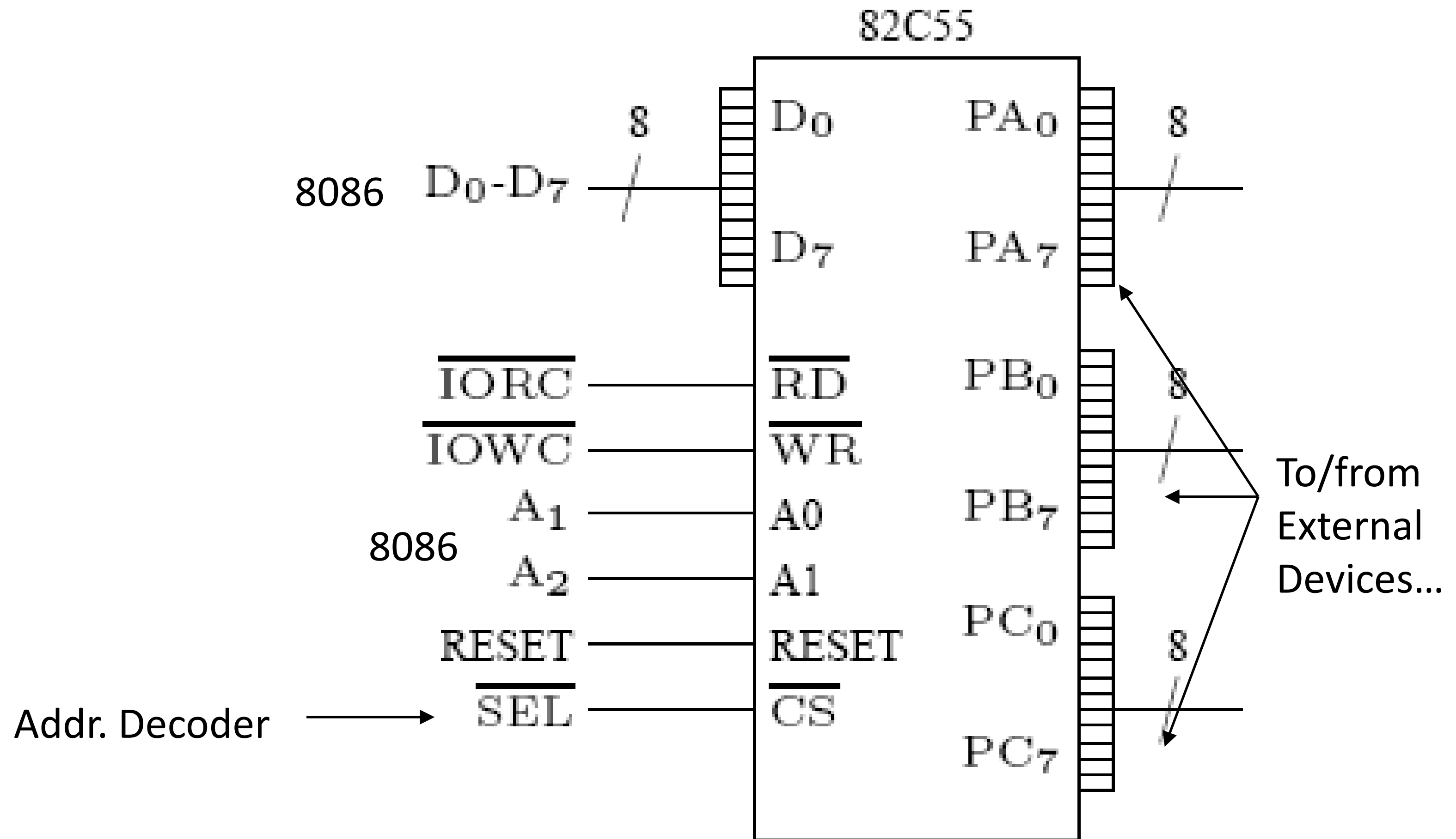


8279



I/O Interfacing

Ex: 8255 Interface:



Motorola 68000 μP – I/O Interfacing

- All I/O is memory-mapped.
- Decoding is the same as for memory.
- One still must generate $\overline{\text{DTACK}}$.

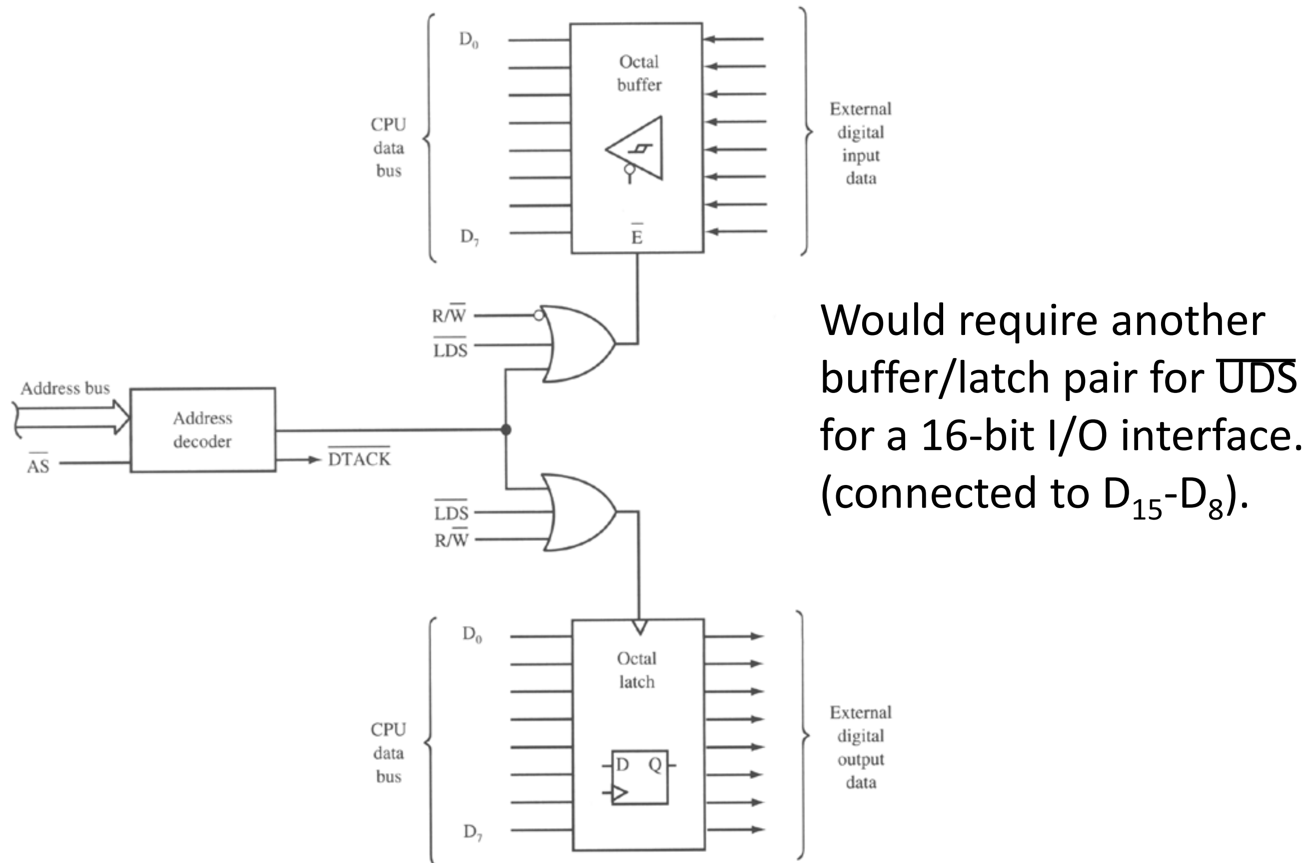
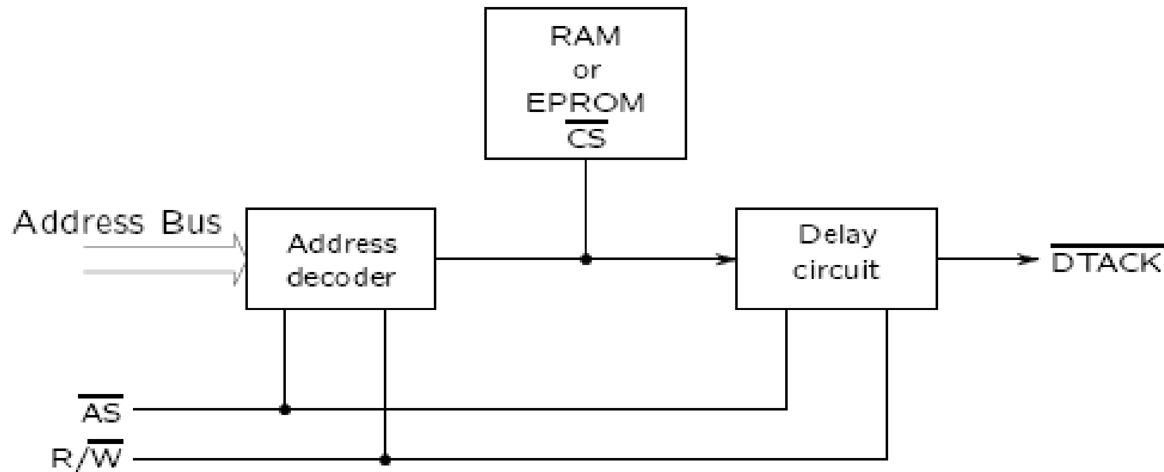


FIGURE 9.1 Memory-mapped I/O circuitry

Motorola 68000 DTACK

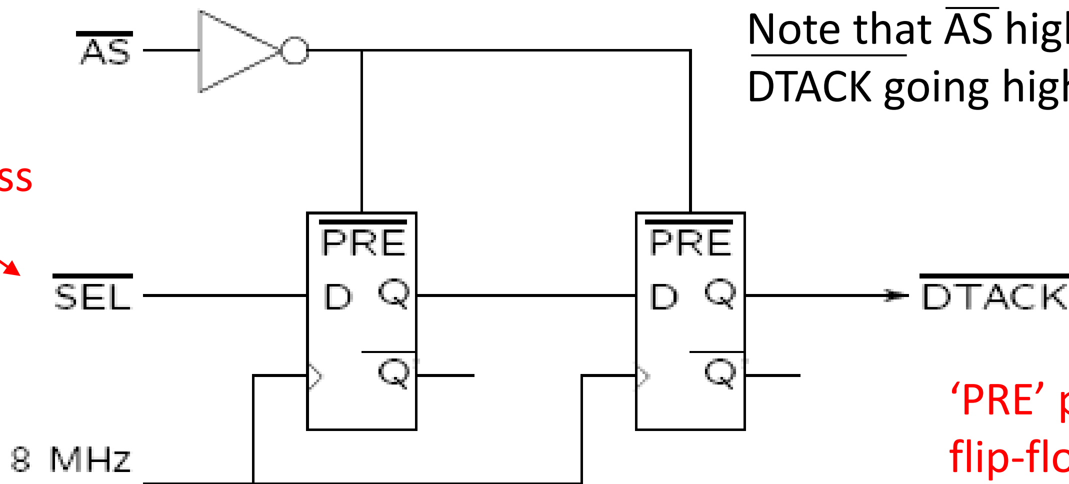
- Block diagram of $\overline{\text{DTACK}}$ circuit:



- $\overline{\text{DTACK}}$ delay generator:

Note that $\overline{\text{AS}}$ high leads to $\overline{\text{DTACK}}$ going high immediately

'Device select' signal from address decoder.



'PRE' presets the flip-flop (sets it to 1)

Intel Interrupt Response Sequence

- Each time the μP completes execution of an instruction, it will check the status of **`NMI`** and **`INTR`**.
- if either is active, or if the next instruction is **`INTO`**, **`INT`** ***n***, or **`BOUND`**, then:

    1. Push flag register onto stack.
    2. Clear `IF` and `TF` (interrupt enable and trap flags). Interrupts are now disabled.
    3. Push `CS` then `IP` on stack.
    4. Fetch the *interrupt vector* (discussed shortly)

- The final statement of an interrupt service route (ISR) is **`IRET`** – it pops `IP`, `CS` and `Flags`.

Intel Interrupt Vector Table

- Located in first 1K of memory (00000-003FF).
- Contains 256, 4-byte interrupt vectors.
- Each interrupt vector contains the address (segment and offset) of the service routine.
- Each entry in the vector table is represented by an integer between 0 and 255, called the *interrupt type*.

Intel Interrupt Vector Table

Address	Contents
	Type 32 — 255 User interrupt vectors
080H	Type 14 — 31 Reserved
	Type 16 Coprocessor error
040H	Type 15 Unassigned
03CH	Type 14 Page fault
038H	Type 13 General protection
034H	Type 12 Stack segment overrun
030H	Type 11 Segment not present
02CH	Type 10 Invalid task state segment
028H	Type 9 Coprocessor segment overrun
024H	Type 8 Double fault
020H	Type 7 Coprocessor not available
01CH	Type 6 Undefined opcode
018H	Type 5 BOUND
014H	Type 4 Overflow (INTO)
010H	Type 3 1-byte breakpoint
00CH	Type 2 NMI pin
008H	Type 1 Single-step
004H	Type 0 Divide error
000H	

(a)

Any interrupt vector

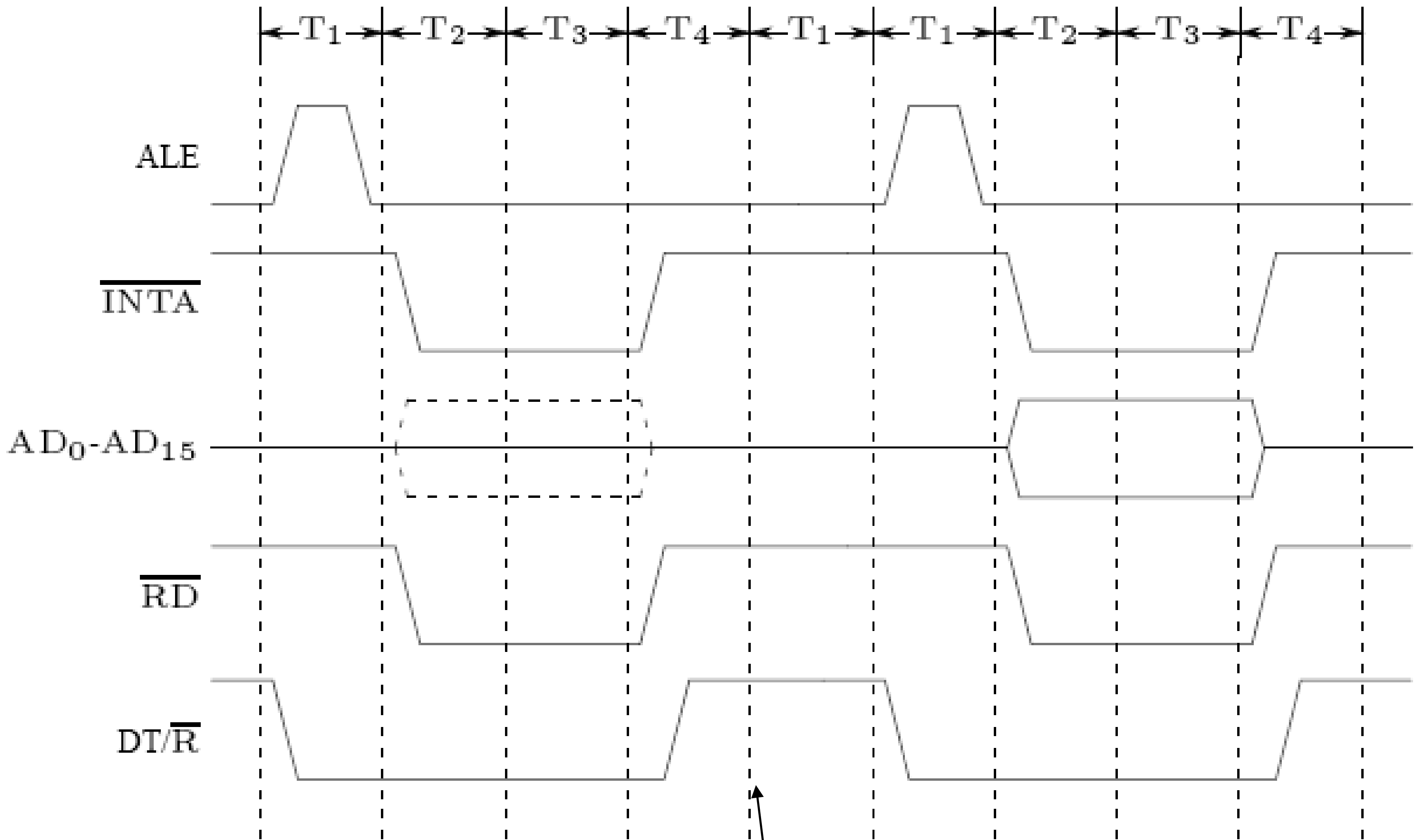
3	Segment (high)
2	Segment (low)
1	Offset (high)
0	Offset (low)

(b)

Intel response to hardware interrupts

- The response to an `INTR` is *two* `INTA` bus cycles separated by two idle clock cycles.
- No address is provided by the 8086, but `ALE` is generated which will load the address latches with unknown data.
- First `INTA` cycle signals devices to prepare to present the **TYPE** number on the next `INTA` (CPU does not capture info on the first `INTA`).
- During the second `INTA`, the device causing the interrupt places a byte on `D7–D0` which represents the interrupt TYPE.

Intel response to hardware interrupts



Note: One more idle clock cycle here

Motorola 68000 μP – Exceptions (Interrupts)

- 68000 Hardware Interrupts
  - Seven levels of external interrupts depending on $\overline{\texttt{IPL2}}$, $\overline{\texttt{IPL1}}$, and $\overline{\texttt{IPL0}}$.
  - Level 0, all $\overline{\texttt{IPLs}}$ = 1, no interrupt.
  - Level 7, all $\overline{\texttt{IPLs}}$ = 0, highest priority (non-maskable).
  - Interrupt priority mask (bits 8, 9, and 10 of `SR`) is set to disable lower priority interrupts.

Example circuit to generate a level-7 interrupt using a single push-button.

We can develop more complex circuits to generate multiple interrupt levels depending on the source of the interrupt request.

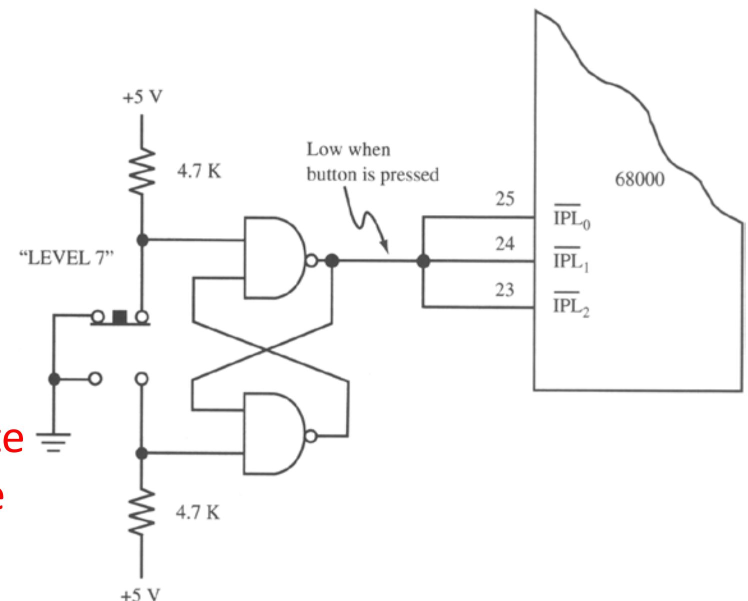


FIGURE 7.10 Generating a level-7 interrupt with a pushbutton

Motorola 68000 μP – Exceptions (Interrupts)

- Interrupt Acknowledge Cycle
  - (asynchronous, hardware interrupt requests)

  1. Device and interrupt logic set $\overline{\mathtt{IPL2}}$, $\overline{\mathtt{IPL1}}$ and $\overline{\mathtt{IPL0}}$.
  2. μP completes current instruction.
  3. μP enters interrupt acknowledge cycle.

     (a) FC2, FC1, FC0 = 111.

     (b) $\overline{AS}$ = 0, $\overline{LDS}$ = 0, R/$\overline{W}$ = 1.

     A_3, A_2, A_1 = requested interrupt level.

Motorola 68000 μP – Exceptions (Interrupts)

- Interrupt Acknowledge Cycle con't

  4. External logic may do one of two things:

     (a) Supply a vector number.

     – Place 8-bit vector number of D_7-D_0.

     – pull $\overline{DTACK}$ low.

     – μP will read D_7-D_0.

     (b) Request an "auto-vector".

     – Pull $\overline{VPA}$ low. Leave $\overline{DTACK}$ high.

     – μP generates its own vector based on interrupt level first supplied to $\overline{IPL}$ inputs.

     – autovectors point to locations $064 through $07f in vector table.

  – Autovectors should be used whenever 7 or less interrupt types are needed.

  5. Proceed with exception handling steps from slide 42

Motorola 68000 μP – Exceptions (Interrupts)

The response to an interrupt:

1. Resolve priorities from external interrupt request, present appropriate 3-bit code on $IPL_{2\text{-}0}$
2. Monitor $FC_{2\text{-}0}$ for intr acknowledge cycle.
   - AS=0, R/W=1, LDS=0
   - $A_{3\text{-}1}$ = requested interrupt level
3. Either:
   3a) provide vector number on $D_{7\text{-}0}$ and pull DTACK low,
   OR
   3b) request autovector by pulling VPA low.

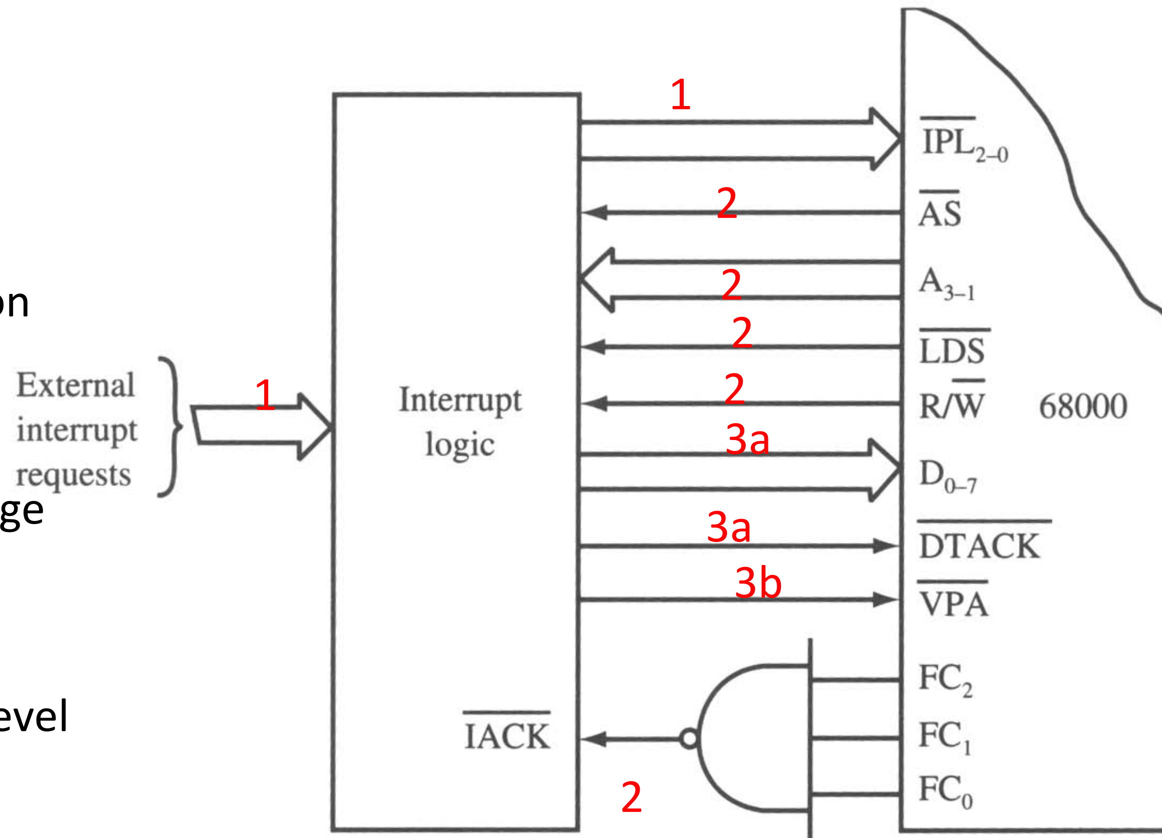


FIGURE 4.9 External interrupt circuitry block diagram