Microprocessor Systems

97.461

Maitham Shams

Course Slide Presentations

Department of Electronics
Carleton University

History of Computation

- Mechanical Age: B.C. to 1800s
  - 500 B.C. Babylonians invented abacus, first mechanical calculator
  - 1642 Blaise Pascal invented calculator using wheels and gears
  - 1823 Charles Babbage created Analytical Engine capable of storing data using punch cards
- Electrical Age: 1800s to 1970s
  - Triggered by advent of electric motor (conceived by Faraday)
  - Motor driven adding machines based on Pascal's idea

- 1896 Hollerith formed Tabulating Machine Company (Today's IBM)
- 1946 ENIAC (Electronics Numerical Integrator and Calculator First general purpose programmable electronic machine Used 17000 vacuum tubes, 500 miles of wires, weighed 30 tons. Performed 100K operations/second, programmed by rewiring)

- Integrated Circuits Age: 1960s to present
  - Triggered by development of transistor at Bell Labs, 1948
  - 1958 IC technology invented by Jack Kibly of Texas Instruments
  - 1971 World's first microprocessor, Intel 4004, 4-bit bus 4K 4-bit(nibble) memory, 50 KIPs, 2300 transistors, 10 µm technology
  - 1972 first 8-bit µP, Intel 8008, 16K bytes, 50 KIPs

- 1973 Intel 808, 64K bytes, 500 KIPS, 6000 transistors, 6 µm followed by other 8-bit µPs lke Motorola MC6800 (1974) and Z-8
- 1978 Intel 8086, 16-bit µP, 1M bytes, 2.5 MIPs

  Used 4-bytes instruction cache to speed up execution time

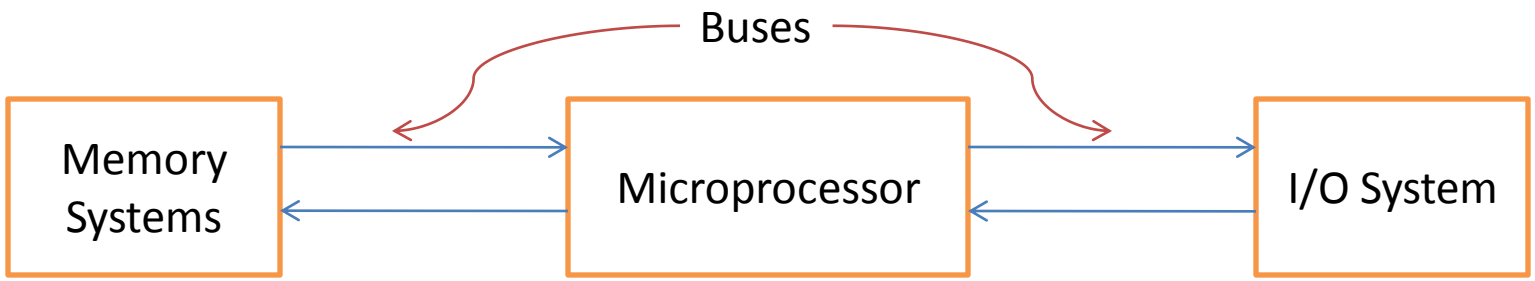
  Base for 80286 µP, also 16-bit with 16M bytes
- 1986 Intel 80386, 32-bit µP, 32-bit data and address busses

  4G bytes, 16 to 33 MHz, 275000 transistors, 1 µm
- 1989 Intel 80486, like 80386 with numeric co-processor. 4G bytes + 8Kb cache, 25 to 50 MHz, 1.2M transistors, 1 and 0.8 µm
- Advancement continues with Intel, AMD, Motorola, and other µPs

Reasons Behind µP Technology

- Speed
  - Graphics, Numerical Analysis, CAD, and Signal Processing applications
- Convenience
  - Large memory, smaller size, and lower weight
- Power Dissipation
  - Portable computers and wireless services
- Reliability
  - Noise tolerance in adverse environments and temperatures
- Cost
  - Get more done for the money

µP BASED Computer Systems



Dynamic RAM (DRAM)
Static RAM (SRAM)
Cache
Read-Only (ROM)
Flash Memory
EEPROM

8086
8088
80186
80286
80386
80486
Pentium
Pentium Pro
Pentium II

Printer
Hard disk drive
Mouse
CD-ROM Drive
Keyboard
Monitor
Scanner

Memory

- Transient Program Area

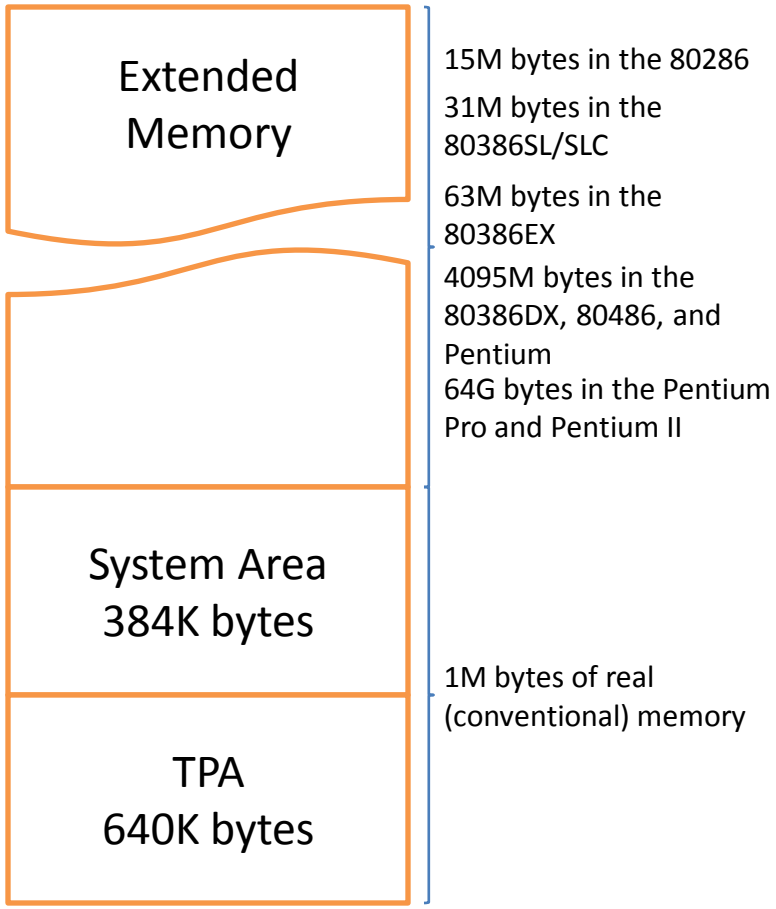
(TPA) 640Kb

- System Area

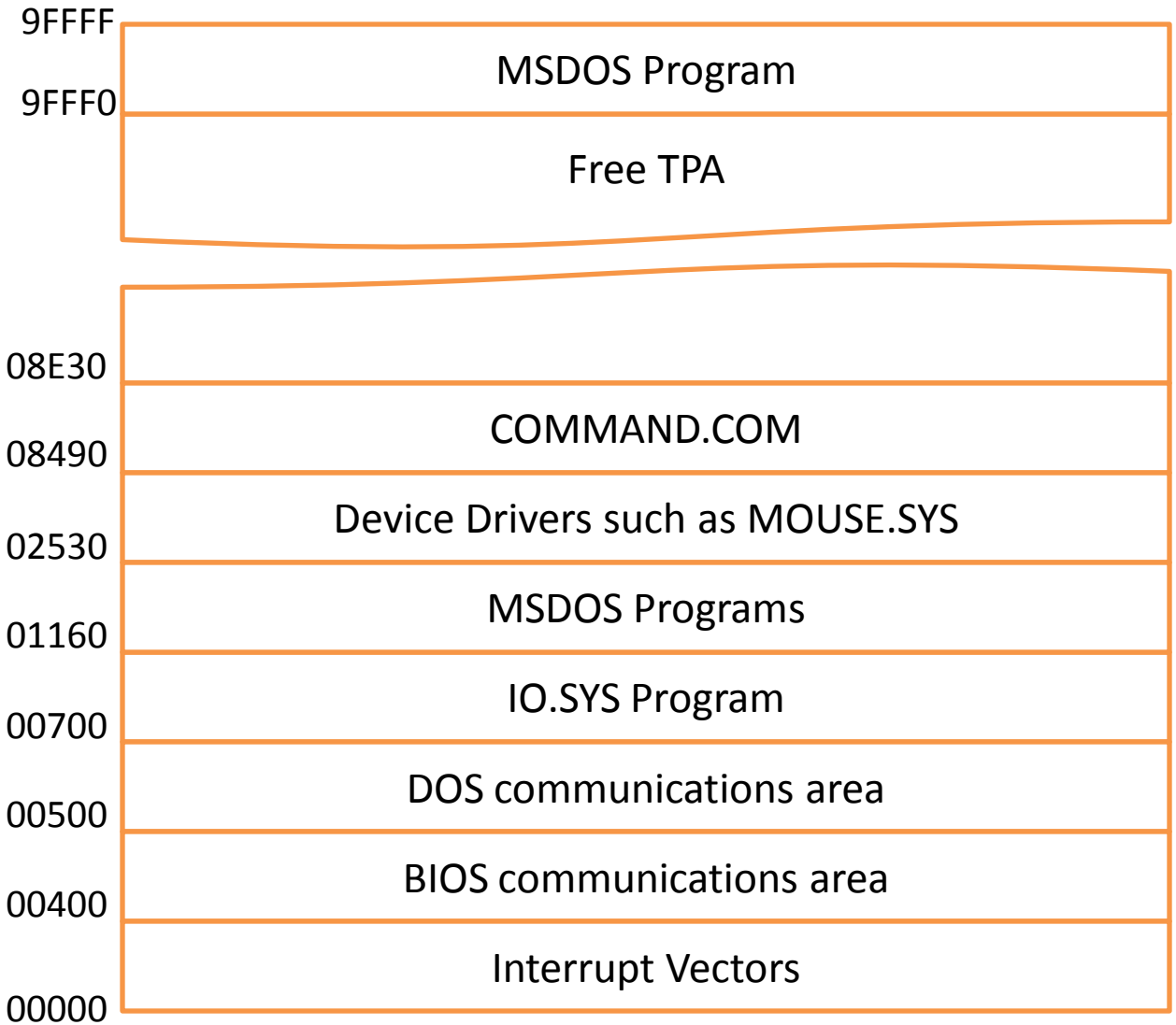
384 Kb

- Extended Memory System

(XMS) over 4MB

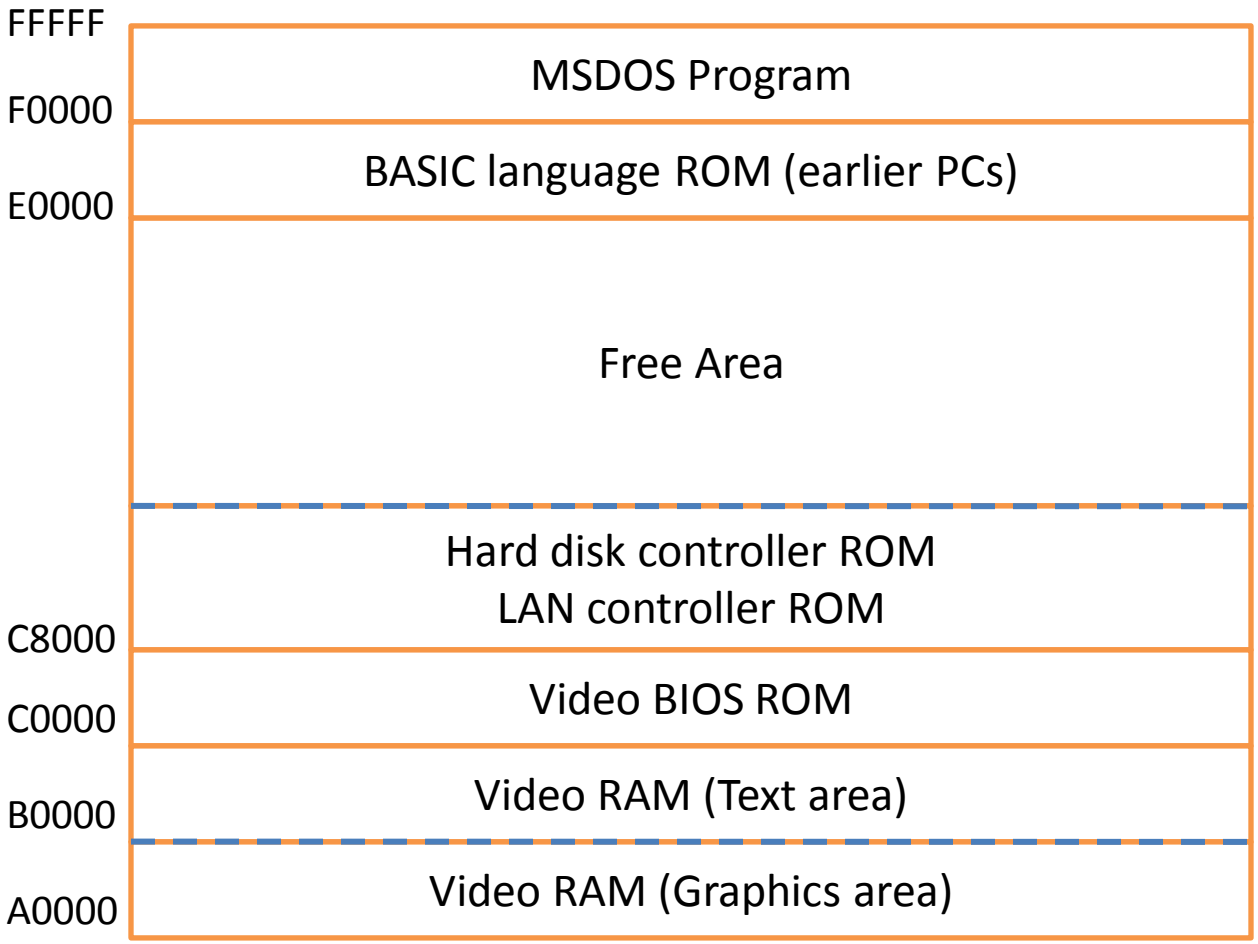


- Transient Program Area (TPA)

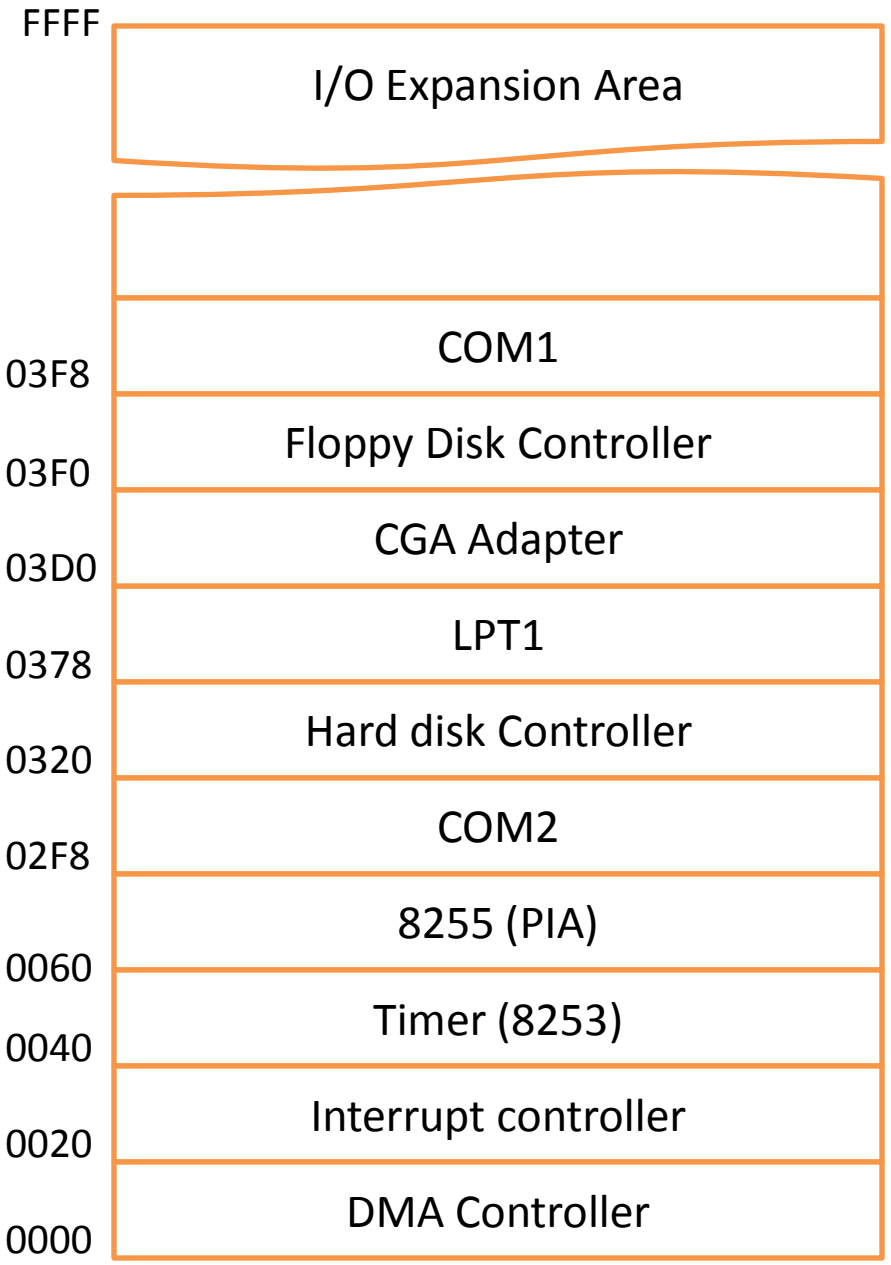


- Programs that control computer system (Operating Systems)
- Also contains data, drivers, and application programs
- Consists of RAM, ROM, EEPROM, and Flash Memory
- DOS controls memory organization and some I/O devices
- Interrupt Vectors contain addresses of interrupt service procedures
- BIOS (Basic I/O system) area controls I/O devices
- IO program allows use of keyboard, video display, printer, etc.
- Command program controls operation of computer through keyboard

- System Area


FFFFF
F0000
E0000
C8000
C0000
B0000
A0000
MSDOS Program
BASIC language ROM (earlier PCs)
Free Area
Hard disk controller ROM
LAN controller ROM
Video BIOS ROM
Video RAM (Text area)
Video RAM (Graphics area)

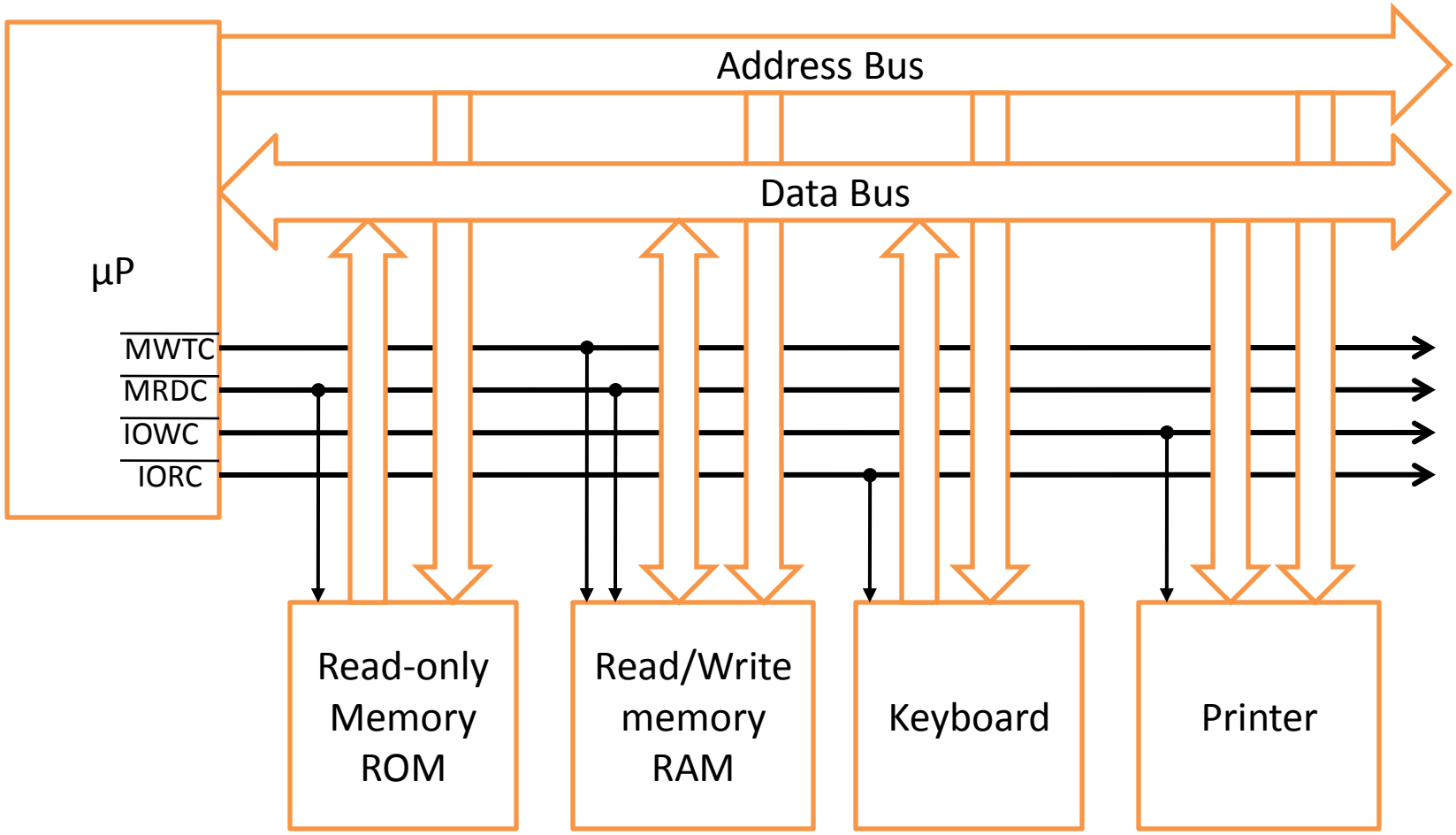
- I/O Space
  - Addresses I/O ports
  - Up to 64K 8-bit devices



Microprocessor

- Data transfer between itself and memory or I/O system
  - Using data, address, and control buses
- Simple arithmetic and logic operations
  - Add, Sub, Mul, Div, AND, OR, NOT, NEG, Shift, Rotate
  - Data width: byte (8-bit), word (16-bit), and double word (32-bit)
- Program flow via simple decisions
  - Zero, Sign, Carry, Parity, Overflow
- Why is it so important?

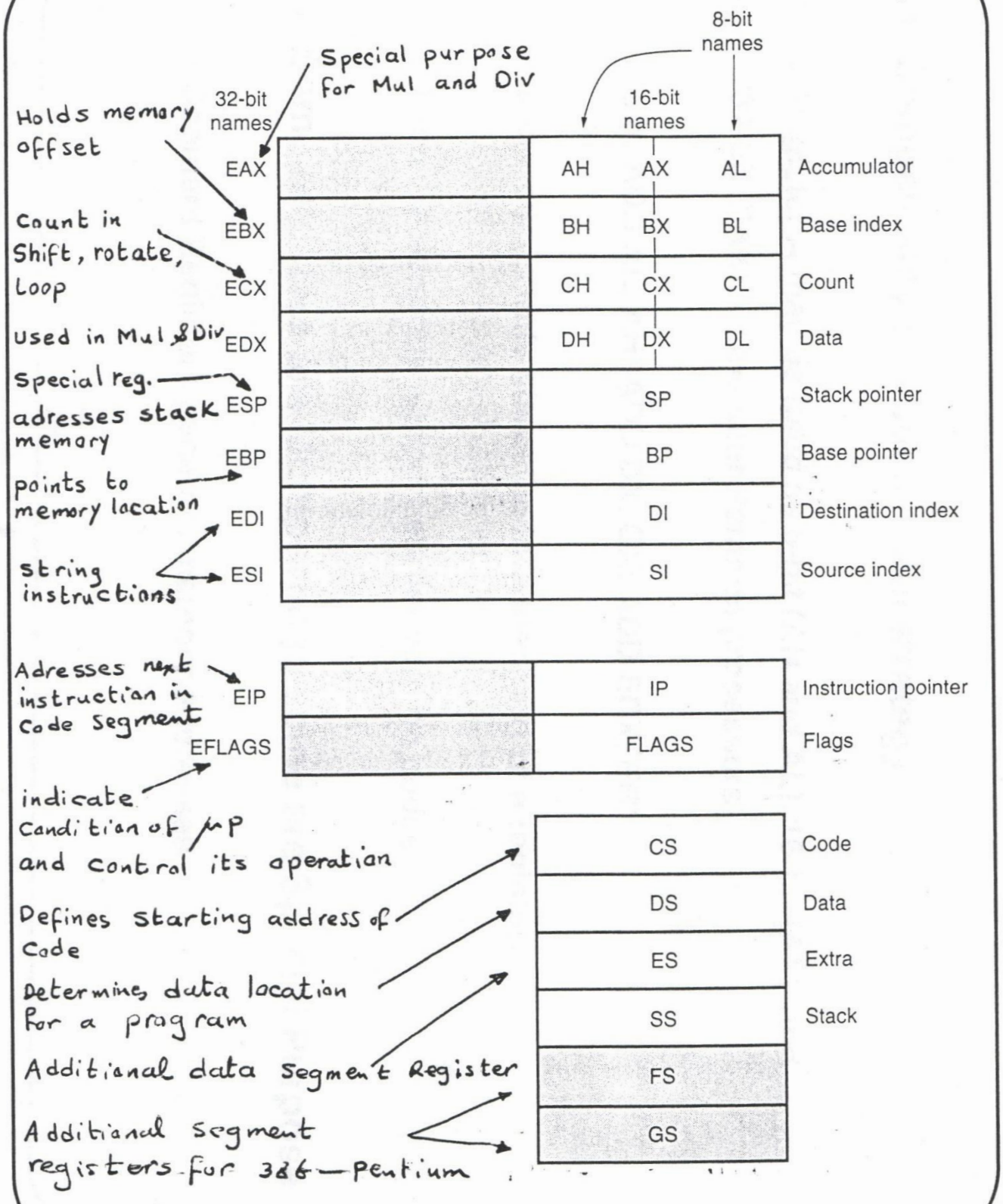
Computer System Block Diagram



- Bus is a common group of wires for interconnection
- Address Bus: 16-bit for I/O and 20 to 36-bit for memory
- Data Bus: 8 to 64-bit, the wider the bus, the more data can be transferred
- Control Bs: contains lines that selects the memory or I/O to perform a read or write operation
  - Four main control lines
  - MRDC‘ (memory read control)
  - MWTC’ (memory write control)
  - IORC’ (I/O read control)
  - IOWC’ (I/O write control)

Intel Microprocessor Architecture

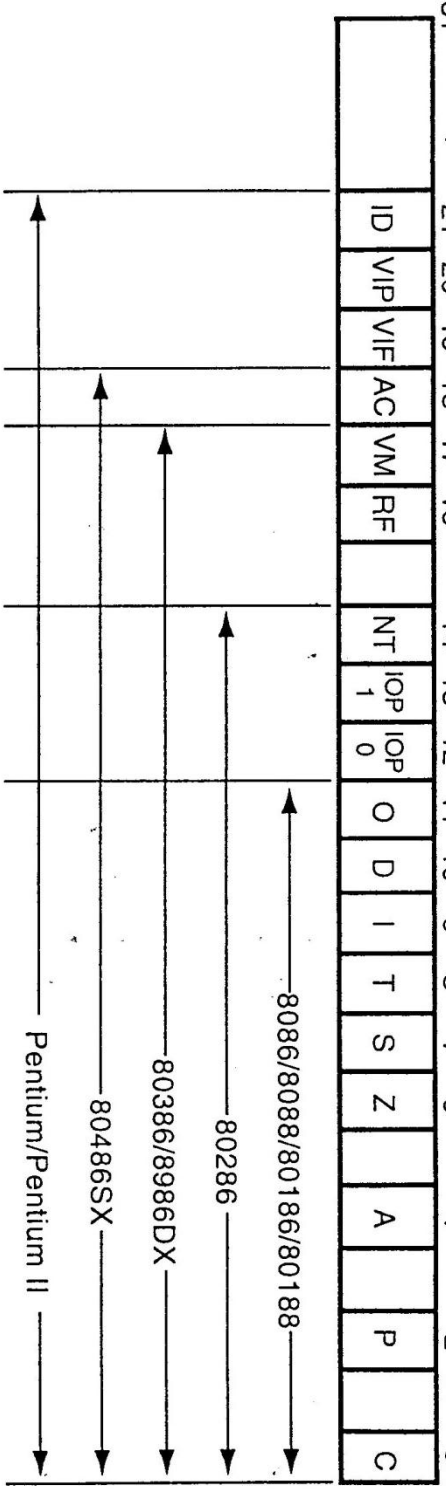
- Operation Modes
  - Real: uses 1st M byte of memory in all versions
  - Protected: uses all parts of memory in 80286 and above
- Register Types
  - Program Visible: used during application programs
  - Program Invisible: not directly addressable, but used by system
- Program Visible Registers
  - 4 Data Registers, 4 Pointer/Index Registers, 4-6 Segment Registers, Instruction Pointer, and Flags


8-bit names
Special purpose for Mul and Div
32-bit names
16-bit names
Holds memory offset
EAX
AH AX AL
Accumulator
Count in Shift, rotate, Loop
EBX
BH BX BL
Base index
ECX
CH CX CL
Count
Used in Mul & Div
EDX
DH DX DL
Data
Special reg. adresses stack memory
ESP
SP
Stack pointer
EBP
BP
Base pointer
points to memory location
EDI
DI
Destination index
string instructions
ESI
SI
Source index
Adresses next instruction in Code segment
EIP
IP
Instruction pointer
EFLAGS
FLAGS
Flags
indicate Condition of μP and control its operation
Defines starting address of Code
CS
Code
Determines data location for a program
DS
Data
ES
Extra
Additional data Segment Register
SS
Stack
FS
Additional segment registers for 386—Pentium
GS

- Compatibility is a successful strategy
  - Register A may be used as 8-bit (AH and AL), 16-bit (AX), and 32-bit (EAX) fir the later Pentium processors
  - e.g. ADD AL, AH; ADD DX, CX; ADD ECX, EBX
  - Instructions only affect the intended part of a register
  - Later µP versions support earlier version codes

- Some registers are Multipurpose, some are Special Purpose
  - Segment Registers generate memory addresses

Flag Register



21 ID flag, indicates support for CPUID istruction (µP Info)

20 Virtual Interrupt Pending

19 Virtual Interrupt Flag — Pentium

18 Alignment Check (486SX µP)

17 Virtual Mode, multiple DOS memory

16 Resume execution after debugging

14 Nested task (software)

13 Highest = 00, Lowest = 11

12 I/O privilege level

11 Overflow for signed operations

10 Direction flag, for DI/SI register

9 Interrupt flag, enable/disable INTR

8 Trap flag for debugging

7 Sign flag, 1 = '−', Ø = '+'

6 Zero flag
result = Ø → Z=1, result ≠ Ø → Z = Ø

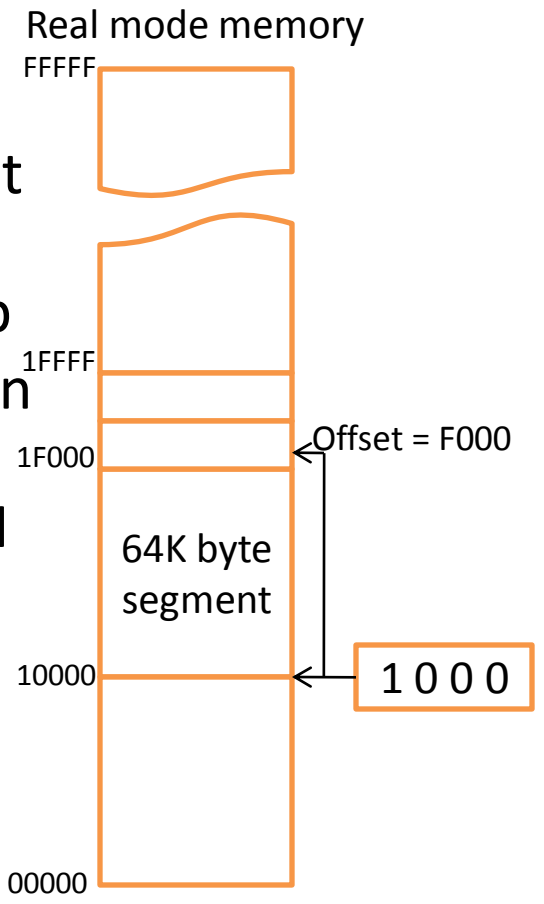
4 Auxiliary carry
holds ½ carry

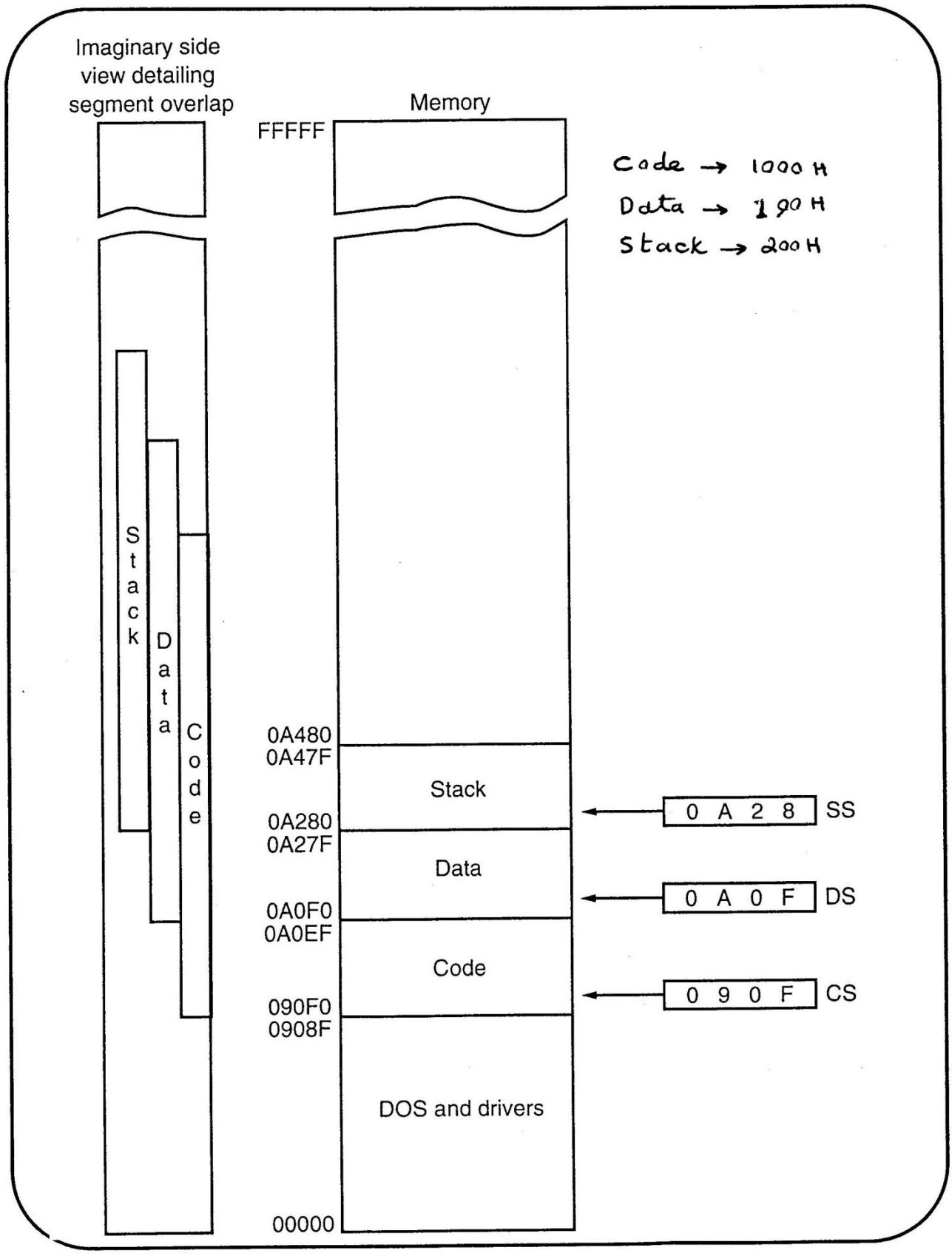
2 Ø for odd parity, 1 for even parity
Parity = number of ones

0 Holds carry/borrow & indicates errors

Real Mode Memory Addressing

- Location = Segment + Offset
  - Segment address located in a segment register; always appended with 0H
  - Segments always have length of 64 Kb
  - Offset or displacement selects location within 64 Kb of segment
  - e.g. 1000:2000 gives location 12000H
- Default Segment and Address Registers
  - e.g. code segment and instruction pointer CS:IP and stack segment and stack pointer SS:SP



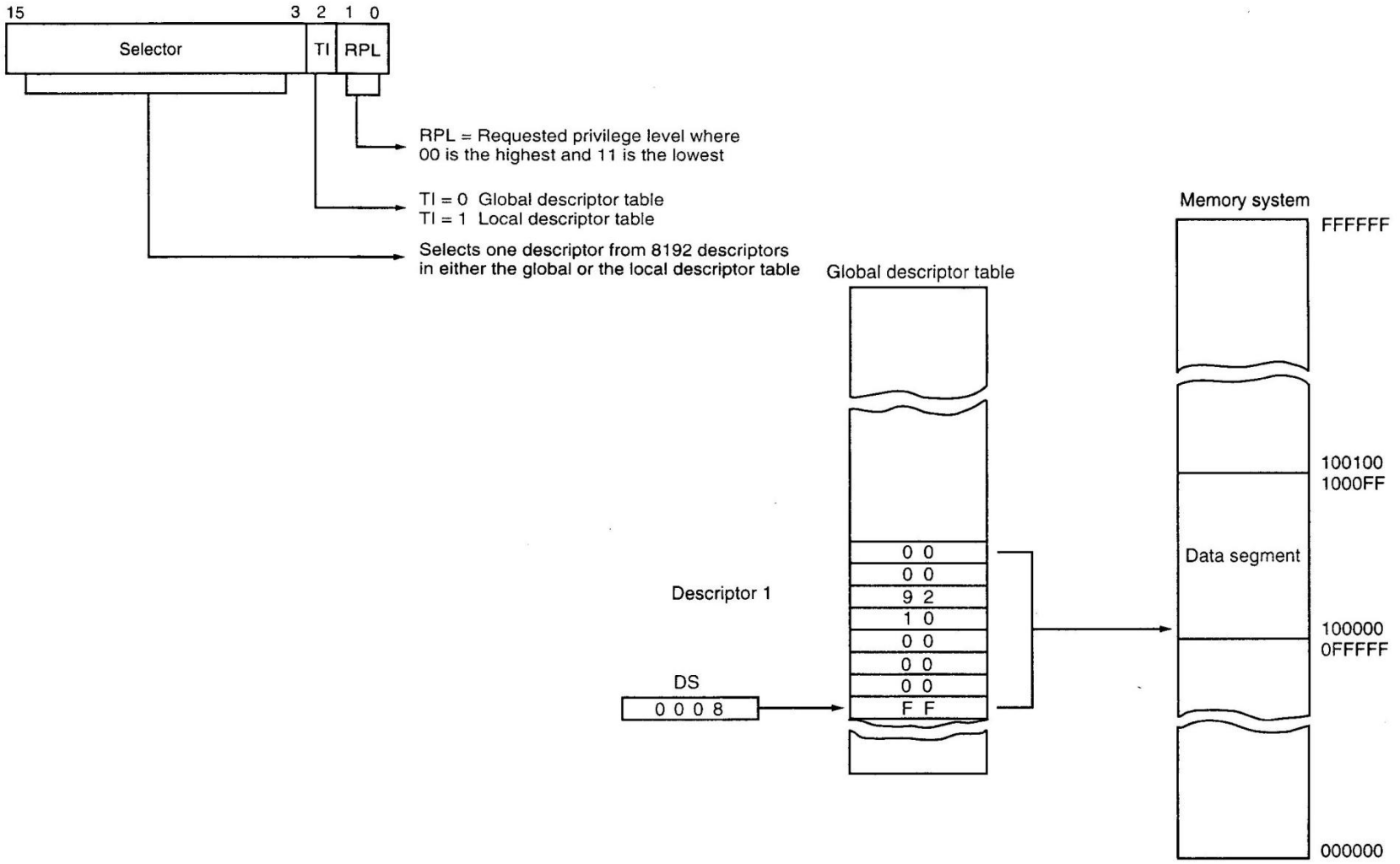

Imaginary side view detailing segment overlap
Memory
FFFFF
Code → 1000 H
Data → 190 H
Stack → 200 H
Stack
Data
Code
0A480
0A47F
Stack
0A280
0A27F
Data
0A0F0
0A0EF
Code
090F0
0908F
DOS and drivers
00000
0 A 2 8
SS
0 A 0 F
DS
0 9 0 F
CS

Protected Mode Memory Addressing

- Accessed via segment and offset address, but
  - Segment register contains a selector
  - Selector selects a descriptor from descriptor table
  - Descriptor: memory segment location, length, and access right
- Two types of descriptor tables
  - Global/system descriptors used for all programs
  - Local/application descriptors used for applications
  - Each descriptor is 8 bytes

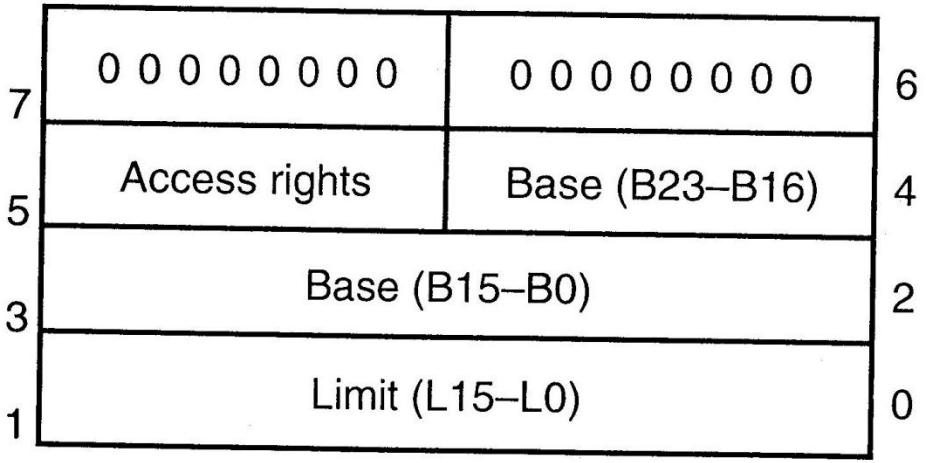
- 16-bit segment register contains 3 parts
  - Left most 13 bits address a descriptor
  - TI bit access global (0) or local descriptor (1) table
  - Right most 2 bits select priority for memory segment access
- How many global and local descriptors in a table?
- How large is a global and a local descriptor table?
- How many memory segments are allowed?

Segment Register

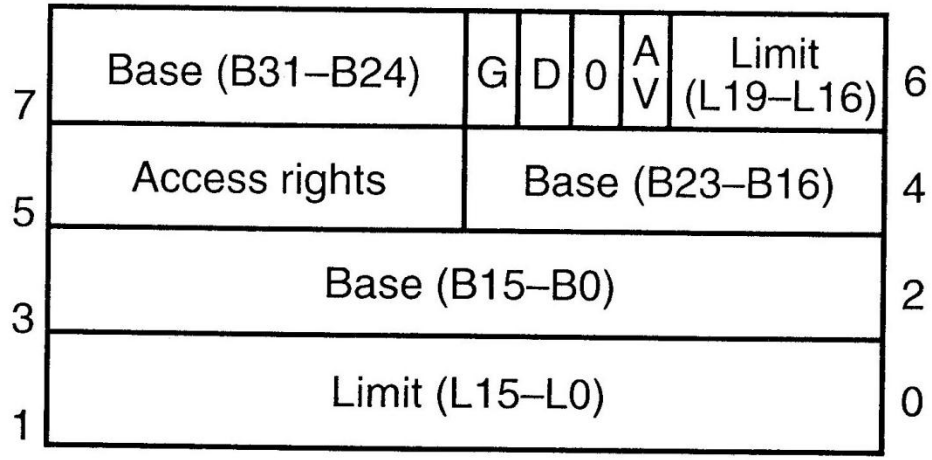


Descriptor Formats

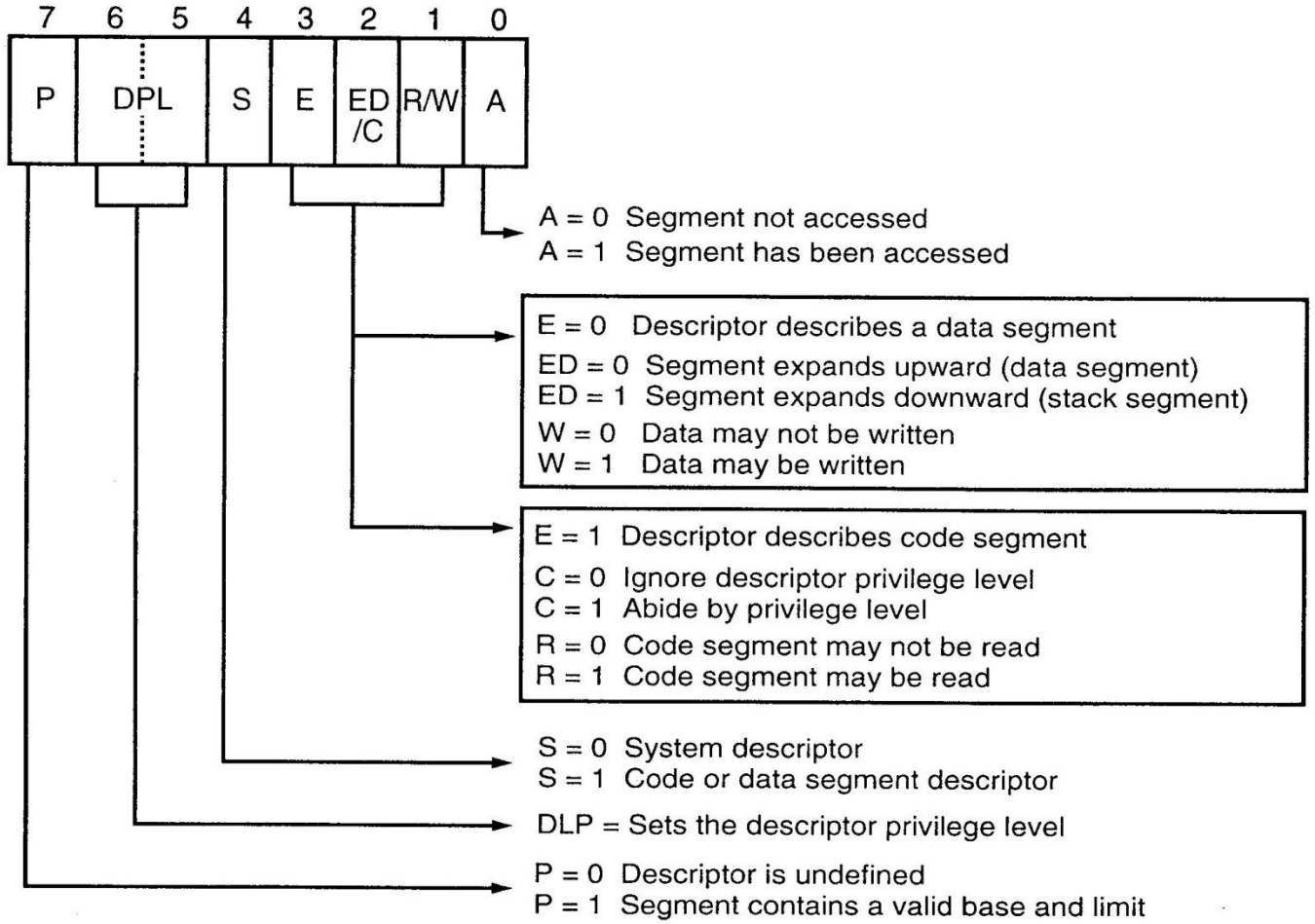
80286 descriptor



80386/80486/Pentium/Pentium Pro/Pentium II descriptor



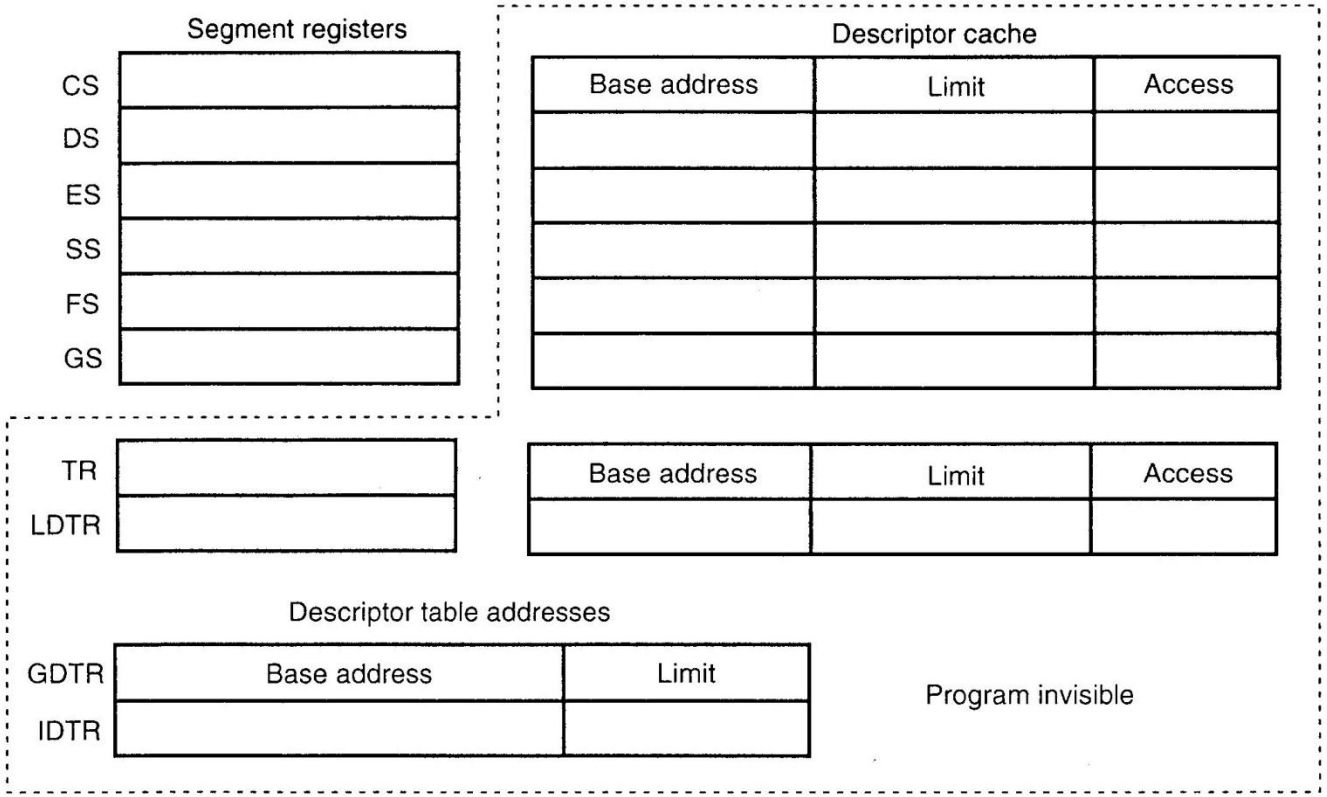
Access Right Byte



Note: Some of the letters used to describe the bits in the access rights bytes vary in Intel documentation.

Program-Invisible Registers

- Each segment register contains a program-invisible portion
  - This register is re-loaded when segment register change
  - Contains base-address, limit, and access information
  - These registers also called descriptor cache
- Other program-invisible registers
  - GDTR (global descriptor table register) contain base address and limit for descriptor table
  - Location of local descriptor table is selected from global descriptor table using the selector held in LDTR (local descriptor table register)

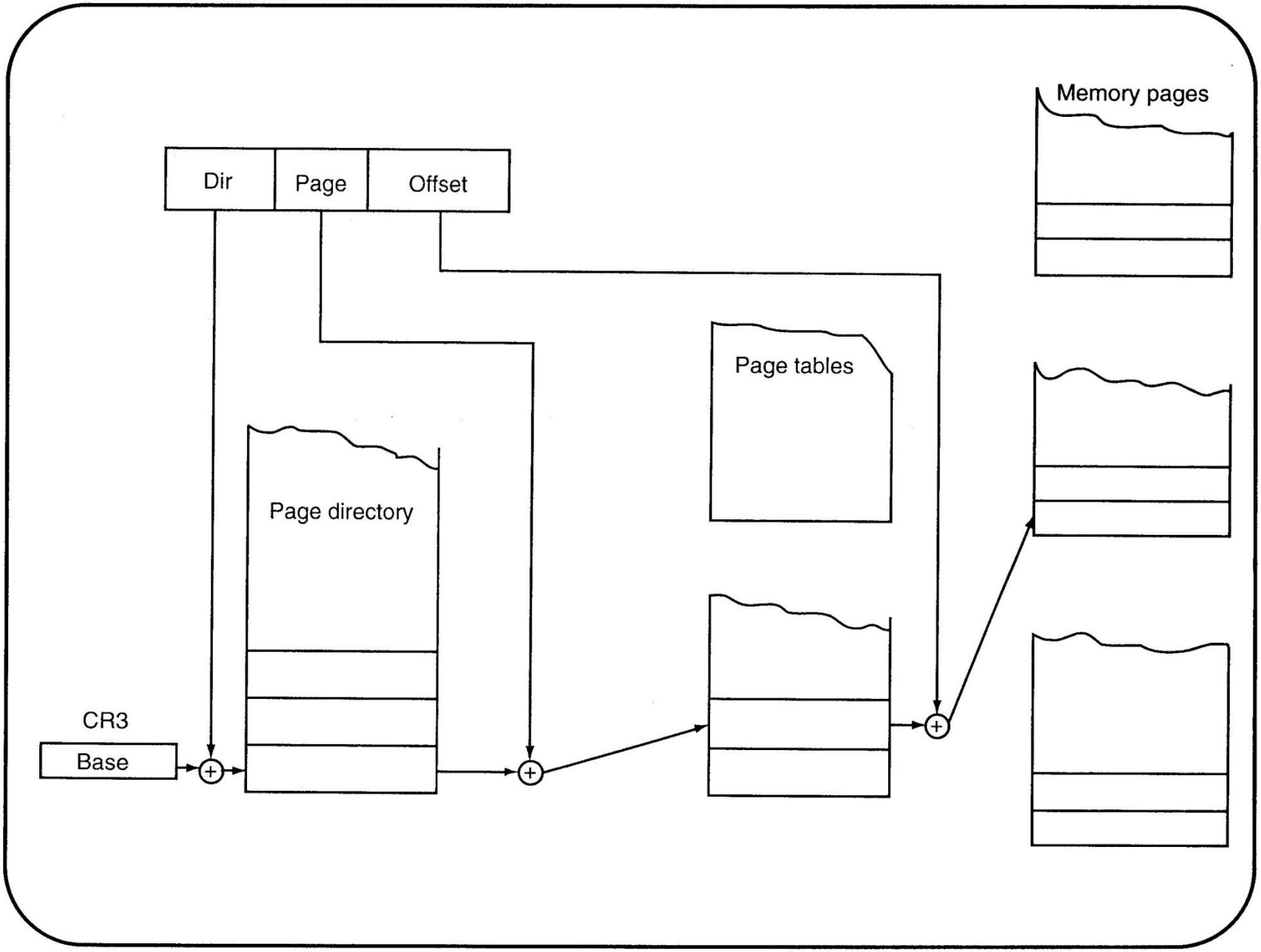


Notes:

1. The 80286 does not contain FS and GS nor the program-invisible portions of these registers.
2. The 80286 contains a base address that is 24-bits and a limit that is 16-bits.
3. The 80386/80486/Pentium/Pentium Pro contain a base address that is 32-bits and a limit that is 20-bits.
4. The access rights are 8-bits in the 80286 and 12-bits in the 80386/80486/Pentium.

Memory Paging

- Memory paging changes a linear address to physical
  - Linear address is produced by software
  - Page directory base is held in a control register (CR3)
  - Linear address is broken into 3 sections: directory, page table, offset
  - Page directory contains 1024 entries of 4 bytes each which addresses a page table that contains 1024 entries of 4 bytes each
  - Each memory page is 4K bytes
  - TLB (table look aside buffer) is a cache which contains the 32 most recent page translation addresses


Memory pages
Dir
Page
Offset
Page tables
Page directory
CR3
Base

Addressing Modes

Data Addressing Modes

- Intel family supports 8 data addressing modes
- Modes differ in the location of data and address calculations
- All modes involve physical address generation
- Consider MOV opcode as example: MOV AX, BX
  - Opcode or operation code tells µP which operation to perform
  - Source operand is to the right
  - Destination operand is to the left

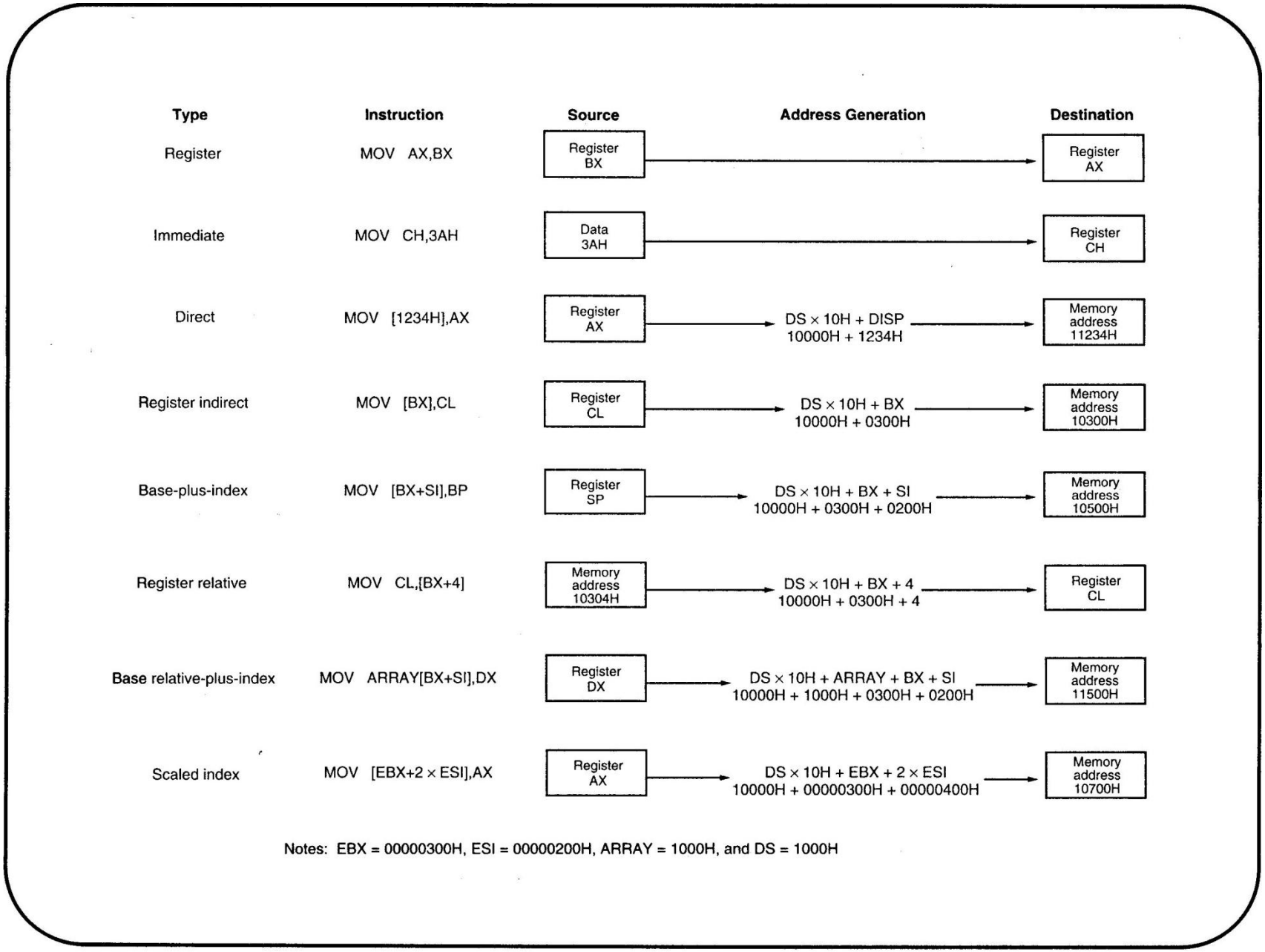
- Register Addressing: MOV CX, DX
  - Copy content of source register to destination register
  - Source and destination must be of the same size
- Immediate Addressing: MOV AL, 22H
  - Transfer the immediate data into destination register
  - This is called constant data, but data transferred from a register is a variable data
- Direct Addressing: MOV CX, LIST
  - Move a byte or word between a memory location and a register
  - Memory address, instead of data, appears in the instruction

- Register Indirect Addressing: MOV AX, [BX]
  - Transfer data between a register and a memory location addressed by a register
  - Sometimes need using special assembler directives BYTE PTR, WORD PTR, DWORD PTR, when size is not clear
  - FOR example MOV DWORD PTR [DI], 10H instead of MOV [DI], 10H
- Base-plus-index Addressing: MOV [BX+DX], CL
  - Transfer data between a register and a memory location addressed by a base register and an index register
- Register Relative Addressing: MOV AX, [BX+4]
  - Move data between a register and a memory location addressed specified by a register plus a displacement

- Base relative-plus-index Addressing:

  MOV AX, ARRAY[BX+DI]
  - Transfer data between a register and a memory location specified by a base and index register plus a displacement
  - Another example is MOV AX, [BX+DI+4]

- Scaled-index Addressing: MOV EDX, [EAX+4*EBX]
  - Address in the second register is modified by a scale factor
  - Scale factor are 2, 4, or 8, word, double-word, and quad-word access, respectively
  - Only available in 80386 through µP
  - Other examples: MOV AL, [EBX+ECX] and MOV AL, [2*EBX]



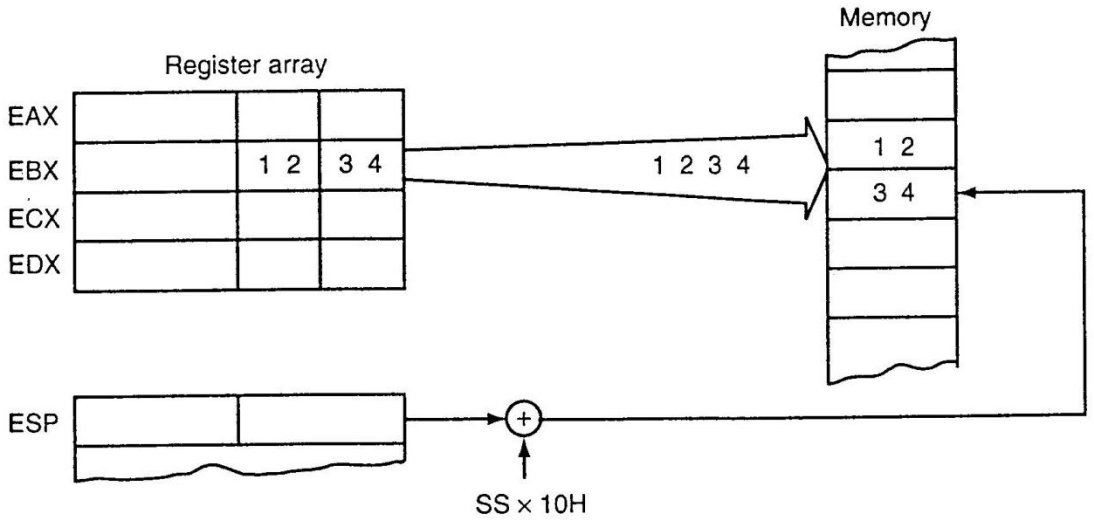
Notes: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, and DS = 1000H

Program Memory-Addressing Modes

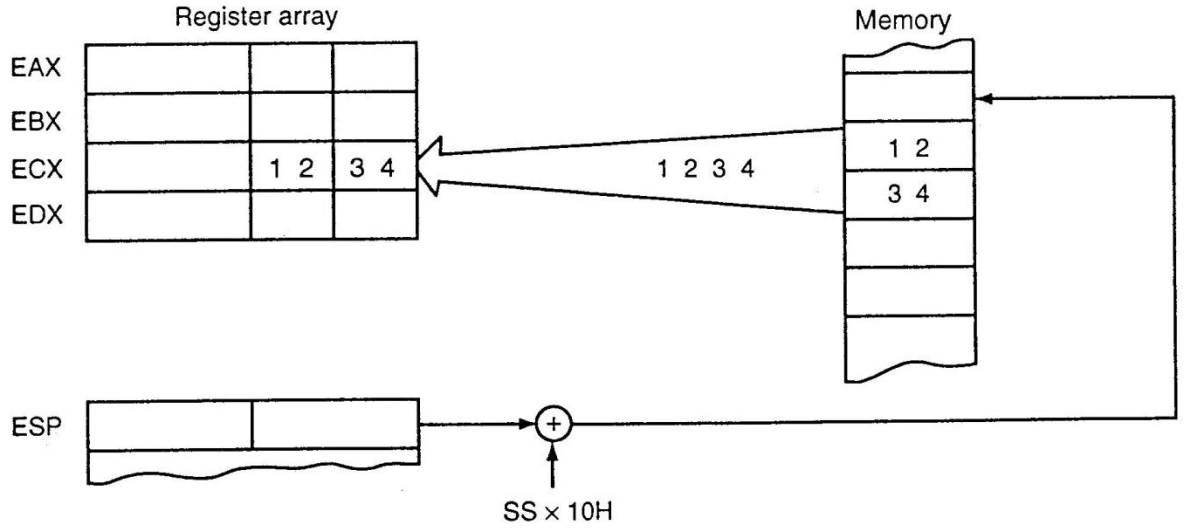
- Three forms, used with JMP and CALL instructions
- Direct Program Memory Addressing: LMP Label
  - Like GOTO or GOSUB in BASIC language
  - Allows going to any location in memory for next instruction
- Relative Program Memory Addressing: JMP [2]
  - Jump relative to instruction pointer (IP)
- Indirect Program Memory Addressing: JMP AX
  - Jump to current code segment location addressed by content of AX
  - Other examples: JMP [DI+2[] and JMP [BX]

Stack Memory-Addressing Modes

- Stack is a LIFO (last-in, first-out memory)
- Data are place by PUSH and removed by POP
  - Stack memory is maintained by stack segment register (ss) and stack pointer (sp)
  - When a word is pushed, high 8 bits are stored at SP-1 low 8 bits are stored at SP-2, the SP is decremented by 2
  - When a word is poped, low 8 bits are removed from location addressed by SP, high 8 bits are removed from location addressed by SP+1, then SP is incremented by 2



(a)

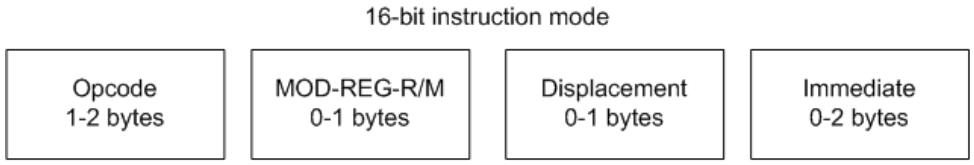


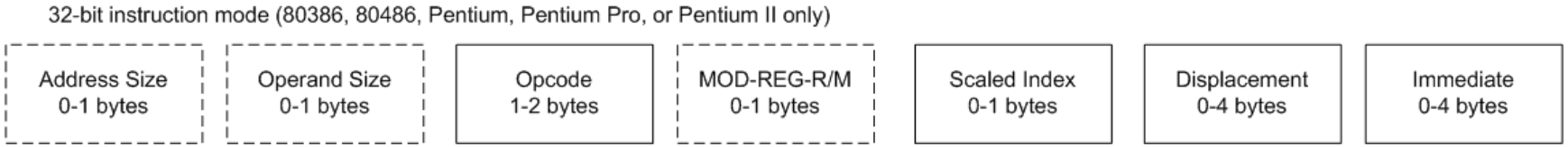
(b)

FIGURE 3–17 The PUSH and POP instructions. (a) PUSH BX places the contents of BX onto the stack, (b) POP CX removes data from the stack and places them into CX. Both instructions are shown after execution.

Instruction Encoding

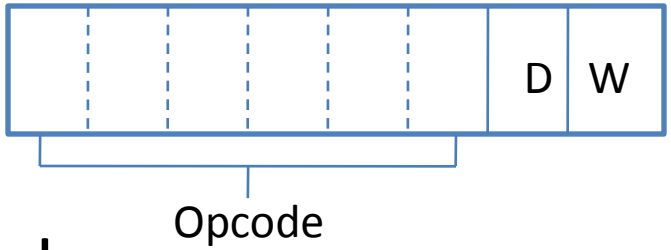
- Assembler translates assembly code into machine language
- Machine language is the native binary code µP understands





- Override Prefixes
  - First two bytes in 32-bit instructions:

    Address size-prefix (67H) and Register size-prefix (66H)
  - They toggle size of register and operand address from 16-bit to 32-bit or vice versa



- First byte of instruction: opcode
  - First 6 bits of instruction are the binary opcode
  - Direction bit (D) determines the direction of data flow
  - Width bit (W) determines data size: 0 for byte, 1 for word and double word

- Second byte of instruction: MOD-REG-R/M

  - MOD specifies addressing mode for instruction and whether displacement is present
  - If MOD=11, then register addressing mode, else memory addressing mod
  - In register addressing mode, R/M specifies a register
  - In memory addressing mode, R/M selects a mode from table
  - If D=1, data flow to REG from R/M, if D=0 data flow to R/M from REG

TABLE 4–1 MOD field for the 16-bit instruction mode.

MOD	*Function*
00	No displacement
01	8-bit sign-extended displacement
10	16-bit displacement
11	R/M is a register

TABLE 4–2 MOD field for the 32-bit instruction mode (80386–Pentium II only).

MOD	*Function*
00	No displacement
01	8-bit sign-extended displacement
10	32-bit displacement
11	R/M is a register

TABLE 4–3 REG and R/M (when MOD = 11) assignments.

Code	*W = 0 (Byte)*	*W = 1 (Word)*	*W = 1 (Doubleword)*
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

TABLE 4–4 16-bit R/M memory-addressing modes.

R/M Code	*Addressing Mode*
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:{BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]*
111	DS:[BX]

**Note:* See text section, Special Addressing Mode.

TABLE 4–5 32-bit addressing modes selected by R/M.

R/M Code	*Function*
000	DS:[EAX]
001	DS:[ECX[
010	DS:[EDX]
011	DS:[EBX]
100	Uses scaled-index byte
101	SS:[EBP]*
110	DS:[ESI]
111	DS:[EDI]

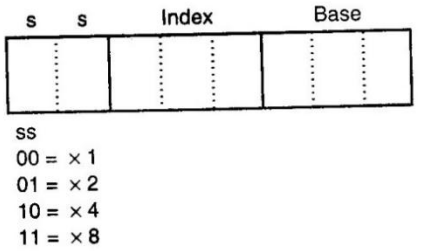
**Note:* See text section, Special Addressing Mode.

TABLE 4–6 Segment register selection.

Code	*Segment Register*
000	ES
001	CS*
010	SS
011	DS
100	FS
101	GS

**Note:* MOV CS,R/M(16) and POP CS are not allowed by the microprocessor. The FS and GS segments are only available to the 80386–Pentium II microprocessors.

FIGURE 4–8 The scaled-index byte.



Examples

Opcode = MOV
D = Transfer to register (REG)
W = Word
MOD = R/M is a register
REG = BP
R/M = SP

FIGURE 4–4 The 8BEC instruction placed into Byte 1 and 2 formats from Figures 4–2 and 4–3. This instruction is a MOV BP,SP.

Opcode = MOV
D = Transfer to register (REG)
W= Byte
MOD = No displacement
REG = DL
R/M = DS:[DI]

FIGURE 4–5 A MOV DL,[DI] instruction converted to its machine language form.

Intel Family Instruction Set

- PUSH and POP for stack operations
- Load Effective Address
  - LEA loads a 16- or 32-bit register with offset address
  - LDS, LES, LFS, LGS, and LSS load a 16- or 32-bit register with offset address and a corresponding segment register DS, ES, FS, GS, or SS with a segment address

- String Data Transfer
  - Uses destination index (DI) and source index (SI) registers
  - Two modes: auto-increment (D=0) and auto-decrement (D=1)

- By default DI access data in extra segment and SI in data segment
- LODS loads AL, AX, or EAX with data addressed by SI in data segment and increments or decrements SI
- STOS stores AL, AX or EAX at the extra segment addressed by DI and increments or decrements DI
- REPS STOS repeats the instruction the number of times stored in CX, i.e. terminates when CX=0
- MOVS is the only instruction that transfers data between memory locations
- INS transfers data from I/O device into extra segment addressed by DI; I/O address is in DX register
- OUTS transfers data from data segment memory addressed by SI to an I/O device addressed by DX

- For inputting or outputting a block of data INS and OUTS are repeated

- Miscellaneous Data Transfer Instructions
  - XCHG exchange contents of a register with any other register or memory location
  - IN and OUT instructions perform I/O operations
  - Two I/O addressing modes: fixed-port and variable port
  - In fixed-port addressing the port address appears in instructions, e.g. when using ROM
  - In variable-port addressing I/O address in a register
  - MOVSX is move and sign extend; MOVZX is move and zero-extend

  - CMOV new to Pentiums moves data only if condition is true; conditions are checked for some prior instruction results

- Segment Override Prefix
  - May be added to any instruction to deviate from default segment

- Arithmetic and Logic Instructions
  - ADD simply adds two numbers and sets the flags
  - ADC adds also the carry flag (C)
  - INC adds one to a register or memory location
  - SUB subtracts two and sets the flags
  - SBB subtract-with-borrow also subtracts (C) from difference

- DEC subtracts one from a register or memory location
- CMP is a subtract that only changes the flag bits; this is normally followed by a conditional jump instruction
- Multiplication can be unsigned (MUL) or signed (IMUL)
- Division can also be unsigned (DIV) or signed (IDIV)
- Basic logic instructions are AND, OR, XOR, NOT
- TEST is like CMP, but for bits zero flag Z=1 if bit is 0 and Z=0 if bit is 1
- TEST performs AND operation, so TEST AL,1 tests the first bit and TEST AL,128 tests the last bit of a byte in AL
- NOT is logical inversion or one's complement

- NEG is arithmetic sign inversion or two's complement

- Shift and Rotate Instructions
  - SHL and SHR are logical shift left and right that insert 0 and put one bit in the carry flag C
  - SAL and SAR are arithmetic shift operations; SAL is similar to SHL, but SAR is different than SHR because it inserts the sign bit instead of 0
  - Rotate instructions rotate data from one end to another, ROL (rotate left) and ROR (rotate right), or through the carry flag (RCL and RCR)

- String Data Comparing
  - String scan instruction SCAS compares register A with memory
  - Compare string instruction CMPS compares two memory locations

Symbolic	*Example*	*Note*
PUSH reg16	PUSH BX	16-bit register
PUSH reg32	PUSH EDX	32-bit register
PUSH mem16	PUSH WORD PTR [BX]	16-bit pointer
PUSH mem32	PUSH DWORD PTR [EBX]	32-bit pointer
PUSH seg	PUSH DS	Segment register
PUSH imm8	PUSH ','	8-bit immediate
PUSHW imm16	PUSHW 1000H	16-bit immediate
PUSHD imm32	PUSHD 20	32-bit immediate
PUSHA	PUSHA	Save all 16-bit registers
PUSHAD	PUSHAD	Save all 32-bit registers
PUSHF	PUSHF	Save flags
PUSHFD	PUSHFD	Save EFLAGs

Symbolic	*Example*	*Note*
POP reg16	POP CX	16-bit register
POP reg32	POP EBP	32-bit register
POP mem16	POP WORD PTR[BX+1]	16-bit pointer
POP mem32	POP DATA3	32-bit memory address
POP seg	POP FS	Segment register
POPA	POPA	Pop all 16-bit registers
POPAD	POPAD	Pop all 32-bit registers
POPF	POPF	Pop flags
POPFD	POPFD	Pop EFLAGs

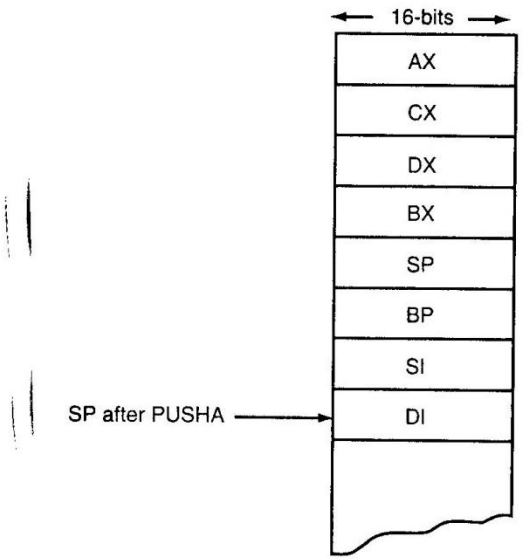


FIGURE 4–12 The operation of the PUSHA instruction, showing the location and order of stack data.

TABLE 4–9 Load-effective address instructions.

Assembly Language	*Operation*
LEA AX,NUMB	Loads AX with the address of NUMB
LEA EAX,NUMB	Loads EAX with the address of NUMB
LDS DI,LIST	Loads DS and DI with the 32-bit contents of data segment memory location LIST
LDS EDI,LIST	Loads DS and EDI with the 48-bit contents of data segment memory location LIST
LES BX,CAT	Loads ES and BX with the 32-bit contents of data segment memory location CAT
LFS DI,DATA1	Loads FS and DI with the 32-bit contents of data segment memory location DATA1
LGS SI,DATA5	Loads GS and SI with the 32-bit contents of data segment memory location DATA5
LSS SP,MEM	Loads SS and SP with the 32-bit contents of memory location MEM

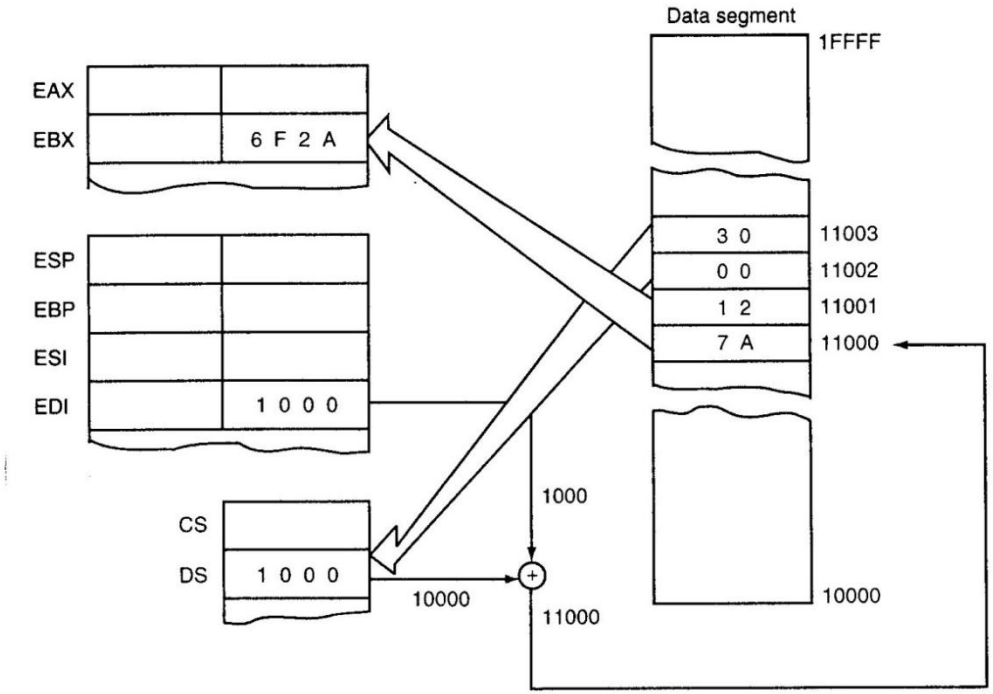


FIGURE 4–15 The LDS BX,[DI] instruction loads register BX from addresses 11000H and 11001H and register DS from locations 11002H and 11003H. This instruction is shown at the point just before DS changes to 3C00H and BX changes to 127AH.

TABLE 4–10 Forms of the LODS instruction.

Assembly Language	*Operation*
LODSB	AL = DS:[SI]; SI = SI ± 1
LODSW	AX = DS:[SI]; SI = SI ± 2
LODSD	EAX = DS:[SI]; SI = SI ± 4
LODS LIST	AL = DS:[SI]; SI = SI ± 1 (if LIST is a byte)
LODS DATA1	AX = DS:[SI], SI = SI ± 2 (if DATA1 is a word)
LODS FROG	EAX = DS:[SI]; SI = SI ± 4 (if FROG is a doubleword)

Note: The segment can be overridden with a segment override prefix as in LODS ES:DATA4.

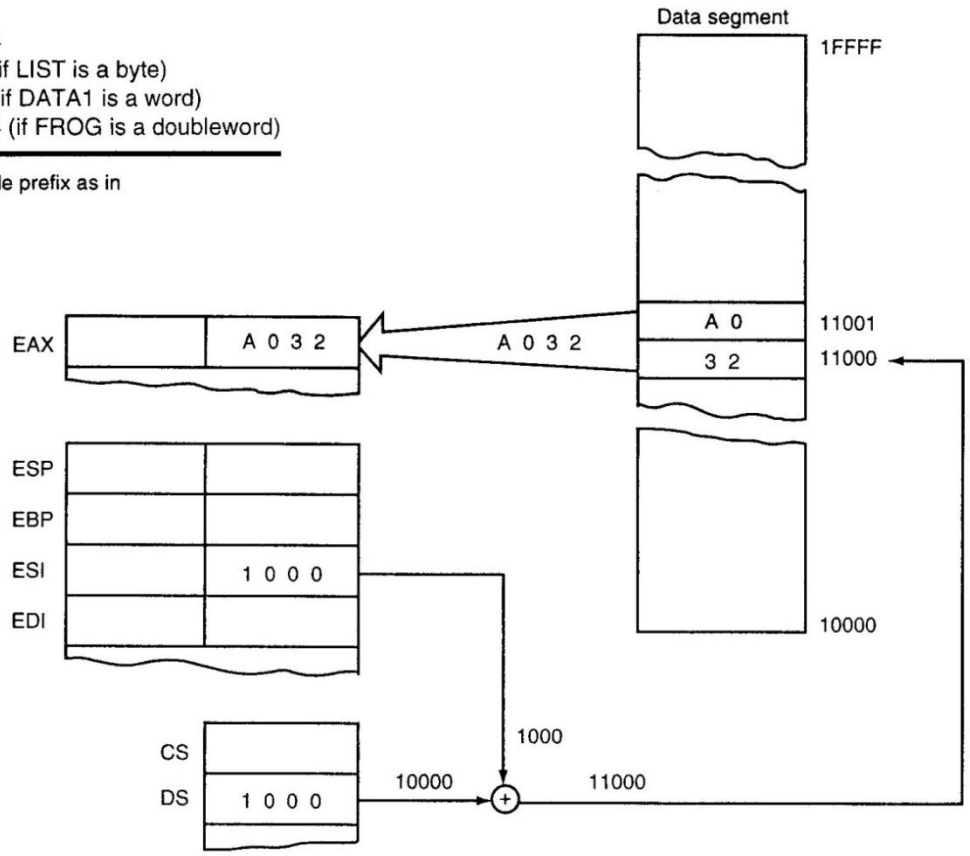


FIGURE 4–16 The operation of the LODSW instruction if DS = 1000H, D = 0, 11000H = 32, and 11001H = A0. This instruction is shown after AX is loaded from memory, but before SI increments by 2.

TABLE 4–11 Forms of the STOS instruction.

Assembly Language	*Operation*
STOSB	ES:[DI] = AL; DI = DI ± 1
STOSW	ES:[DI] = AX; DI = DI ± 2
STOSD	ES:[DI] = EAX; DI = DI ± 4
STOS LIST	ES:[DI] = AL; DI = DI ± 1 (if list is a byte)
STOS DATA3	ES:[DI] = AX; DI = DI ± 2 (if DATA3 is a word)
STOS DATA4	ES:[DI] = EAX; DI = DI ± 4 (if DATA4 is a doubleword)

TABLE 4–12 Common operand operators.

Operator	*Example*	*Comment*
+	MOV AL,6+3	Copies 9 into AL
–	MOV AL,8–2	Copies 6 into AL
*	MOV AL,4*3	Copies 12 into AL
/	MOV AX,12/5	Copies 2 into AX (remainder is lost)
MOD	MOV AX, 12 MOD 7	Copies 5 into AX (quotient is lost)
AND	MOV AX,12 AND 4	Copies 4 into AX (1100 AND 0100 = 0100)
OR	MOV AX,12 OR 1	Copies 13 into AX (1100 OR 0001 = 1101)
NOT	MOV AL,NOT 1	Copies 254 into AL (0000 0001 NOT equals 1111 1110 or 254)

TABLE 4–13 Forms of the MOVS instruction.

Assembly Language	*Operation*
MOVSB	ES:[DI] = DS:[SI]; DI = DI ± 1; SI = SI ± 1 (byte transferred)
MOVSW	ES:[DI] = DS:[SI]; DI = DI ± 2; SI = SI ± 2 (word transferred)
MOVSD	ES:[DI] = DS:[SI]; DI = DI ± 4; SI = SI ± 4 (doubleword transferred)
MOVS BYTE1,BYTE2	ES:[DI] = DS:[SI]; DI = DI ± 1; SI = SI ± 1 (if BYTE1 and BYTE2 are bytes)
MOVS WORD1,WORD2	ES:[DI] = DS:[SI]; DI = DI ± 2, SI = SI ± 2 (if WORD1 and WORD2 are words)
MOVS DWORD1, DWORD2	ES:[DI] = DS:[SI]; DI = DI ± 4; SI = SI ± 4 (if DWORD1 and DWORD2 are doublewords)

TABLE 4–14 Forms of the INS instruction.

Assembly Language	*Operation*
INSB	ES:[DI] = [DX]; DI = DI ± 1 (byte transferred)
INSW	ES:[DI] = [DX]; DI = DI ± 2 (word transferred)
INSD	ES:[DI] = [DX]; DI = DI ± 4 (doubleword transferred)
INS LIST	ES:[DI] = [DX]; DI = DI ± 1 (if LIST is a byte)
INS DATA4	ES:[DI] = [DX]; DI = DI ± 2 (if DATA4 is a word)
INS DATA5	ES:[DI] = [DX]; DI = DI ± 4 (if DATA5 is a doubleword)

Note: [DX] indicates that DX contains the I/O device address. These instructions are not available on the 8086/8088 microprocessors.

TABLE 4–15 Forms of the OUTS instruction.

Assembly Language	*Operation*
OUTSB	[DX] = DS:[SI]; SI = SI ± 1 (byte transferred)
OUTSW	[DX] = DS:[SI]; SI = SI ± 2 (word transferred)
OUTSD	[DX] = DS:[SI]; SI = SI ± 4 (doubleword transferred)
OUTS DATA7	[DX] = DS:[SI]; SI = SI ± 1 (if DATA7 is a byte)
OUTS DATA8	[DX] = DS:[SI]; SI = SI ± 2 (if DATA8 is a word)
OUTS DATA9	[DX] = DS:[SI]; SI = SI ± 4 (if DATA9 is a doubleword)

Note: [DX] indicates that DX contains the I/O device address. These instructions are not available on the 8086/8088 microprocessors.

TABLE 4–16 Forms of the XCHG instruction.

Assembly Language	*Operation*
XCHG AL,CL	Exchanges the contents of AL with CL
XCHG CX,BP	Exchanges the contents of CX with BP
XCHG EDX,ESI	Exchanges the contents of EDX with ESI
XCHG AL,DATA2	Exchanges the contents of AL with data segment memory location DATA2

TABLE 4–17 IN and OUT instructions.

Assembly Language	*Operation*
IN AL,p8	8-bits are input to AL from I/O port p8
IN AX,p8	16-bits are input to AX from I/O port p8
IN EAX,p8	32-bits are input to EAX from I/O port p8
IN AL,DX	8-bits are input to AL from I/O port DX
IN AX,DX	16-bits are input to AX from I/O port DX
IN EAX,DX	32-bits are input to EAX from I/O port DX
OUT p8,AL	8-bits are output from AL to I/O port p8
OUT p8,AX	16-bits are output from AX to I/O port p8
OUTp8,EAX	32-bits are output from EAX to I/O port p8
OUT DX,AL	8-bits are output from AL to I/O port DX
OUT DX,AX	16-bits are output from AX to I/O port DX
OUT DX,EAX	32-bits are output from EAX to I/O port DX

Note: p8 = an 8-bit I/O port number and DX = the 16-bit port address held in DX.

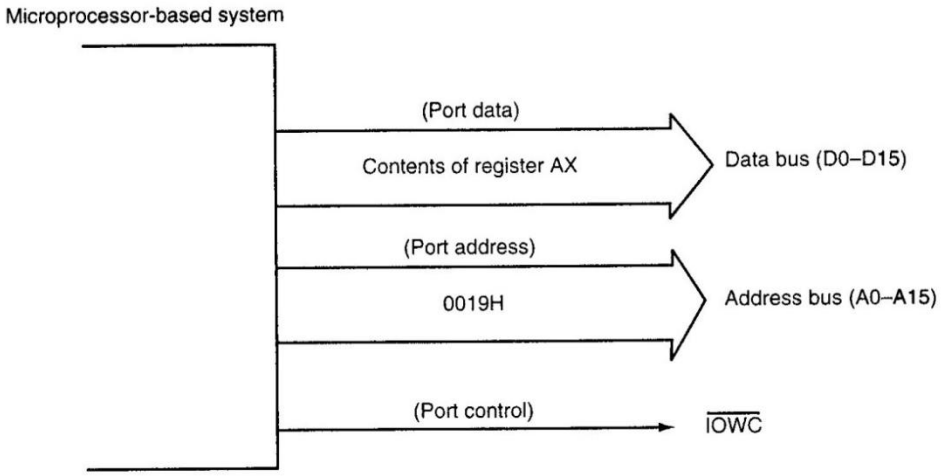


FIGURE 4–18 The signals found in the microprocessor-based system for an OUT 19H,AX instruction.

TABLE 4–18 The MOVSX and MOVZX instructions.

Assembly Language	*Operation*
MOVSX CX,BL	Sign-extends BL into CX
MOVSX ECX,AX	Sign-extends AX into ECX
MOVSX BX,DATA1	Sign-extends the byte at DATA1 into BX
MOVSX EAX,[EDI]	Sign-extends the word at the data segment memory location addressed by EDI into EAX
MOVZX DX,AL	Zero-extends AL into DX
MOVZX EBP,DI	Zero-extends DI into EBP
MOVZX DX,DATA2	Zero-extends the byte at data segment memory location DATA2 into DX
MOVZX EAX,DATA3	Zero-extends the word at data segment memory location DATA3 into EAX

TABLE 4–19 The conditional move instructions.

Assembly Language	*Condition Tested*	*Operation*
CMOVB	C = 1	Move if below
CMOVAE	C = 0	Move if above or equal
CMOVBE	Z = 1 or C = 1	Move if below or equal
CMOVA	Z = 0 and C = 0	Move if above
CMOVE or CMOVZ	Z = 1	Move if equal or set if zero
CMOVNE or CMOVNZ	Z = 0	Move if not equal or set if not zero
CMOVL	S <> O	Move if less than
CMOVLE	Z = 1 or S <> O	Move if less than or equal
CMOVG	Z = 0 and S = O	Move if greater than
CMOVGE	S = O	Move if greater than or equal
CMOVS	S = 1	Move if sign (negative)
CMOVNS	S = 0	Move if no sign (positive)
CMOVC	C = 1	Move if carry
CMOVNC	C = 0	Move if no carry
CMOVO	O = 1	Move if overflow
CMOVNO	O = 0	Move if no overflow
CMOVP or CMOVPE	P = 1	Move if parity or set if parity even
CMOVNP or CMOVPO	P = 0	Move if no parity or set if parity odd

TABLE 4–20 Instructions that include segment override prefixes.

Assembly Language	*Segment Accessed*	*Default Segment*
MOV AX,DS:[BP]	Data	Stack
MOV AX,ES:[BP]	Extra	Stack
MOV AX,SS:[DI]	Stack	Data
MOV AX,CS:LIST	Code	Data
MOV AX,ES:[SI]	Extra	Data
LODS ES:DATA1	Data	Extra
MOV EAX,FS:DATA2	Data	FS
MOV BL,GS:[ECX]	Data	GS

TABLE 5–1 Addition instructions.

Assembly Language	*Operation*
ADD AL,BL	AL = AL + BL
ADD CX,DI	CX = CX + DI
ADD EBP,EAX	EBP = EBP + EAX
ADD CL,44H	CL = CL + 44H
ADD BX,245FH	BX = BX + 245FH
ADD EDX,12345H	EDX = EDX + 00012345H
ADD [BX],AL	AL adds to the contents of the data segment memory location addressed by BX with the sum stored in the same memory location
ADD CL,[BP]	The byte contents of the stack segment memory location addressed by BP add to CL with the sum stored in CL
ADD AL,[EBX]	The byte contents of the data segment memory location addressed by EBX add to AL with the sum stored in AL
ADD BX,[SI + 2]	The word contents of the data segment memory location addressed by the sum of SI plus 2 add to BX with the sum stored in BX
ADD CL,TEMP	The byte contents of the data segment memory location TEMP add to CL with the sum stored in CL
ADD BX,TEMP[DI]	The word contents of the data segment memory location addressed by TEMP plus DI add to BX with the sum stored in BX
ADD [BX + DI],DL	DL adds to the contents of the data segment memory location addressed by BX plus DI with the sum stored in the same memory location
ADD BYTE PTR [DI],3	A 3 adds to the byte contents of the data segment memory location addressed by DI
ADD BX,[EAX + 2*ECX]	The word contents of the data segment memory location addressed by the sum of 2 times ECX plus EAX add to BX with the sum stored in BX

TABLE 5–3 Add-with-carry instructions.

Assembly Language	*Operation*
ADC AL,AH	AL = AL + AH + carry
ADC CX,BX	CX = CX + BX + carry
ADC EBX,EDX	EBX = EBX + EDX + carry
ADC DH,[BX]	The byte contents of the data segment memory location addressed by BX add to DH with carry with the sum stored in DH
ADC BX,[BP + 2]	The word contents of the stack segment memory location addressed by BP plus 2 add to BX with carry with the sum stored in BX
ADC ECX,[EBX]	The doubleword contents of the data segment memory location addressed by EBX add to ECX with carry with the sum stored in ECX

TABLE 5–2 Increment instructions.

Assembly Language	*Operation*
INC BL	BL = BL + 1
INC SP	SP = SP + 1
INC EAX	EAX = EAX + 1
INC BYTE PTR [BX]	Adds 1 to the byte contents of the data segment memory location addressed by BX
INC WORD PTR [SI]	Adds 1 to the word contents of the data segment memory location addressed by SI
INC DWORD PTR [ECX]	Adds 1 to the doubleword contents of the data segment memory location addressed by ECX
INC DATA1	Increments the contents of data segment memory location DATA1

TABLE 5–4 Subtraction instructions.

Assembly Language	*Operation*
SUB CL,BL	CL = CL – BL
SUB AX,SP	AX = AX – SP
SUB ECX,EBP	ECX = ECX – EBP
SUB DH,6FH	DH = DH – 6FH
SUB AX,0CCCCH	AX = AX – CCCCH
SUB ESI,2000300H	ESI = ESI – 2000300H
SUB [DI],CH	Subtracts the contents of CH from the contents of the data segment memory location addressed by DI
SUB CH,[BP]	Subtracts the byte contents of the stack segment memory location addressed by BP from CH
SUB AH,TEMP	Subtracts the byte contents of the data segment memory location TEMP from AH
SUB DI,TEMP[ESI]	Subtracts the word contents of the data segment memory location addressed by TEMP plus ESI from DI
SUB ECX,DATA1	Subtracts the doubleword contents of the data segment memory location addressed by DATA1 from ECX

TABLE 5–5 Decrement instructions.

Assembly Language	*Operation*
DEC BH	BH = BH – 1
DEC CX	CX = CX – 1
DEC EDX	EDX = EDX – 1
DEC BYTE PTR [DI]	Subtracts 1 from the byte contents of the data segment memory location addressed by DI
DEC WORD PTR[BP]	Subtracts 1 from the word contents of the stack segment memory location addressed by BP
DEC DWORD PTR[EBX]	Subtracts 1 from the doubleword contents of the data segment memory location addressed by EBX
DEC NUMB	Subtracts 1 from the contents of the data segment memory location NUMB

TABLE 5–7 Comparison instructions.

Assembly Language	*Operation*
CMP CL,BL	CL – BL
CMP AX,SP	AX – SP
CMP EBP,ESI	EBP – ESI
CMP AX,2000H	AX – 2000H
CMP [DI],CH	CH subtracts from the contents of the data segment memory location addressed by DI
CMP CL,[BP]	The byte contents of the stack segment memory location addressed by BP subtract from CL
CMP AH,TEMP	The byte contents of the data segment memory location TEMP subtract from AH
CMP DI,TEMP[BX]	The word contents of the data segment memory location addressed by the sum of TEMP plus BX subtract from DI
CMP AL,[EDI + ESI]	The byte contents of the data segment memory location addressed by the sum of EDI plus ESI subtract from AL

TABLE 5–8 8-bit multiplication instructions.

Assembly Language	*Operation*
MUL CL	AL is multiplied by CL; the unsigned product is in AX
IMUL DH	AL is multiplied by DH; the signed product is in AX
IMUL BYTE PTR[BX]	AL is multiplied by the byte contents of the data segment memory location addressed by BX; the signed product is in AX
MUL TEMP	AL is multiplied by the byte contents of the data segment memory location addressed by TEMP; the unsigned product is in AX

TABLE 5–9 16-bit multiplication instructions.

Assembly Language	*Operation*
MUL CX	AX is multiplied by CX; the unsigned product is in DX–AX
IMUL DI	AX is multiplied by DI; the signed product is in DX–AX
MUL WORD PTR[SI]	AX is multiplied by the word contents of the data segment memory location addressed by SI; the unsigned product is in DX–AX

TABLE 5–10 32-bit multiplication instructions.

Assembly Language	*Operation*
MUL ECX	EAX is multiplied by ECX; the unsigned product is in EDX–EAX
IMUL EDI	EAX is multiplied by EDI; the signed product is in EDX–EAX
MUL DWORD PTR[ECX]	EAX is multiplied by the doubleword contents of the data segment memory location addressed by ECX; the unsigned product is in EDX–EAX

TABLE 5–11 8-bit division instructions.

Assembly Language	*Operation*
DIV CL	AX is divided by CL; the unsigned quotient is in AL and the remainder is in AH
IDIV BL	AX is divided by BL; the signed quotient is in AL and the remainder is in AH
DIV BYTE PTR[BP]	AX is divided by the byte contents of the stack segment memory location addressed by BP; the unsigned quotient is in AL and the remainder is in AH

TABLE 5–12 16-bit division instructions.

Assembly Language	*Operation*
DIV CX	DX–AX is divided by CX; the unsigned quotient is in AX and the remainder is in DX
IDIV SI	DX–AX is divided by SI; the signed quotient is in AX and the remainder is in DX
DIV NUMB	AX is divided by the contents of the data segment memory location NUMB; the unsigned quotient is in AX and the remainder is in DX

TABLE 5–13 32-bit division instructions.

Assembly Language	*Operation*
DIV ECX	EDX–EAX is divided by ECX; the unsigned quotient is in EAX and the remainder is in EDX
DIV DATA2	EDX–EAX is divided by the doubleword contents of data segment memory location DATA2; the unsigned quotient is in EAX and the remainder is in EDX
IDIV DWORD PTR[EDI]	EDX–EAX is divided by the doubleword contents of the data segment memory location addressed by EDI; the signed quotient is in EAX and the remainder is in EAX

TABLE 5–14 AND instructions.

Assembly Language	*Operation*
AND AL,BL	AL = AL AND BL
AND CX,DX	CX = CX AND DX
AND ECX,EDI	ECX = ECX AND EDI
AND CL,33H	CL = CL AND 33H
AND DI,4FFFH	DI = DI AND 4FFFH
AND ESI,34H	ESI = ESI AND 00000034H
AND AX,[DI]	AX is ANDed with the word contents of the data segment memory location addressed by DI
AND ARRAY[SI],AL	The byte contents of the data segment memory location addressed by the sum of ARRAY plus SI is ANDed with AL; the result moves to memory
AND [EAX],CL	CL is ANDed with the byte contents of the data segment memory location addressed by EAX; the result moves to memory

TABLE 5–15 OR instructions.

Assembly Language	*Operation*
OR AH,BL	AH = AH OR BL
OR SI,DX	SI = SI OR DX
OR EAX,EBX	EAX = EAX OR EBX
OR DH,0A3H	DH = DH OR A3H
OR SP,990DH	SP = SP OR 990DH
OR EBP,10	EBP = EBP OR 0000000AH
OR DX,[BX]	DX is ORed with the word contents of the data segment memory location addressed by BX
OR DATES[DI + 2],AL	The byte contents of the data segment memory location addressed by the sum of DATES, DI, and 2 are ORed with AL

TABLE 5–17 TEST instructions.

Assembly Language	*Operation*
TEST DL,DH	DL is ANDed with DH
TEST CX,BX	CX is ANDed with BX
TEST EDX,ECX	EDX is ANDed with ECX
TEST AH,4	AH is ANDed with 4
TEST EAX,256	EAX is ANDed with 256

TABLE 5–18 Bit test instructions.

Assembly Language	*Operation*
BT	Tests a bit in the destination operand specified by the source operand
BTC	Tests and complements a bit in the destination operand specified by the source operand
BTR	Tests and resets a bit in the destination operand specified by the source operand
BTS	Tests and sets a bit in the destination operand specified by the source operand

TABLE 5–19 NOT and NEG instructions.

Assembly Language	*Operation*
NOT CH	CH is one's complemented
NEG CH	CH is two's complemented
NEG AX	AX is two's complemented
NOT EBX	EBX is one's complemented
NEG ECX	ECX is two's complemented
NOT TEMP	The contents of the data segment memory location TEMP is one's complemented
NOT BYTE PTR[BX]	The byte contents of the data segment memory location addressed by BX is one's complemented

TABLE 5–21 Rotate instructions.

Assembly Language	*Operation*
ROL SI,14	SI rotates left 14 places
RCL BL,6	BL rotates left through carry 6 places
ROL ECX,18	ECX rotates left 18 places
RCR AH,CL	AH rotates right through carry the number of places specified by CL
ROR WORD PTR[BP],2	The word contents of the stack segment memory location addressed by BP rotate right 2 places

FIGURE 5–10 The rotate instructions showing the direction and operation of each rotate.

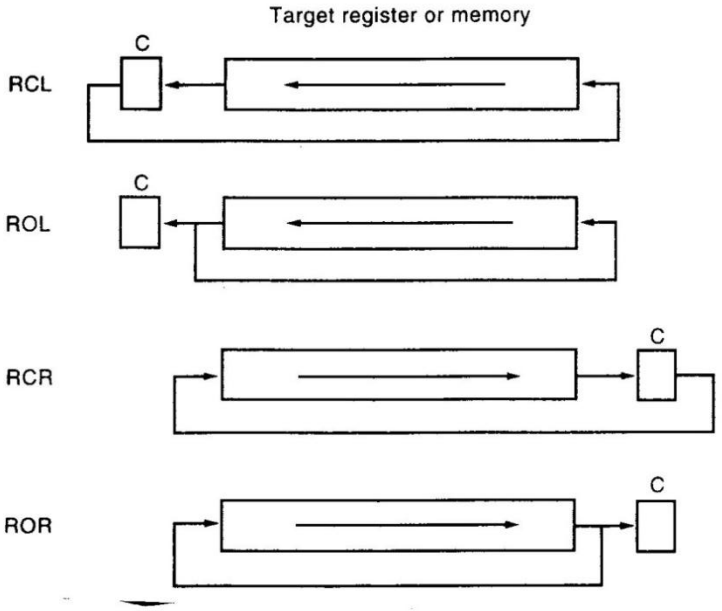
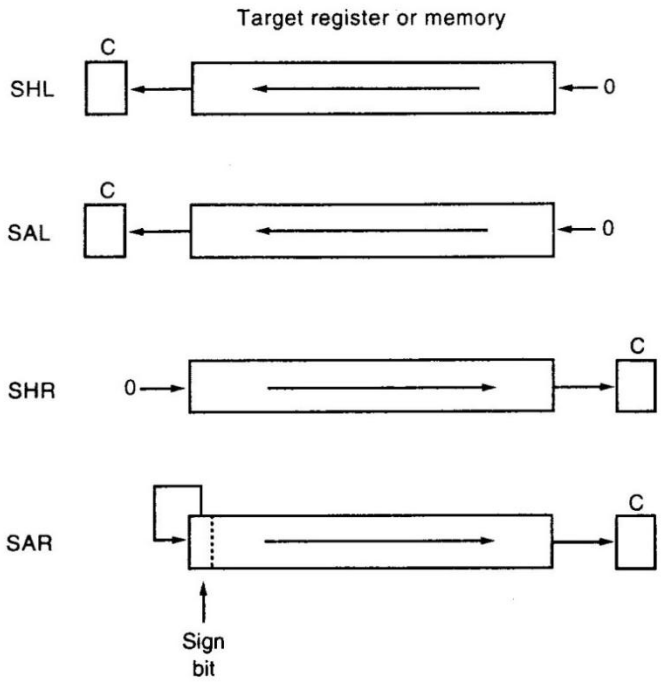


TABLE 5–20 Shift instructions.

Assembly Language	*Operation*
SHL AX,1	AX is logically shifted left 1 place
SHR BX,12	BX is logically shifted right 12 places
SHR ECX,10	ECX is logically shifted right 10 places
SAL DATA1,CL	The contents of the data segment memory location DATA1 is arithmetically shifted left the number of places specified by CL
SAR SI,2	SI is arithmetically shifted right 2 places
SAR EDX,14	EDX is arithmetically shifted right 14 places

FIGURE 5–9 The shift instructions showing the operation and direction of the shift.



Intel 8086 Hardware

- Similar to 8088 but has 16-bit data bus instead of 8-bit
- Power Supply Requirements
  - Requires 5V with 10% tolerance
  - Maximum supply current of 360 mA
  - Operates between 32 to 180 degrees F
  - CMOS version uses only 10mA and operates in -40 to 225 degrees F
- Noise Immunity
  - Difference between logic 0 output and logic 0 input voltages (= 0.35V)

• Fan Out

○ Maximum logic gate load at the output (=10)

TABLE 9–5 Bus cycle status (8088) using $\overline{SS0}$.

$IO/\overline{M}$	$DT/\overline{R}$	$\overline{SS0}$	*Function*
0	0	0	Interrupt acknowledge
0	0	1	Memory read
0	1	0	Memory write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	I/O read
1	1	0	I/O write
1	1	1	Passive

TABLE 9–6 Bus control functions generated by the bus controller (8288) using $\overline{S2}$, $\overline{S1}$, and $\overline{S0}$.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	*Function*
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

TABLE 9–7 Queue status bits.

QS1	*QS0*	*Function*
0	0	Queue is idle
0	1	First byte of opcode
1	0	Queue is empty
1	1	Subsequent byte of opcode

TABLE 9–4 Function of status bits S3 and S4.

S4	*S3*	*Function*
0	0	Extra segment
0	1	Stack segment
1	0	Code or no segment
1	1	Data segment

TABLE 9–1 Input characteristics of the 8086 and 8088 microprocessors.

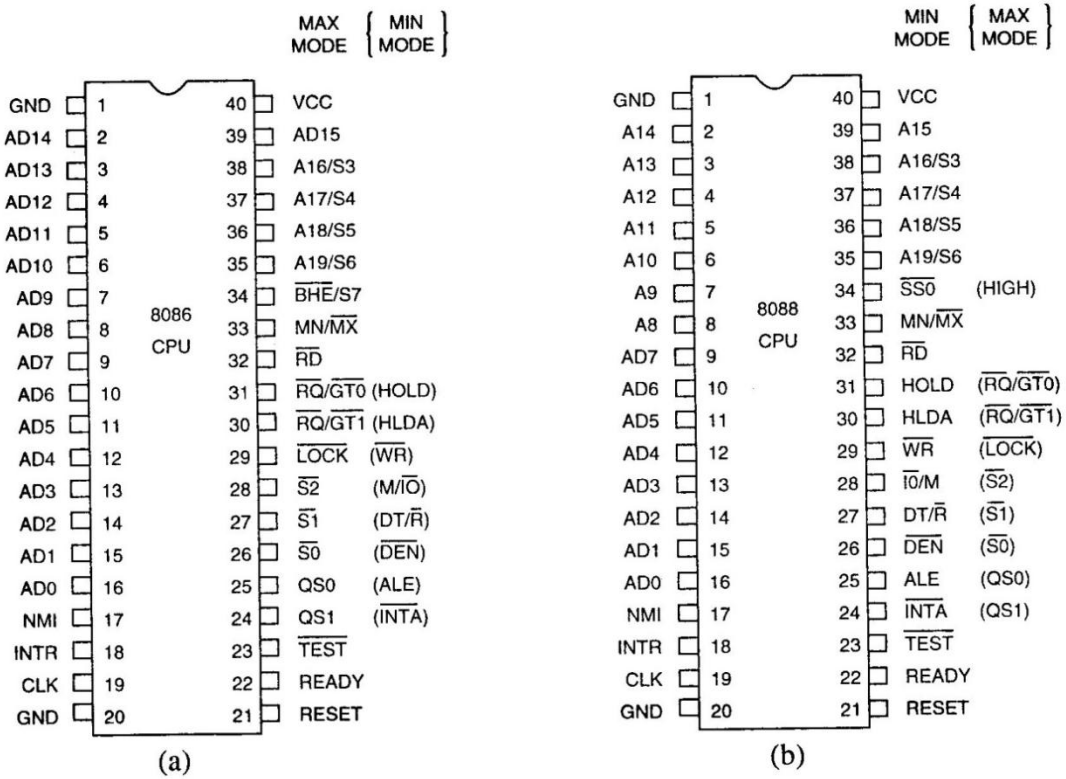
Logic Level	*Voltage*	*Current*
0	0.8 V maximum	±10 μA maximum
1	2.0 V minimum	±10 μA maximum

TABLE 9–3 Recommended fan-out from any 8086/8088 pin connection.

Family	*Sink Current*	*Source Current*	*Fan-out*
TTL (74)	–1.6 mA	40 μA	1
TTL (74LS)	–0.4 mA	20 μA	5
TTL (74S)	–2.0 mA	50 μA	1
TTL (74ALS)	–0.1 mA	20 μA	10
TTL (74AS)	–0.5 mA	25 μA	10
TTL (74F)	–0.5 mA	25 μA	10
CMOS (74HC)	–10 μA	10 μA	10
CMOS (CD4)	–10 μA	10 μA	10
NMOS	–10 μ	10 μA	10

Pin-Outs and Pin Functions

FIGURE 9–1 (a) The pin-out of the 8086 microprocessor; (b) the pin-out of the 8088 microprocessor.



- AD15-AD0: multiplexed address/data pins
- A19/S6-A16/S3: multiplexed address/status pins

  S6 always remains 0, S5 is related to Flags, S4 and S3 show which segment in memory is accessed
- RD : Read Signal (0 when receiving data from memory or I/O)
- READY: for inserting wait states in µP timing (0)
- INTR: for requesting hardware interrupt if IF=1
- TEST: works with WAIT instruction
- NMI: Non-maskable interrupt (regardless of IF bit)
- Reset: causes reset and disables interrupts

- CLK: clock input pin of µP with 1/3 duty cycle
- Vcc: power supply input
- GND: two ground connections
- MN/MX: minimum/maximum operation mode
- BHE/S7: bus high enable used to enable D15-D8

- Minimum Mode Pins
  - IO/M: selects memory or I/O for address bus
  - WR: indicates µP is outputting data
  - INTA: interrupt acknowledge responds to INTR input

- ALE: address latch enable shows µP bus contains address
- DT/R: data transmit/receive shows that µP is transmitting (1) or receiving data (0)
- DEN: data bus enable activates external data bus buffers
- HOLD: requests direct memory address (DMA) if 1; another bus master wants to control the bus
- HOLA: hold acknowledge indicates the µP is in hold state and all buses are floating
- SS0: used with IO/M and DT/R to detect function of current bus cycle

- Maximum Mode Pins for use with a co-processor
  - S2, S1, S0: status bits indicate function of current bus cycle

- R0/GT0 and R0/GT1: request/grant bi-directional pins request and grant DMA
- LOCK: lock output locks peripherals off the system
- QS1 and QS0: queue status pins indicate the internal instruction queue for numeric co-processor

Clock Generator

- Provides 5 MHz for µP and 2.5 MHz for peripherals
- Uses an external clock for 15 MHz crystal
- Provides a system reset signal

FIGURE 9–2 The pin-out of the 822844A clock generator.

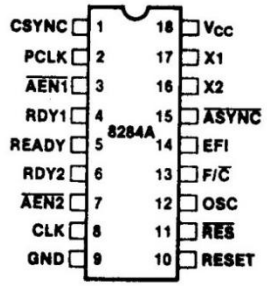
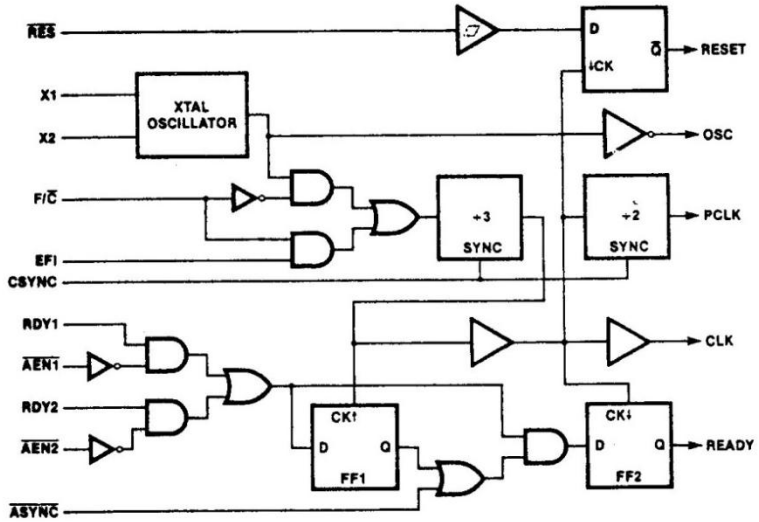


FIGURE 9–3 The internal block diagram of the 8284A clock generator.



Bus Buffering and Latching

- Multiplexing reduces number of pins
- Demultiplexing required to have stable addresses for memory or I/O
- Transparent latch is like a wire when enabled and hold previous state when disabled
- Buffers used to drive high-capacitance loads
- Data bus uses bi-directional buffers

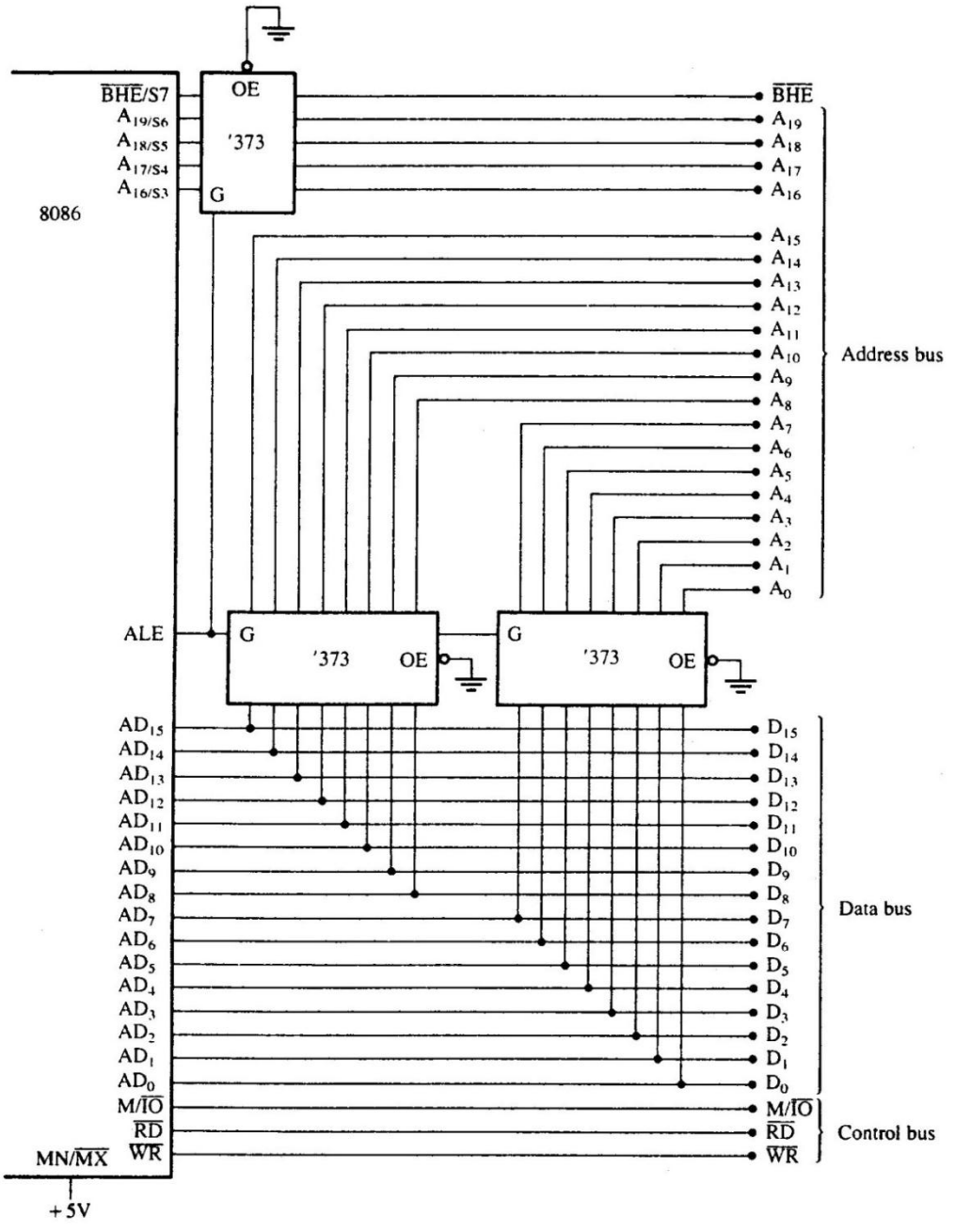


FIGURE 9–6 The 8086 microprocessor shown with a demultiplexed address bus. This is the model used to build many 8086-based systems.

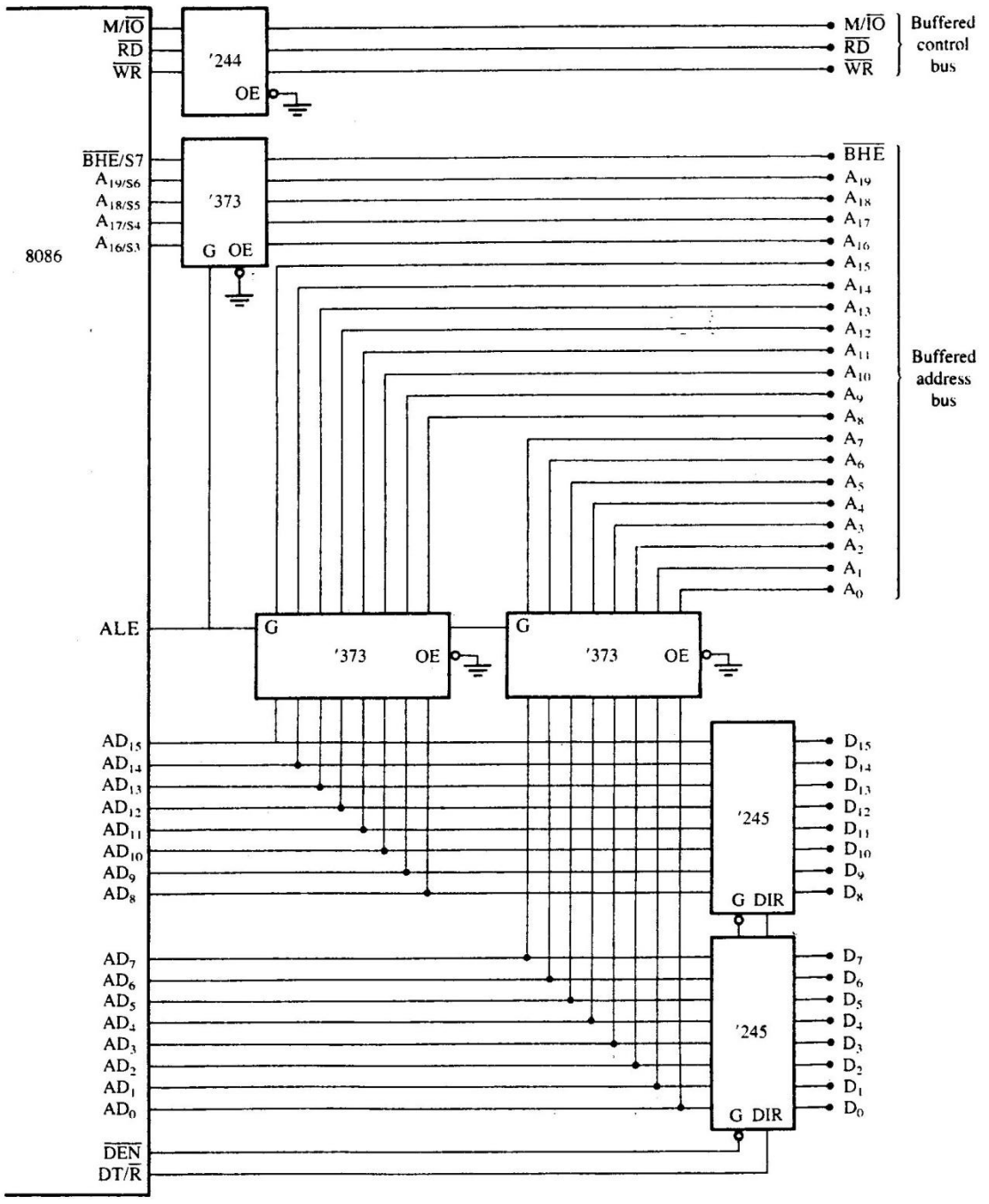


FIGURE 9–8 A fully buffered 8086 microprocessor.

Bus Timing

- µP uses memory or I/O in periods called bus cycle
- Each bus cycle equals 4 system-clocking period (T state)
- In T1 the address is placed, ALE, DT/R, and IO/M are activated
- In case of write, data appears on data bus in T2
- READY is sampled at the end of T2, if low then T3 is wait state
- In T4 all signals are deactivated and prepared for next cycle

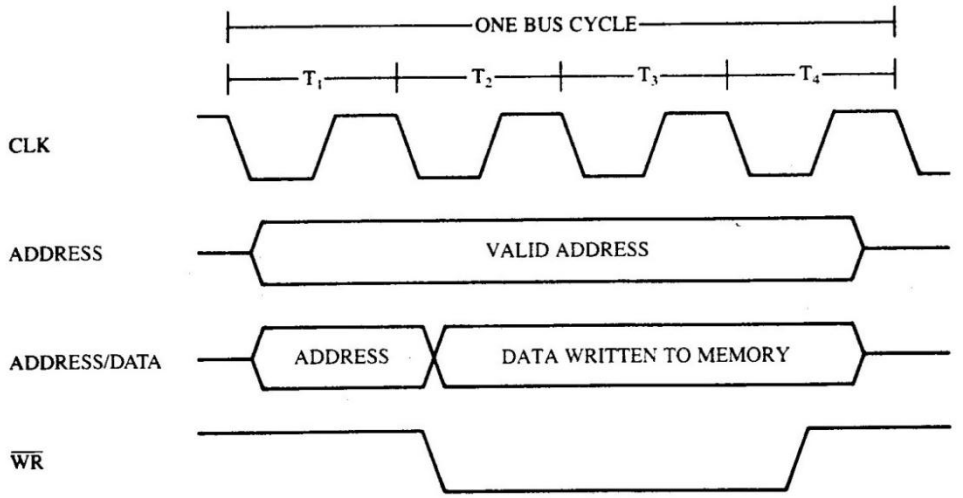


FIGURE 9–9 Simplified 8086/8088 write bus cycle.

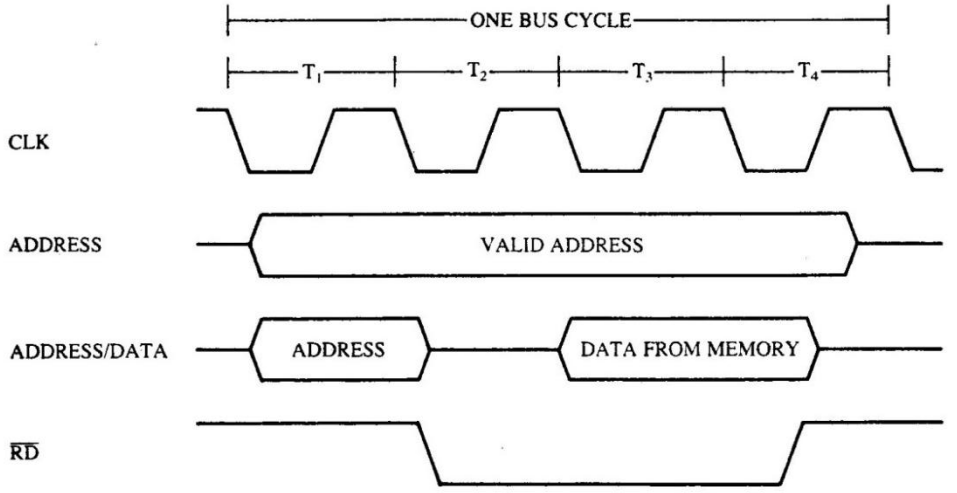


FIGURE 9–10 Simplified 8086/8088 read bus cycle.

- Ready and Wait State
  - READY input to µP causes wait state for slower access
  - Wait states appear between T2 and T3 to lengthen bus cycle
  - Memory access time is the period between when address appears on bus until data is sampled by µP
  - For 8086, at 5 MHz, each state is 200 ns; normal access times are 460 ns
  - READY is sampled at the end of T2 and again middle of Tw
  - Clock generator is used to synchronize READY signal

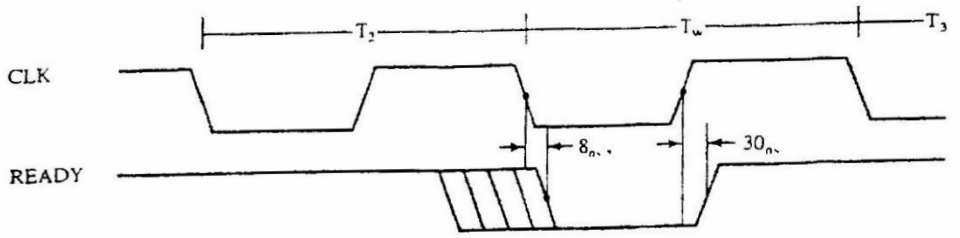


FIGURE 9–14 8086/8088 READY input timing.

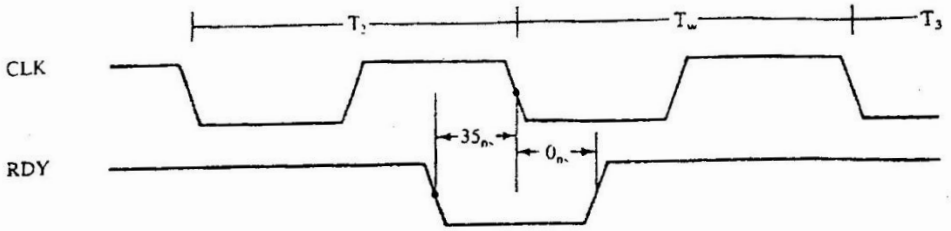
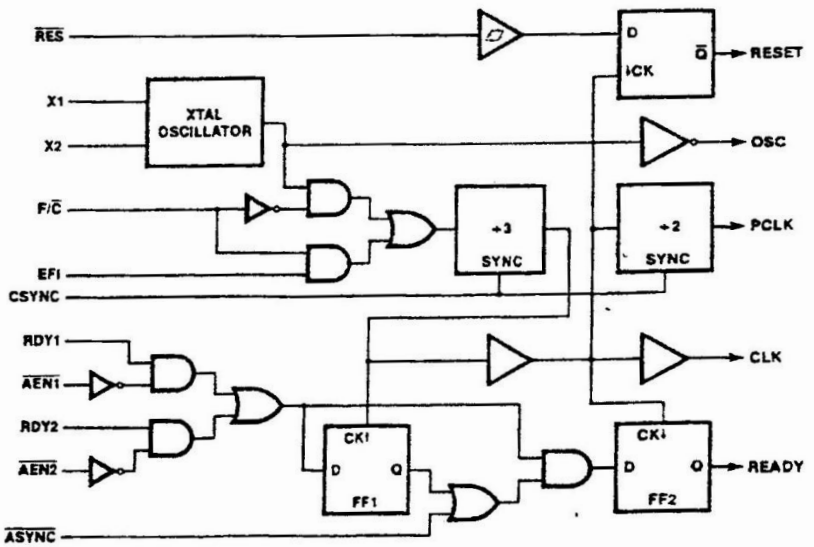


FIGURE 9–15 8284A RDY input timing.

FIGURE 9–16 The internal block diagram of the 8284A clock generator. (Courtesy of Intel Corporation).



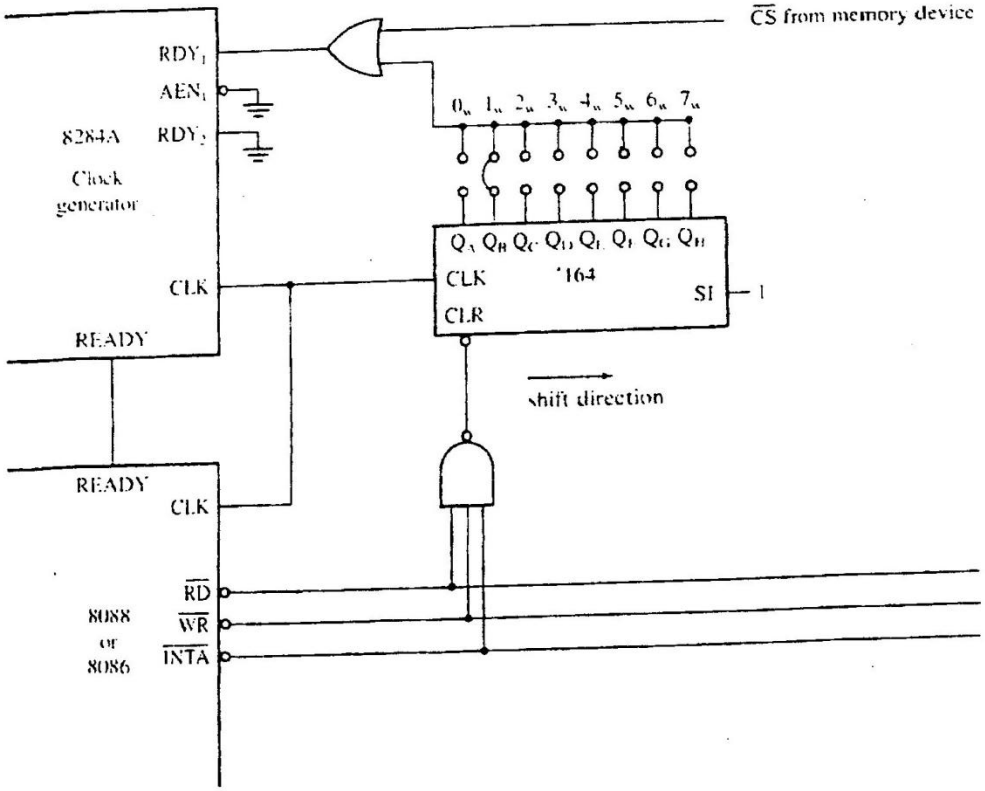


FIGURE 9–17 A circuit that will cause between 0 and 7 wait states.

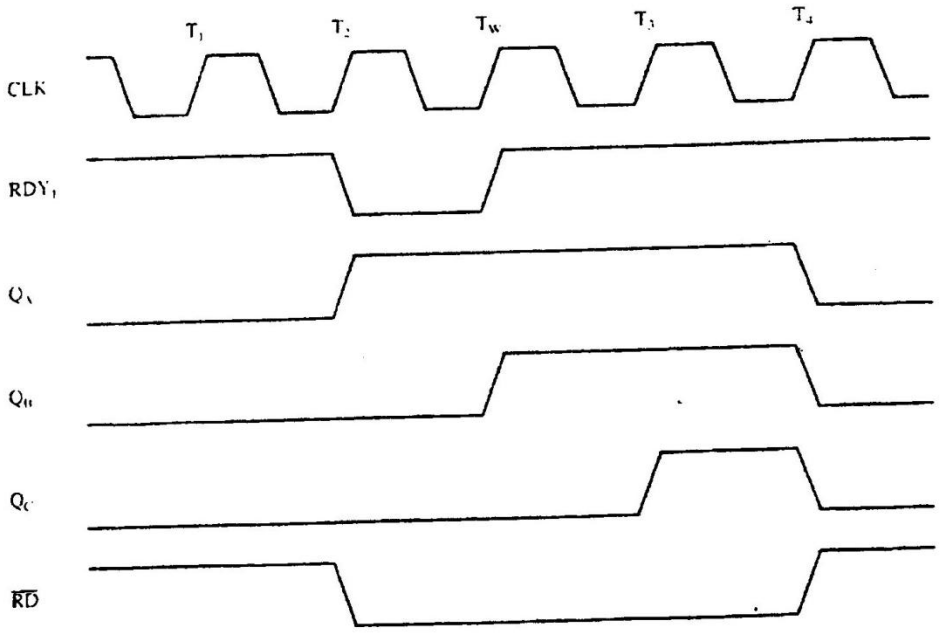


FIGURE 9–18 Wait state generation timing of the circuit of Figure 9–17.

Memory Interface

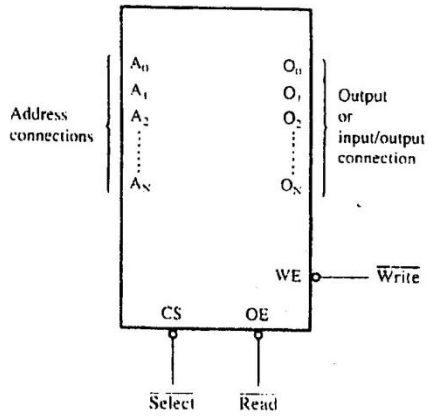
- Two types: ROM and RAM, with 4 types of connection lines
- Address Connection: labeled A0 to An for n+1 lnes
  - Number of location = 2^{n+1}, e.g. 10 pins means 1 K
- Data Connections: outputs (Os) or input/output (Ds)
  - A byte-wide memory stores 8 bits per memory location. Memories often referred to as locations times bits per location
  - E.g. 16K x 1 memory has 16K 1-bit locations
- Selection Connections
  - Enables memory like chip select (CS), or select (S) pins in RAM and chip enable (CE) in ROM

- Control Connections
  - One or more pins for operation control

    ROM has only one called output enable (OE) or gate (G) which enable or disable tri-state output buffers;

    RAM sometimes has one: (R/W) which enables read/write, and sometimes two: (WE or W) for enabling writing and (OE) for enabling reading

FIGURE 10–1 A pseudo-memory component illustrating the address, data, and control connections.



ROM

- Programmed during manufacturing; Data is permanent, so called nonvolatile memory
- Programmable ROM (PROM)
  - Programmed in-field by burning NI-chrom or silicon oxide fuses
- Erasable Programmable ROM (EPROM)
  - Programmed in-field with EPROM programmer; erasable if exposed to high-intensity ultraviolet light
- Electrically Erasable Programmable ROM (EEPROM)
  - Erasable in system but need more time than normal RAM

also called read-mostly memory (RMM), flash memory, electrically alterable ROM, and nonvolatile RAM (NVRAM)

  - Flash memory stores system setup information

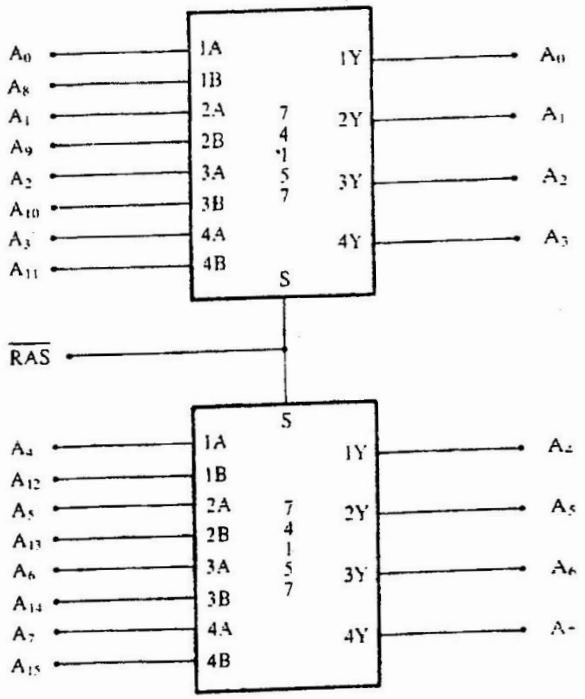
Static RAM (SRAM)

- Retains data as long as DC power applied (volatile)
- Used for cache memory because of fast access

Dynamic RAM (DRAM)

- Retains data on integrated capacitances
- Needs to be refreshed every 2 to 4 ms
- Much larger capacity than SRAM

FIGURE 10–9 Address multiplexer for the TMS4464 DRAM.



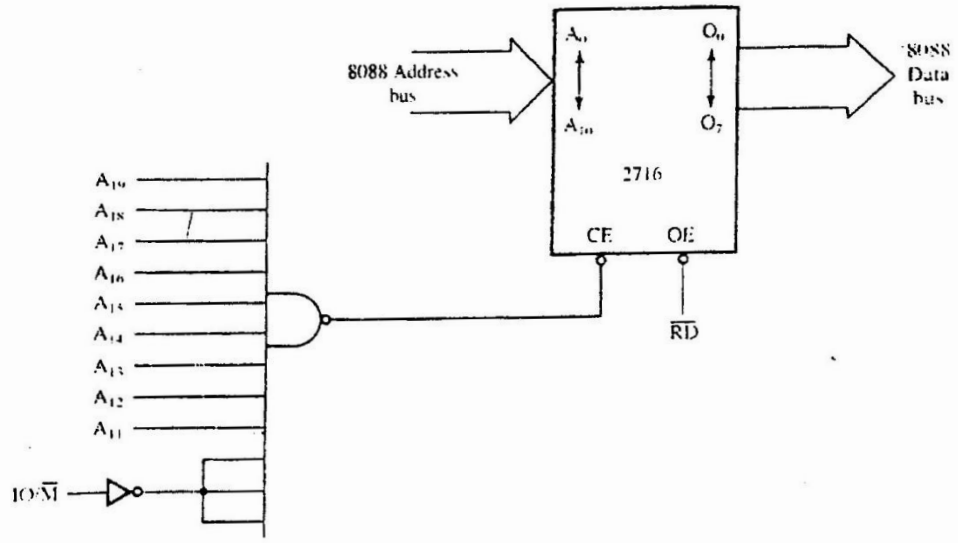


FIGURE 10–13 A simple NAND gate decoder used to select a 2716 EPROM memory component for memory locations FF800H–FFFFFH.

- Refresh is done by reading and rewriting data
  - RAS selects a row for refreshing while DRAM is operational. Called hidden refresh, transparent refresh, or cycle stealing

- Extended Data Output (EDO): DRAM with output latches
  - Latch holds next data, this 15% to 25% faster
  - Refresh is done by writing into these latches

- Synchronous DRAM (SDRAM) used with newer systems
  - SDRAM read four 64-bit numbers in one burst
  - First number takes 3 to 4 clock cycles, rest only 1
  - Faster than both normal DRAM and EDO

256K X 1 DRAM needs refreshing every 4 ms.

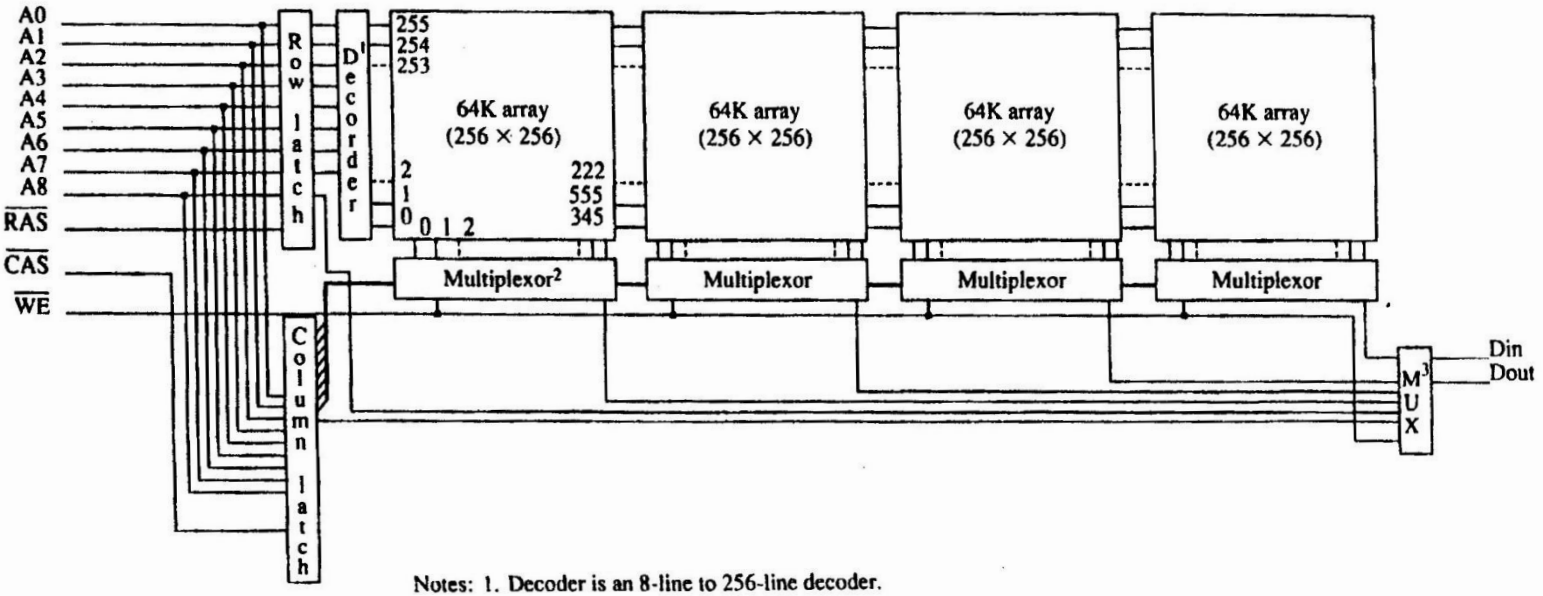
In one refresh cycle one row is refreshed; has 256 rows.

⇒ 4 ms ÷ 256 = 15.6 μs i.e. refresh cycle needed every 15.6 μs.

Each read/write cycle is 4 T states = 800 ns.

15.6 μs ÷ 800 ns = 19 i.e. one refresh per 19 memory access.

≃ 5% lost of Computer Time + Energy loss!



Notes: 1. Decoder is an 8-line to 256-line decoder.
2. Multiplexor is 256 to 1 line.
3. Multiplexor is 4 to 1 line.

FIGURE 10–39 The internal structure of a 256K x 1 DRAM. Note that each of the internal 256 words are 1025-bits wide.

Address Decoding

- Usually more than one memory chip is connected µP
- Decoding allocates each chip to a part of memory map
- Types of decoders
  - NAND gate: expensive because multi-input NAND gates are required for each memory device
  - Decoder chips: more commonly used than NAND, like 3-to-8
  - PLD: programmable logic device; used today

    1-PROM: economical because of large number of inputs

    2-PLA: programmable logic arrays; has replaced PROM because of higher flexibility

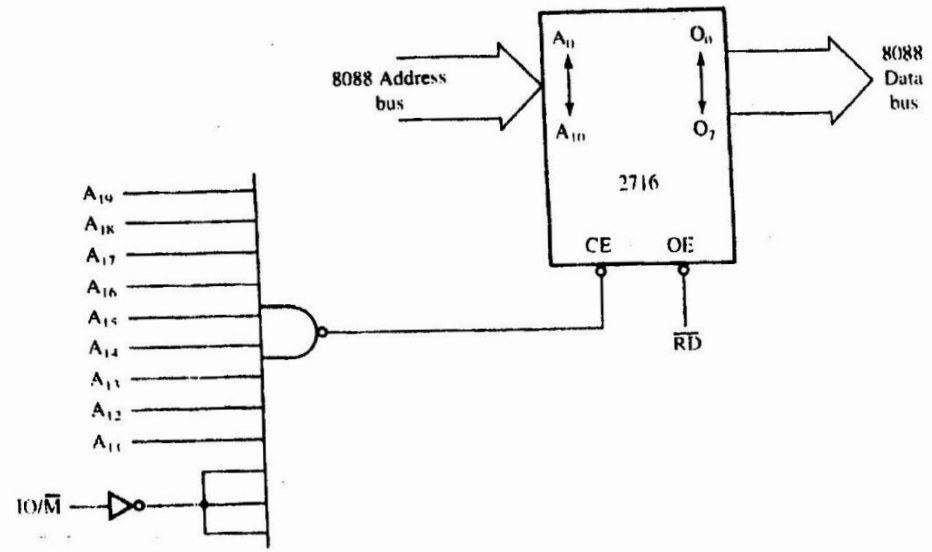


FIGURE 10–13 A simple NAND gate decoder used to select a 2716 EPROM memory component for memory locations FF800H–FFFFFH.

8088 has 20 address lines → 2^{20} = 1 M locations

1111 1111 1XXX XXXX XXXX

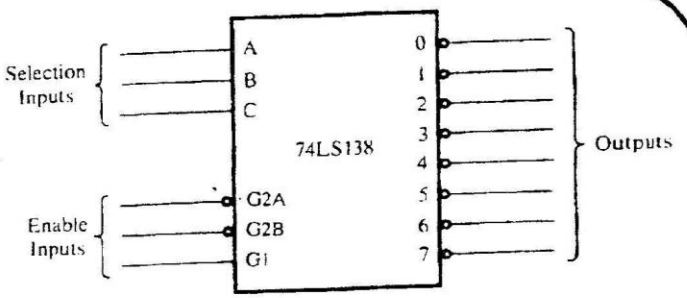
⇒ 1111 1111 1000 0000 0000 = $FF\,800_H$

To

1111 1111 1111 1111 1111 = $FFFFF_H$

2K byte of memory mapped to FF800 – FFFFF.

FIGURE 10–14 The 74LS138, 3-to-8 line decoder and function table.



whole addren range used is F0000 to FFFFF

i.e. 64 K.

each slot is

2^{13} = 8 K

addren for 1st device:

F0000 to F1FFFH

Inputs						Outputs							
Enable			Select										
$\overline{G2A}$	$\overline{G2B}$	G1	C	B	A	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

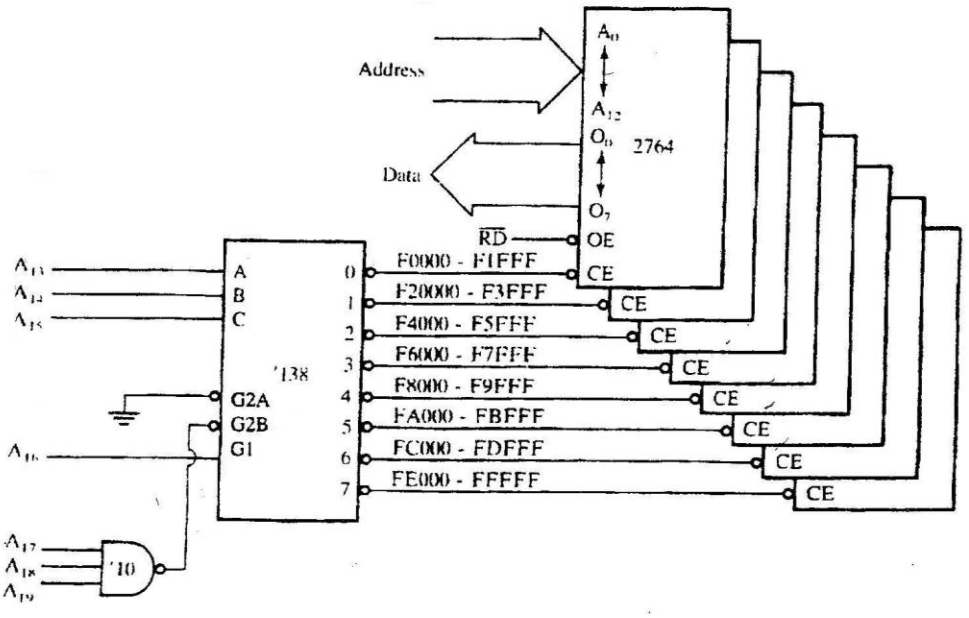


FIGURE 10–15 A circuit that uses eight 2764 EPROMs for a 64K × 8 section of memory in an 8088 microprocessor-based system. The addresses selected in this circuit are F0000H–FFFFFH.

PROM location's are all logic 1s manufactured, then 0s are programmed.

Here only 8 of 512 locations are programmed then.

Example shows same mapping as previan example.

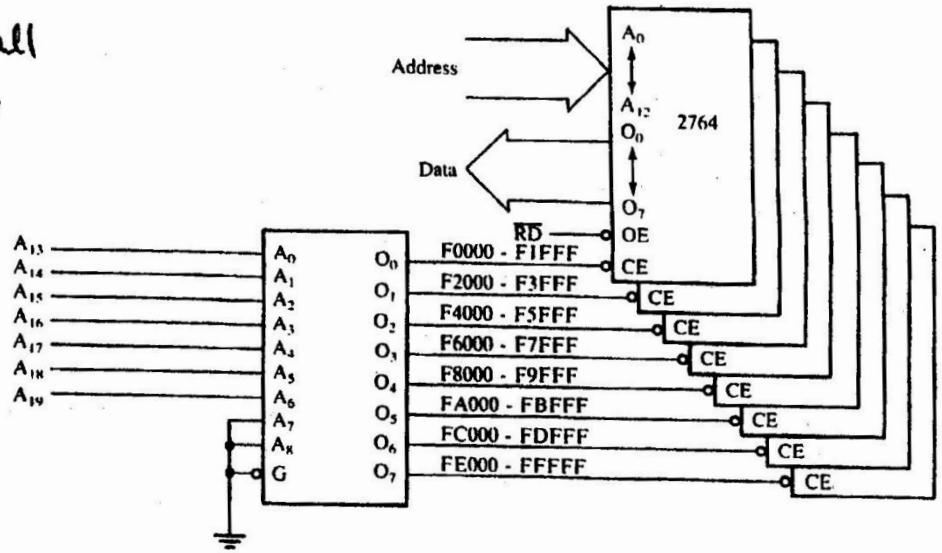


FIGURE 10–17 A memory system using the TPB28L42, 512 × 8 PROM as an address.

64KB F0000 - FFFFF

TABLE 10–1 The 82S147 PROM programming pattern for the circuit of Figure 10–17.

	Inputs									*Outputs*							
~~OE~~	A8	A7	A6	A5	A4	A3	A2	A1	A0	O0	O1	O2	O3	O4	O5	O6	O7
0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1
0	0	0	1	1	1	1	0	0	1	1	0	1	1	1	1	1	1
0	0	0	1	1	1	1	0	1	0	1	1	0	1	1	1	1	1
0	0	0	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1
0	0	0	1	1	1	1	1	0	0	1	1	1	1	0	1	1	1
0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
all	other	combinations								1	1	1	1	1	1	1	1

```
TITLE        Address Decoder
PATTERN      Test 1
REVISION     A
AUTHOR       Barry B. Brey
COMPANY      BreyCo
DATE         6/6/99
CHIP         DECODER1 PAL16L8

;pins 1   2   3   4   5   6   7   8  9  10
     A19 A18 A17 A16 A15 A14 A13 NC NC GND

;pins 11 12 13 14 15 16 17 18 19  20
      NC O8 O7 O6 O5 O4 O3 O2 O1 VCC

EQUATIONS

/O1 = A19 * A18 * A17 * A16 * /A15 * /A14 * /A13
/O2 = A19 * A18 * A17 * A16 * /A15 * /A14 * A13
/O3 = A19 * A18 * A17 * A16 * /A15 * A14 * /A13
/O4 = A19 * A18 * A17 * A16 * /A15 * A14 * A13
/O5 = A19 * A18 * A17 * A16 * A15 * /A14 * /A13
/O6 = A19 * A18 * A17 * A16 * A15 * /A14 * A13
/O7 = A19 * A18 * A17 * A16 * A15 * A14 * /A13
/O8 = A19 * A18 * A17 * A16 * A15 * A14 * A13
```

10 Fixed inputs
2 Fixed outputs
6 in/out pins

programming blows some fuses to connect some inputs to OR gates.

produces same results as previous two examples.

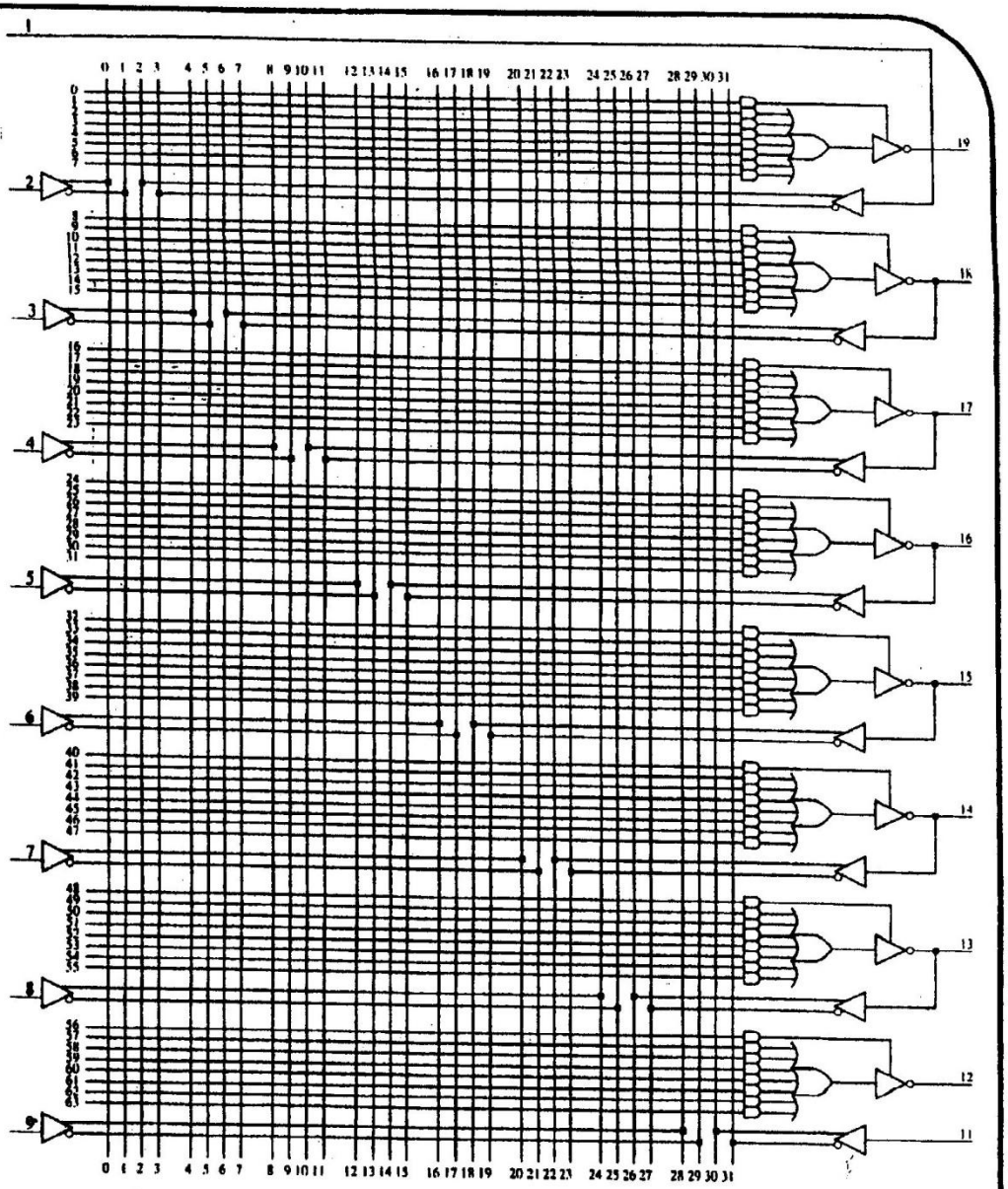


FIGURE 10–18 The PAL 16L8. (Copyright Advanced Micro Devices, Inc., 1988. Reprinted with permission of copyright owner. All rights reserved.)

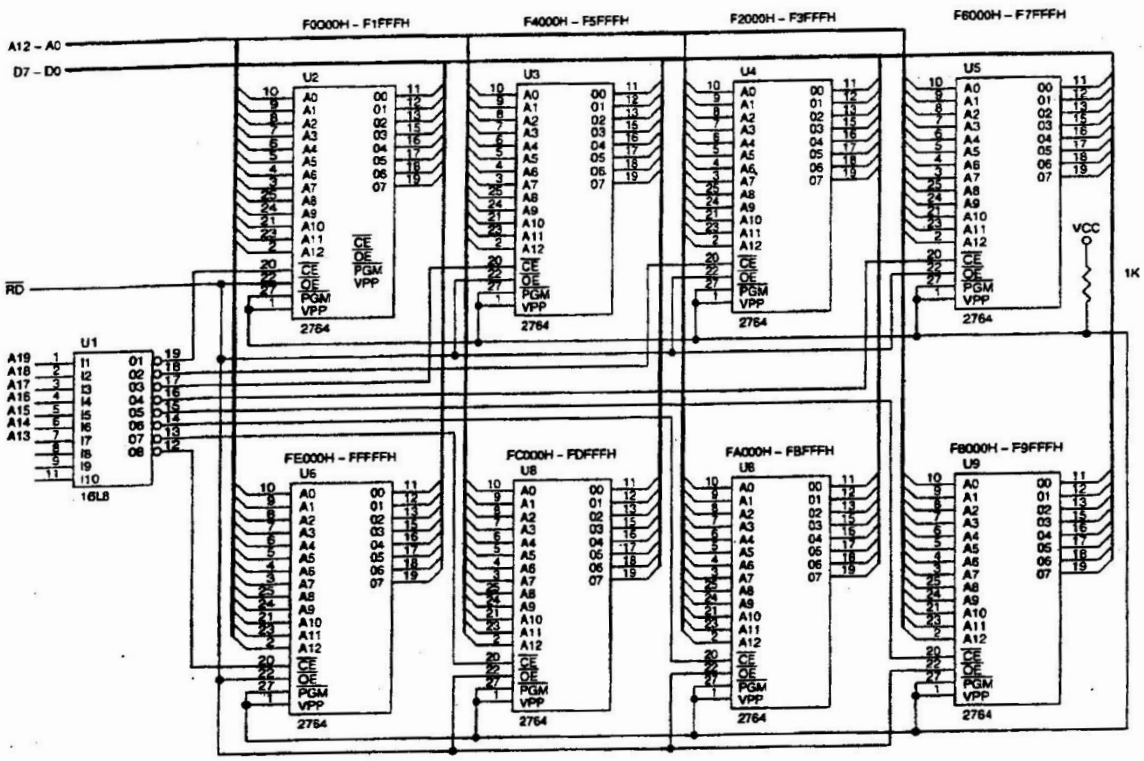
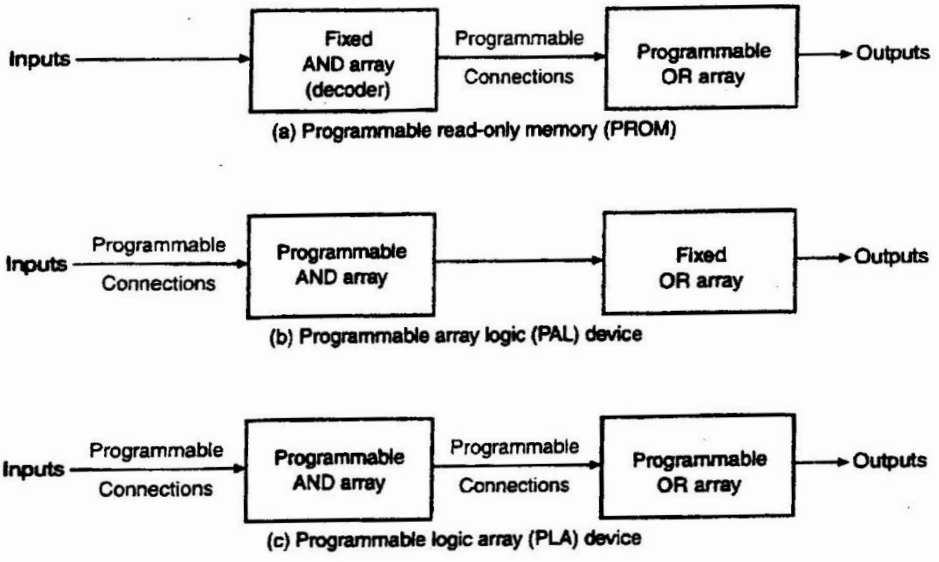


FIGURE 10–19 A PAL16L8 that decodes eight 2764 (8K x 8) memory devices.



□ FIGURE 6-19
Basic Configuration of Three PLDs

MANO & KIM "Logic and computer Design Fundamentals" Prentice Hall 1997

Interfacing Memory to 8-bit Data Bus μP

8088 @ 5 MHz has 460 ns access time allowed for memory.

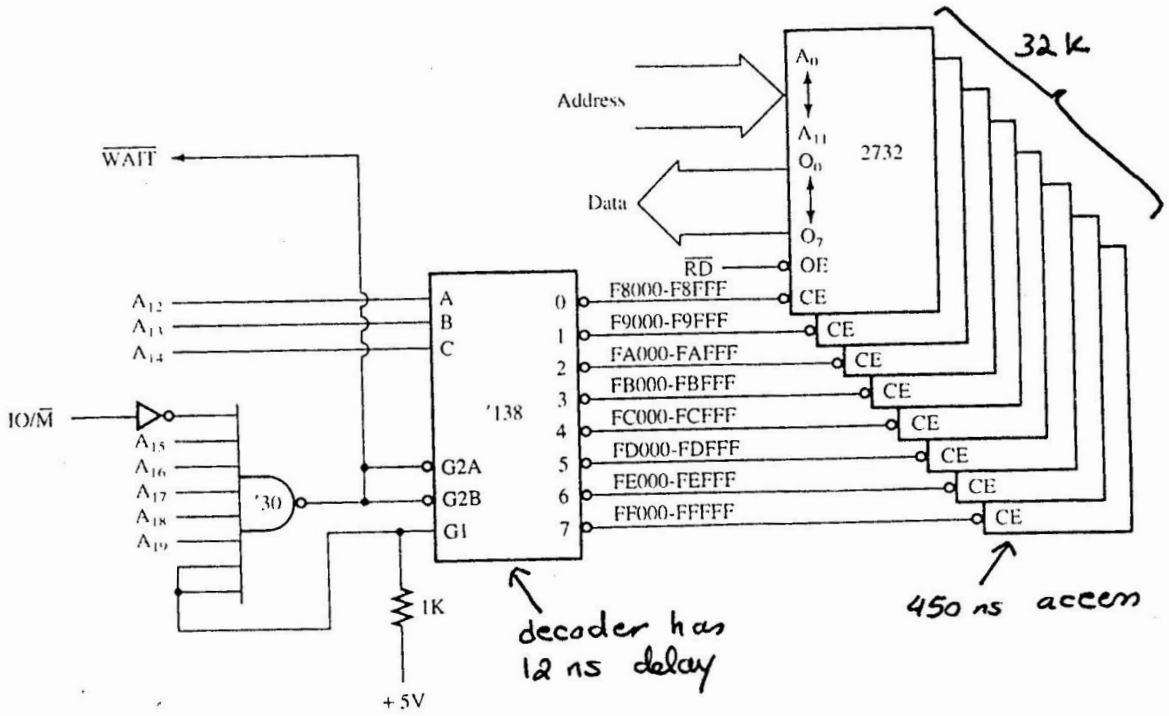


FIGURE 10–20 Eight 2732 EPROMs interfaced to the 8088 microprocessor. Note that the output of the NAND gate is used to cause a wait state whenever this section of the memory is selected.

Need to add wait state (200 ns), which increases access time to 660 ns.

RAM Interfacing

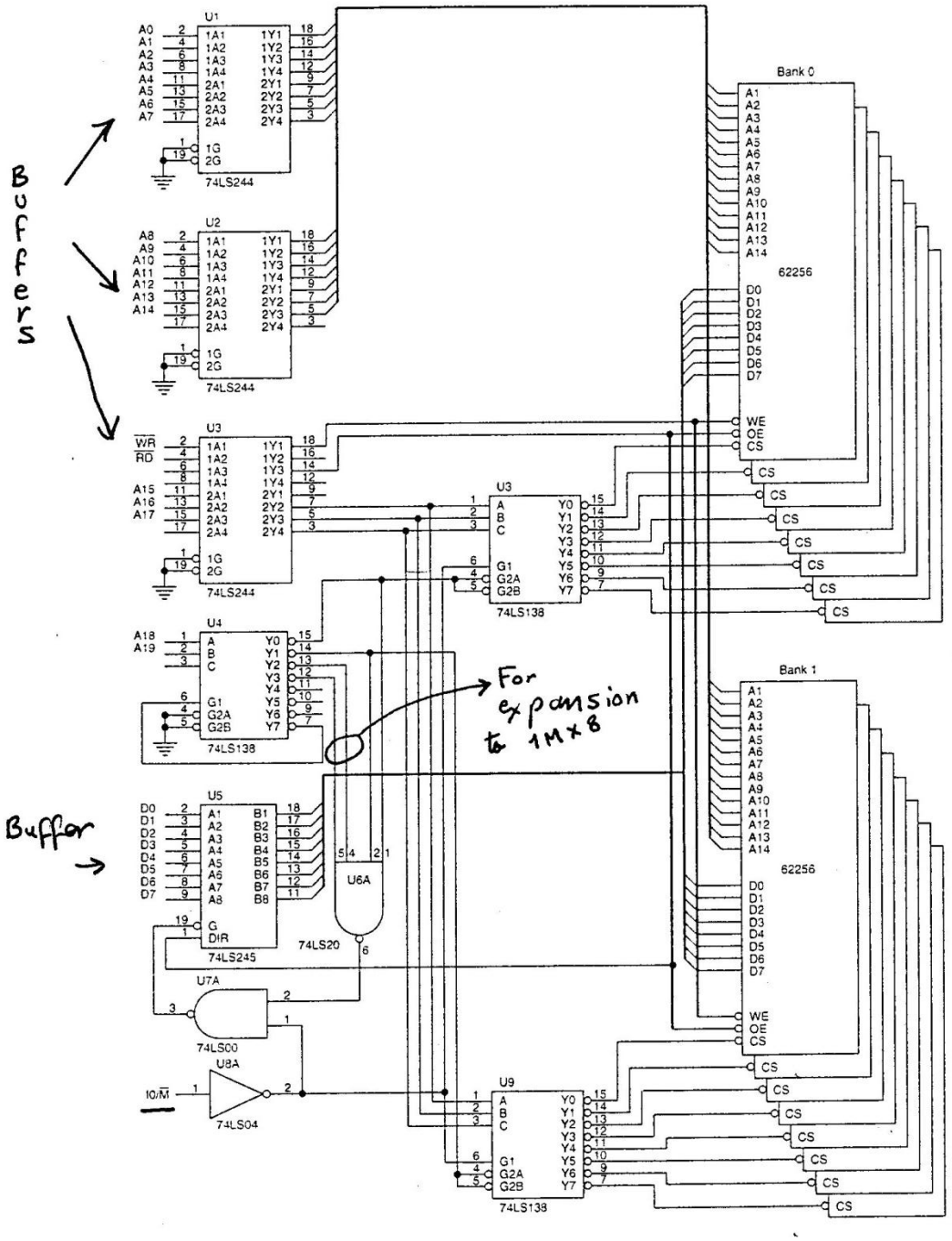


FIGURE 10–21 A 512K byte static memory system using 16 62255 SRAMs.

16 x (32K x 8) = 512K byte,
location 00000H to 7FFFFH

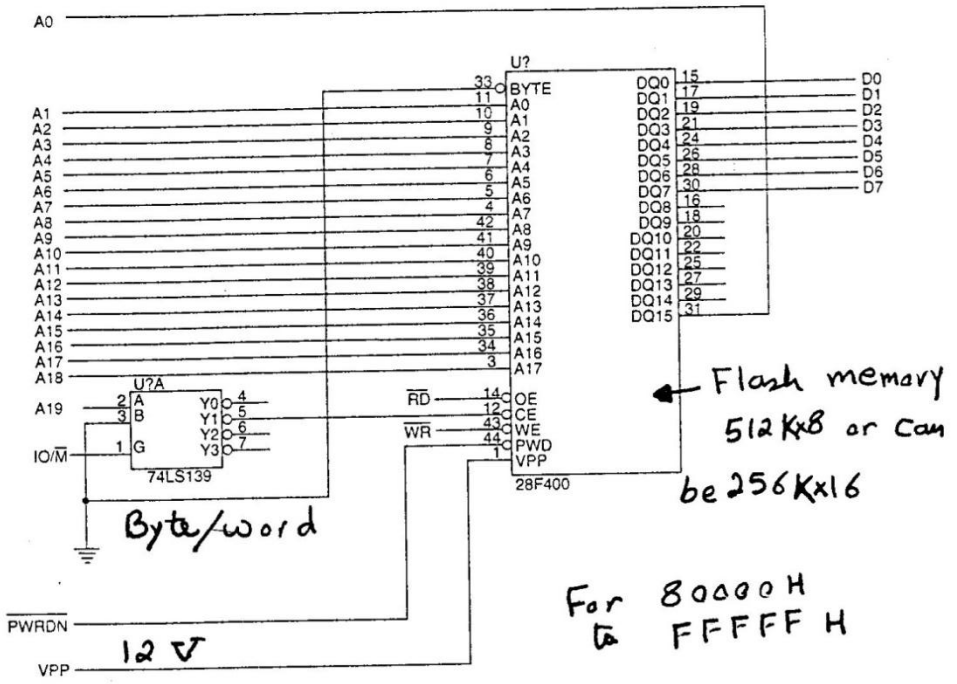


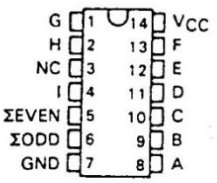
FIGURE 10–22 The 28F400 flash memory device interfaced to the 8088 microprocessor.

Flash memory requires 12 V to erase & write.
RAM works with 5 V. So, 5 to 12 V converter needed.

Parity is used for error detection in stored data

FIGURE 10–23 The pin-out and function table of the 74AS280 9-bit parity generator/detector. (Courtesy of Texas Instruments Incorporated.)

SN54AS280 . . . J PACKAGE
SN74AS280 . . . N PACKAGE
(TOP VIEW)



(a)

FUNCTION TABLE

NUMBER OF INPUTS A THRU I THAT ARE HIGH	OUTPUTS	
	Σ EVEN	Σ ODD
0,2,4,6,8	H	L
1,3,5,7,9	L	H

(b)

Error Detection and Correction

- Parity is number of 1s in data
- This detects single bit errors

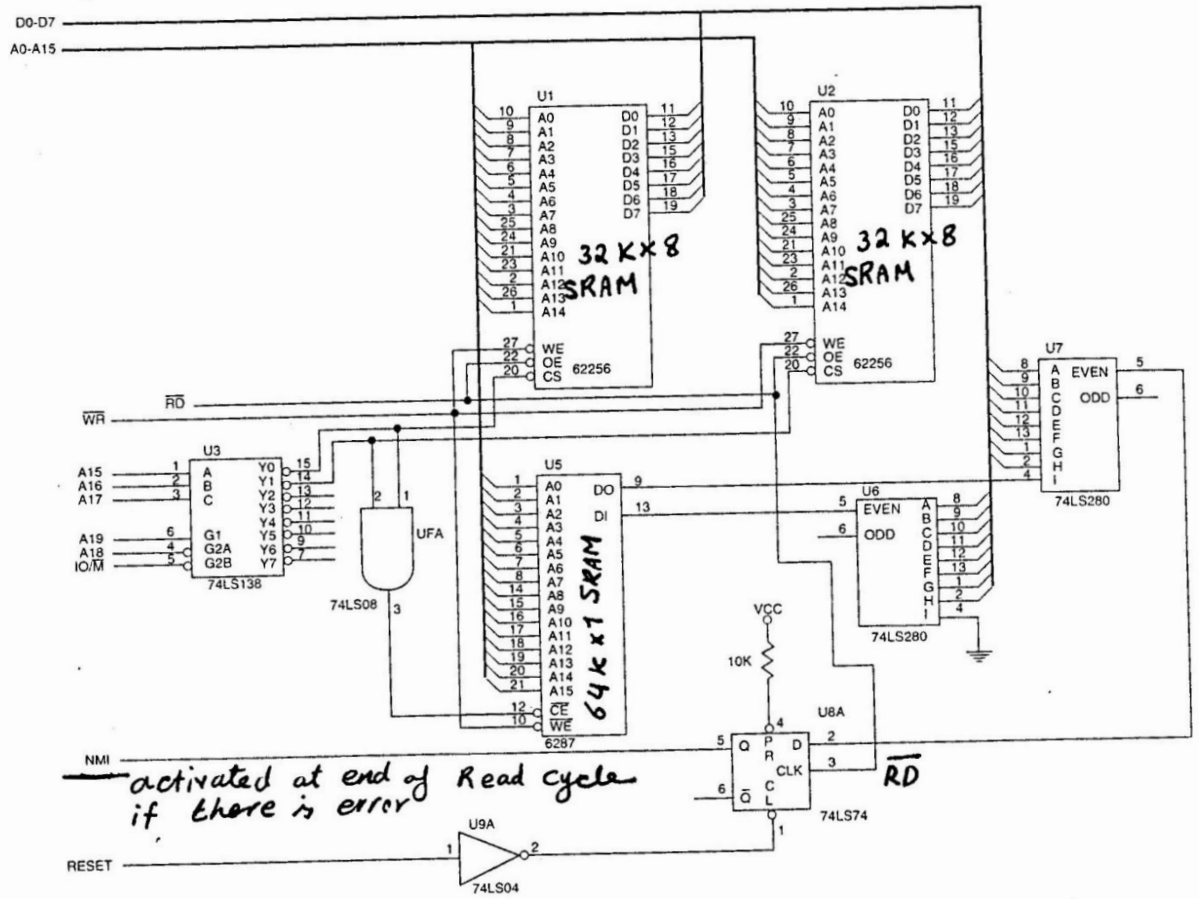


FIGURE 10–24 A 64K memory system that contains a parity error detection circuit.

pin assignments

J, N PACKAGES			
1	DEF	11	CB4
2	DB0	12	nc
3	DB1	13	CB3
4	DB2	14	CB2
5	DB3	15	CB1
6	DB4	16	CB0
7	DB5	17	S0
8	DB6	18	S1
9	DB7	19	SEF
10	GND	20	V_{CC}

(a)

Error detection and correction circuit

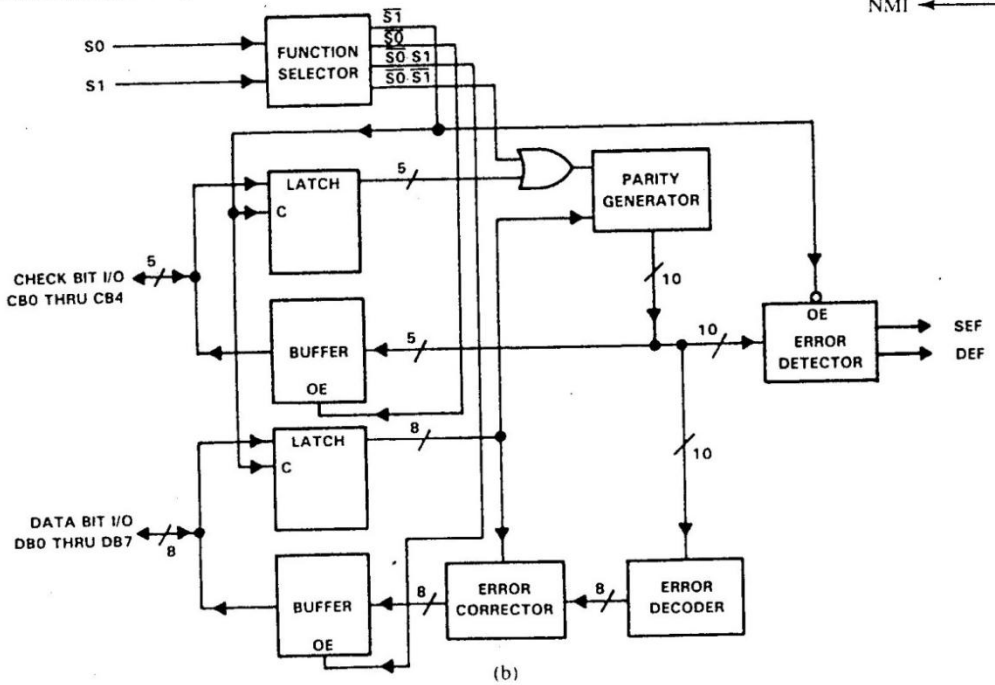
TABLE 10–2 Control bits S0 and S1.

S0	*S1*	*Function*	*SEF*	*DEF*
0	0	Write check word	0	0
0	1	Correct data word	*	*
1	0	Read data	0	0
1	1	Latch data	*	*

**Note:* These levels are determined by the type of error.

FIGURE 10–26 An error detection and correction circuit using the 74LS636.

functional block diagram



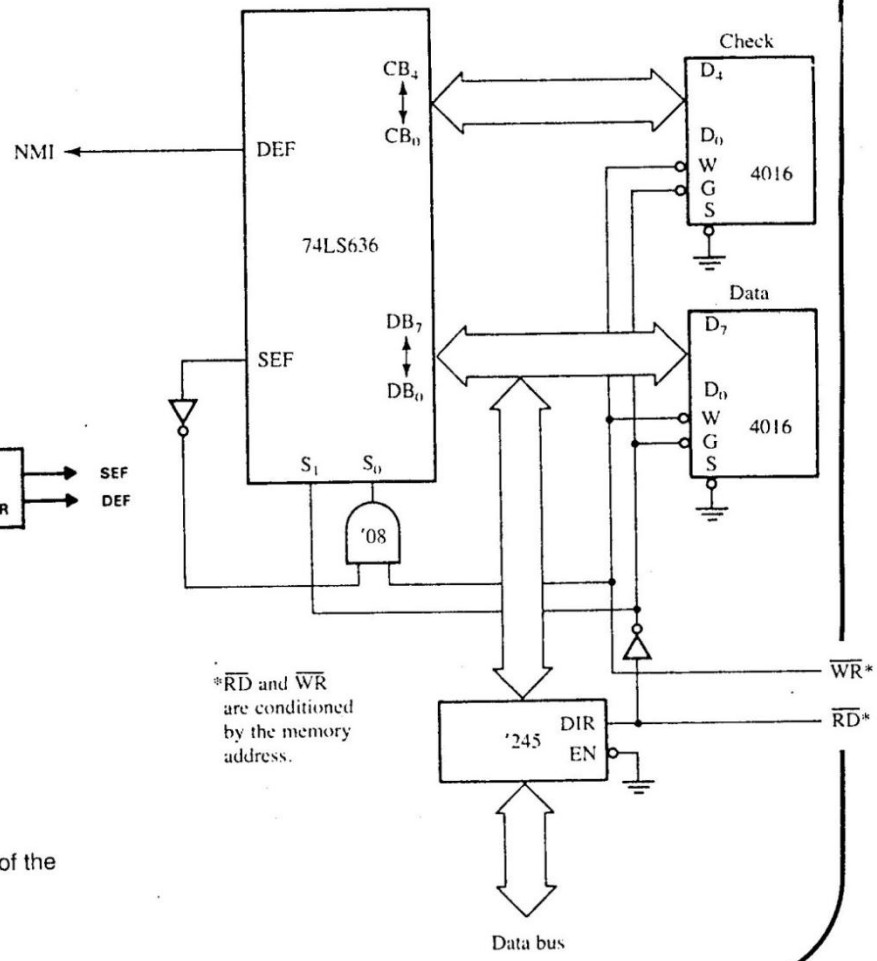
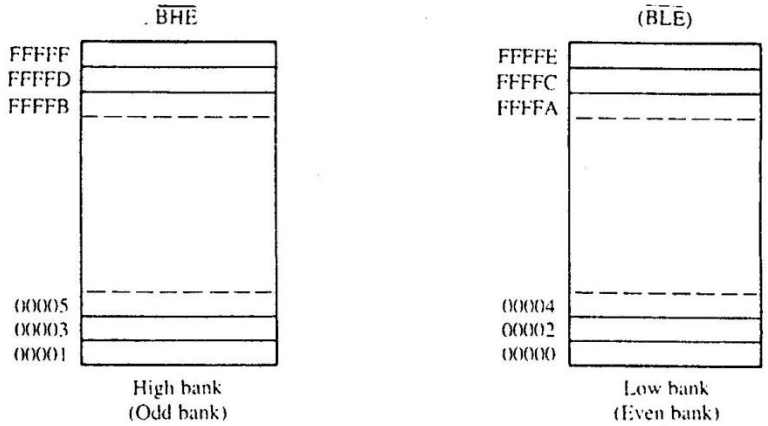


FIGURE 10–25 (a) The pin connections of the 74LS636. (b) The block diagram of the 74LS636. (Courtesy of Texas Instruments Incorportated.)

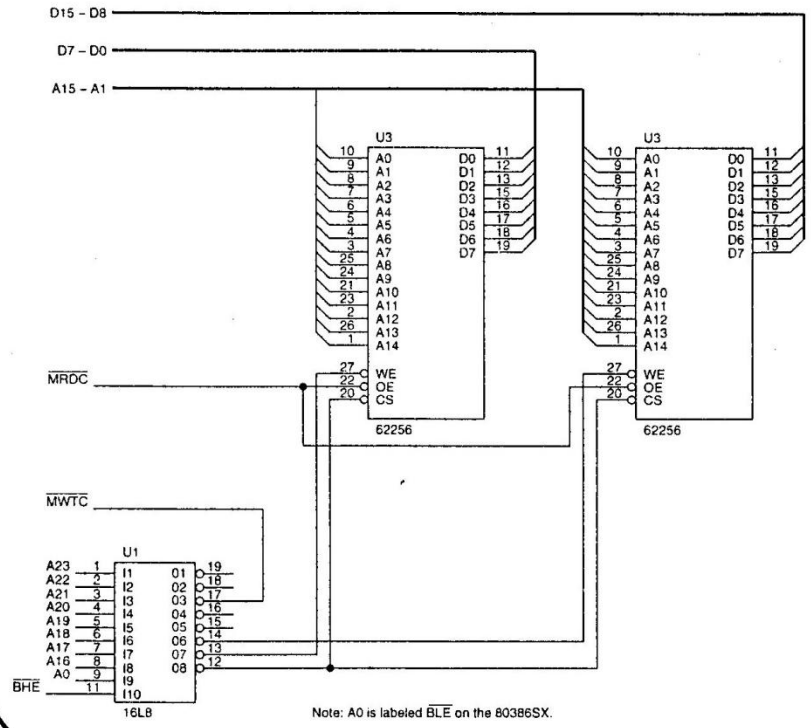
Interfacing Memory to 16-bit Data Bus μP

FIGURE 10–27 The high (odd) and low (even) 8-bit memory banks of the 8086/80286/80386SX microprocessors.



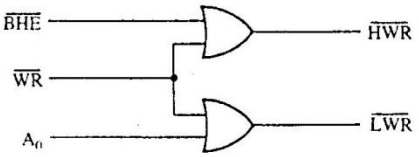
Note: A_0 is labeled $\overline{BLE}$ (Bus low enable) on the 80386SX.

FIGURE 10–30 A 16-bit memory decoder that places memory at locations 060000H–06FFFFH.



Note: A0 is labeled $\overline{BLE}$ on the 80386SX.

FIGURE 10–29 The memory bank write selection input signals: $\overline{HWR}$ (high bank write) and $\overline{LWR}$ (low bank write).



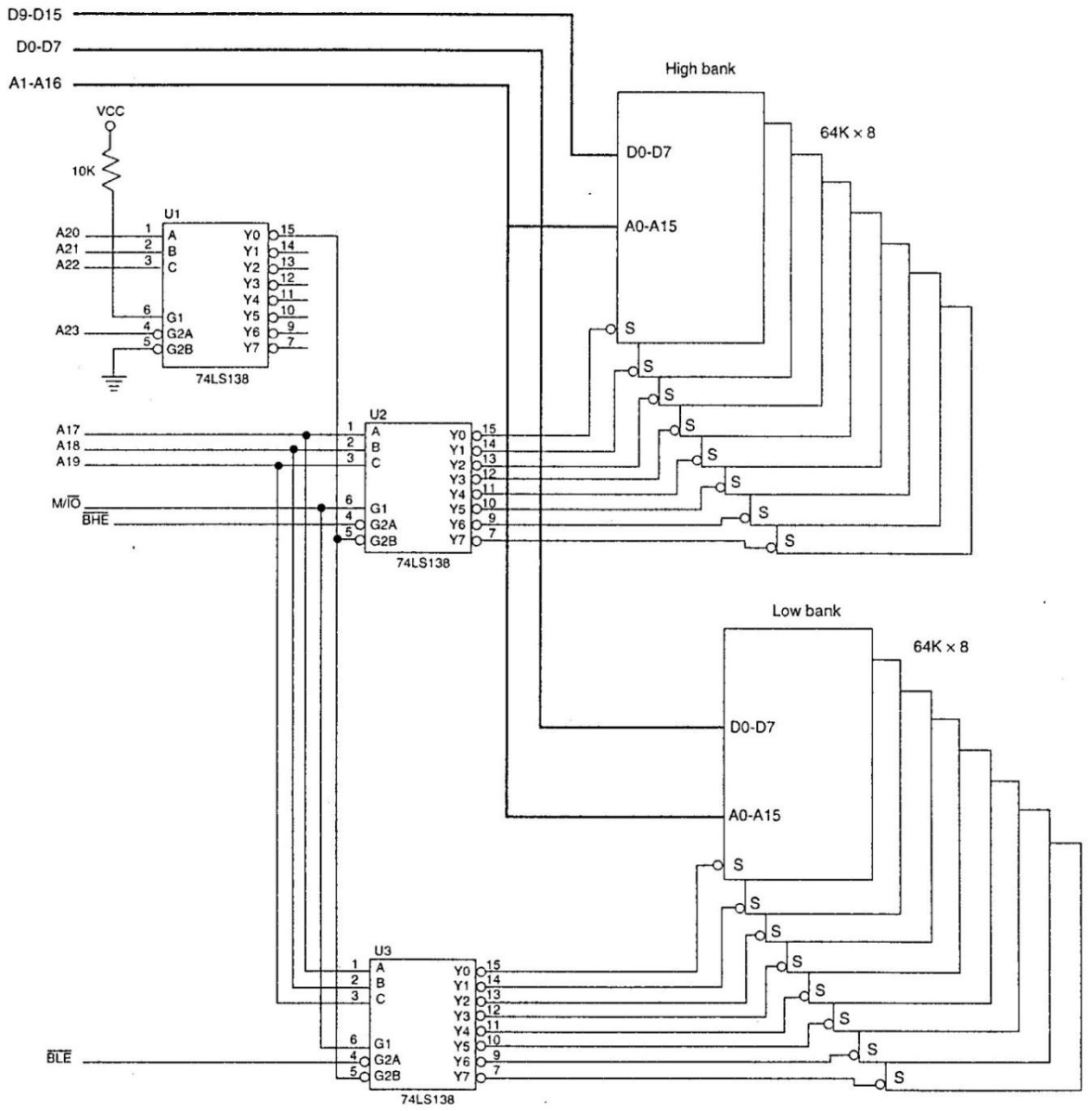


FIGURE 10–28 Separate bank decoders.

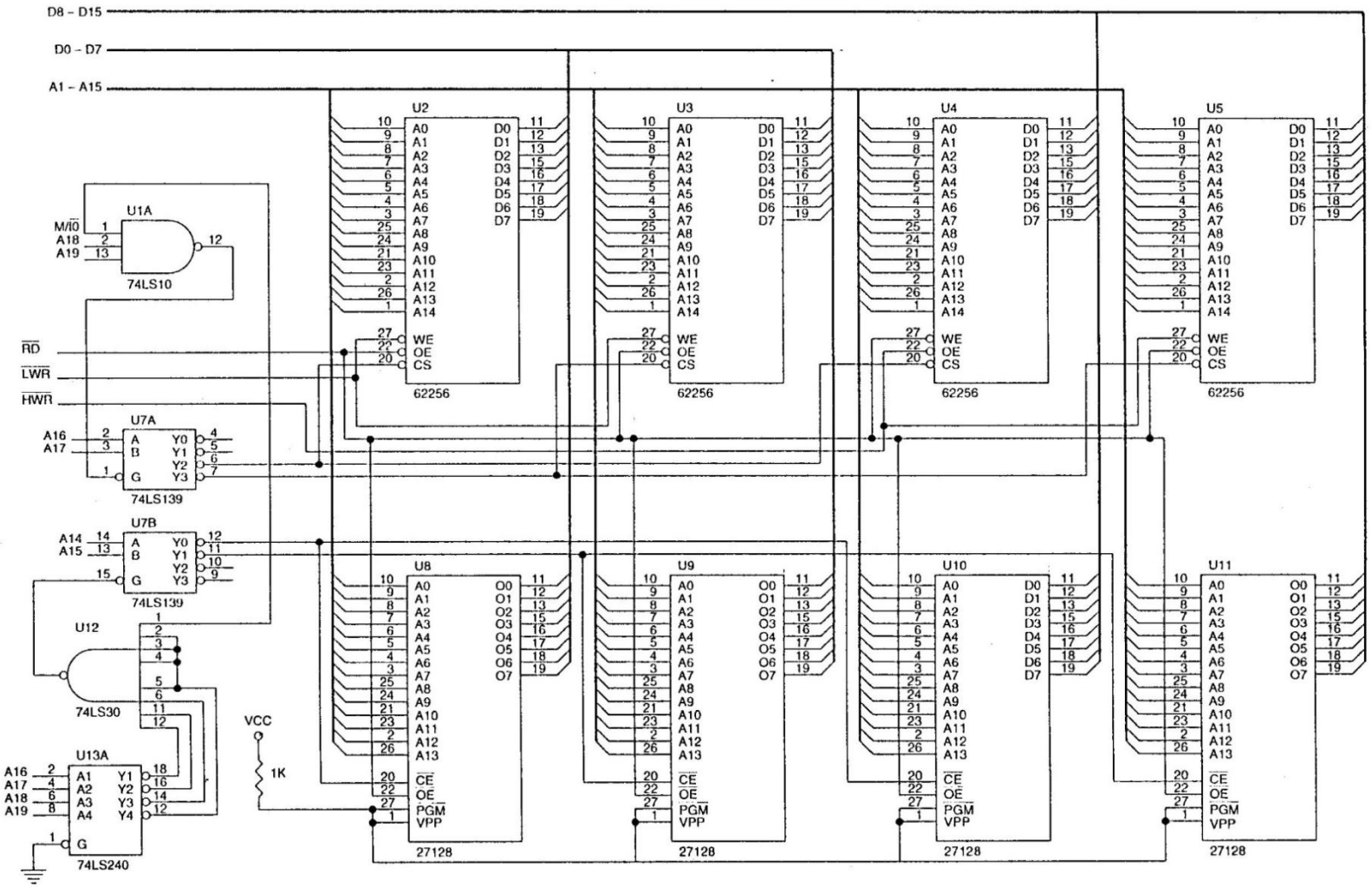


FIGURE 10–31 A memory system for the 8086 that contains a 64K-byte EPROM and a 128K-byte SRAM.

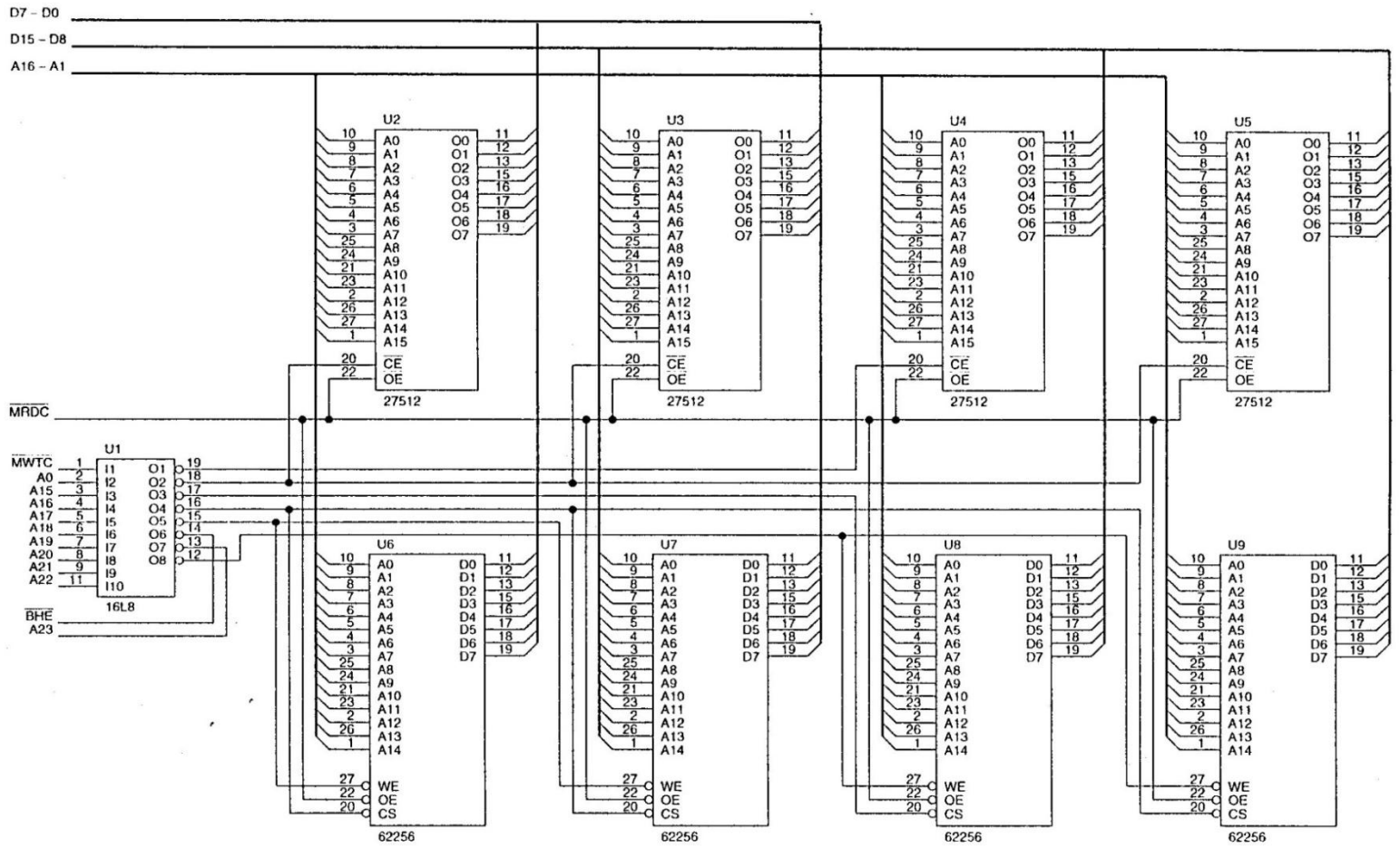


FIGURE 10–32 An 80386SX memory system containing 256K of EPROM and 128K of SRAM.

Interfacing Memory to 32- and 64-bit Data Bus μP

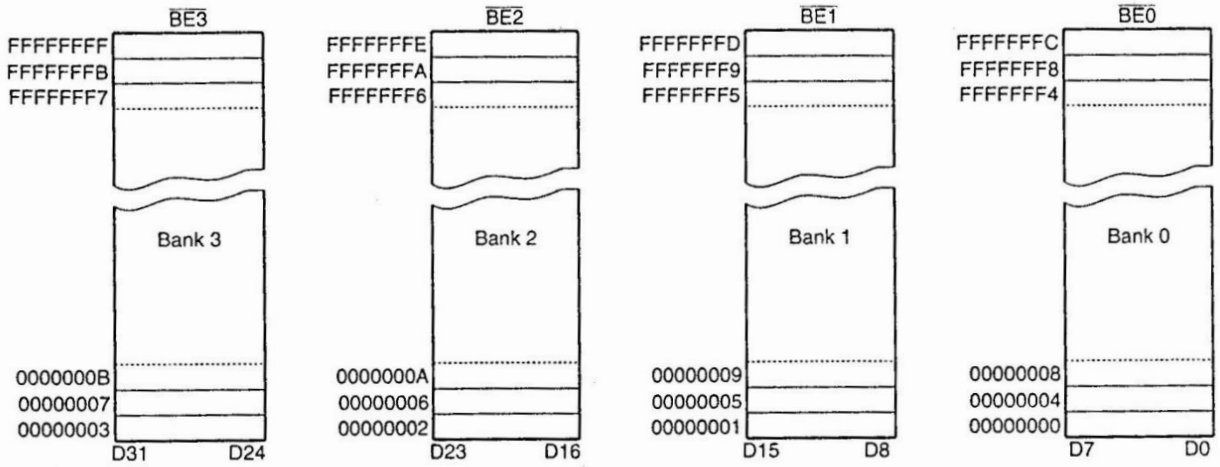
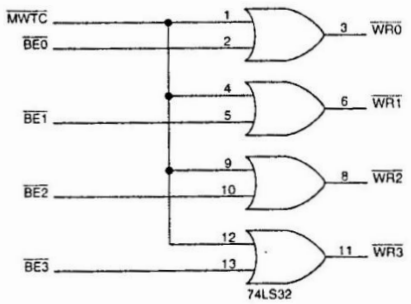


FIGURE 10–33 The memory organization for the 80386DX and 80486 microprocessors.

FIGURE 10–34 Bank write signals for the 80386DX and 80486 microprocessors.



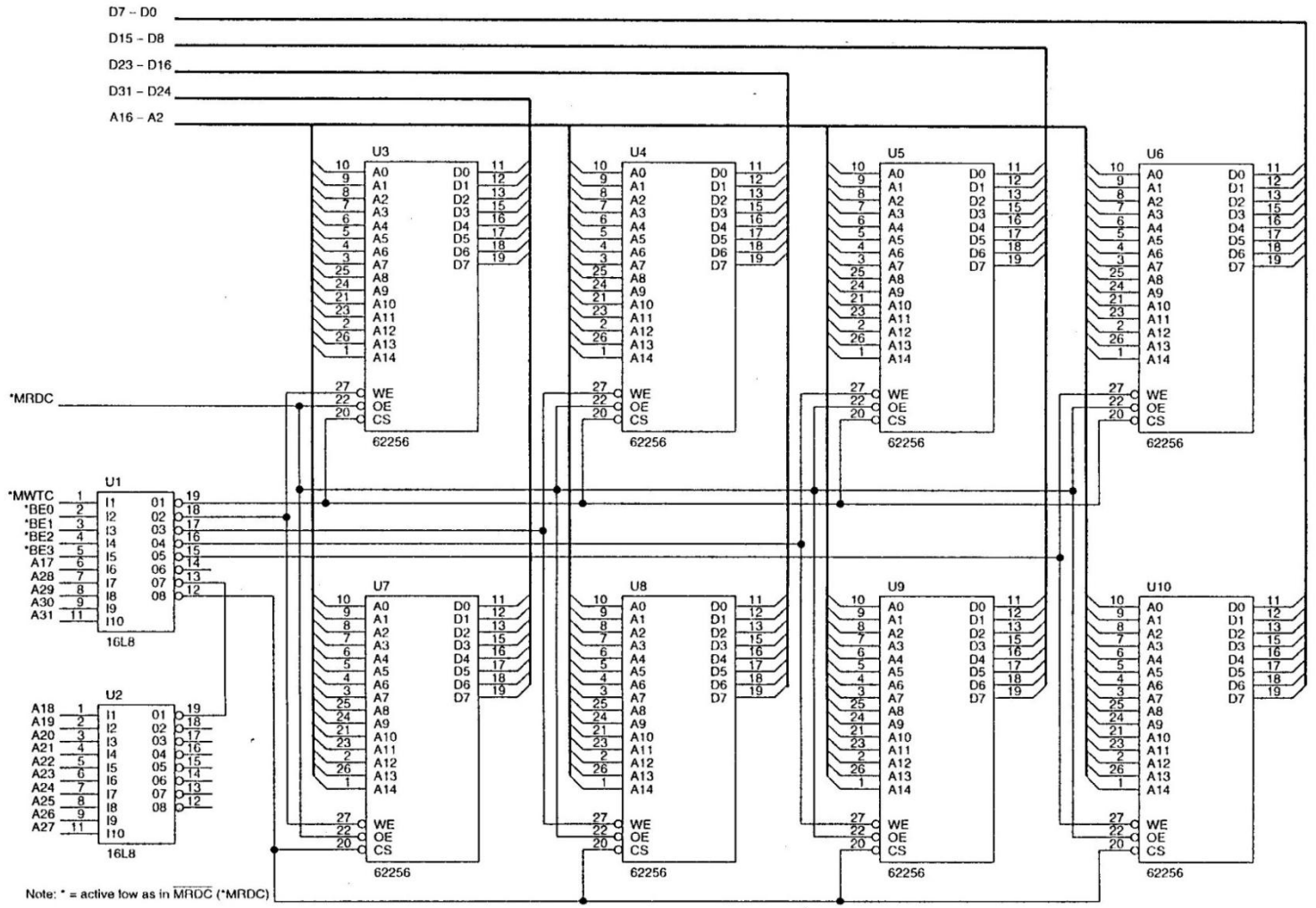


FIGURE 10–35 A small 256K SRAM memory system interfaced to the 80486 microprocessor.

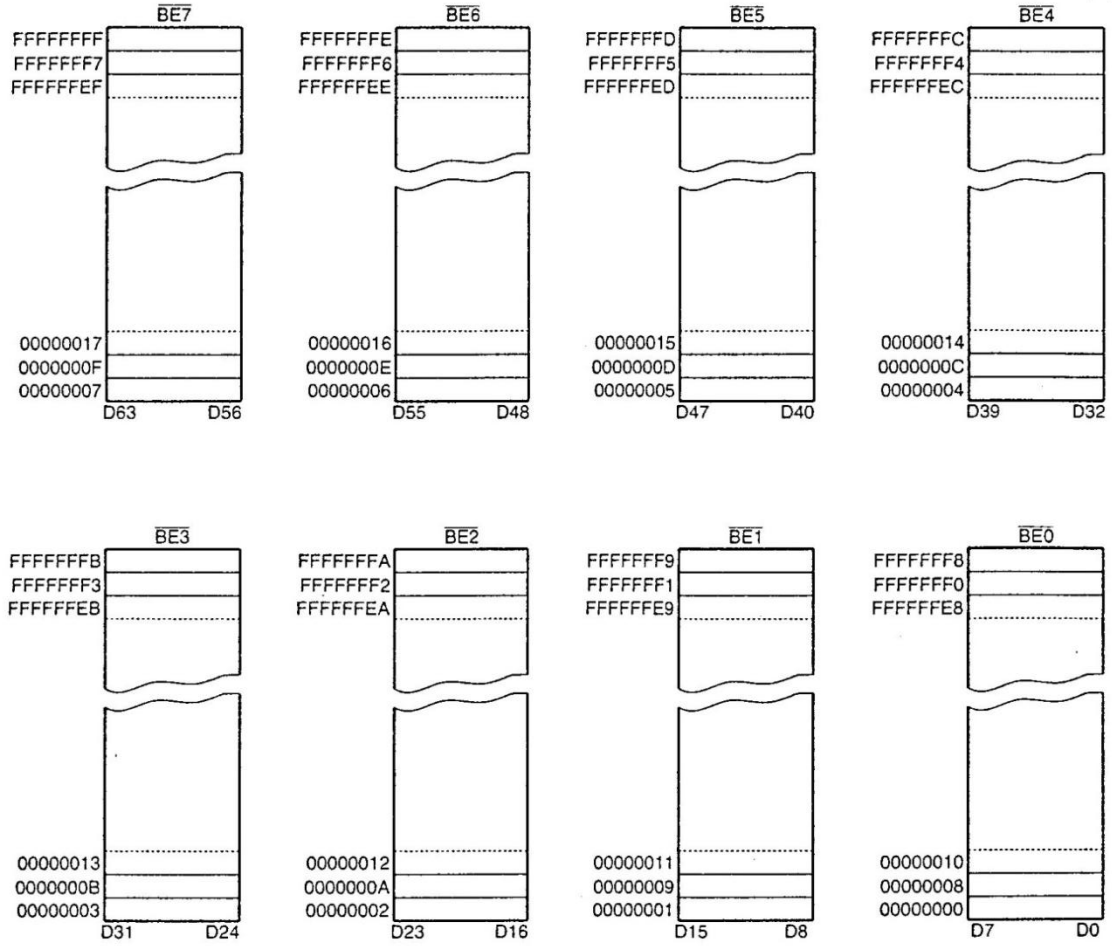
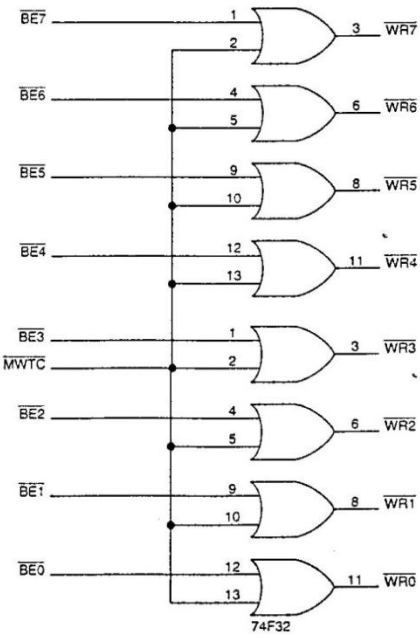


FIGURE 10–36 The memory organization of the Pentium–Pentium II microprocessors.

FIGURE 10–37 The generation of the write strobes for the Pentium–Pentium II microprocessors.



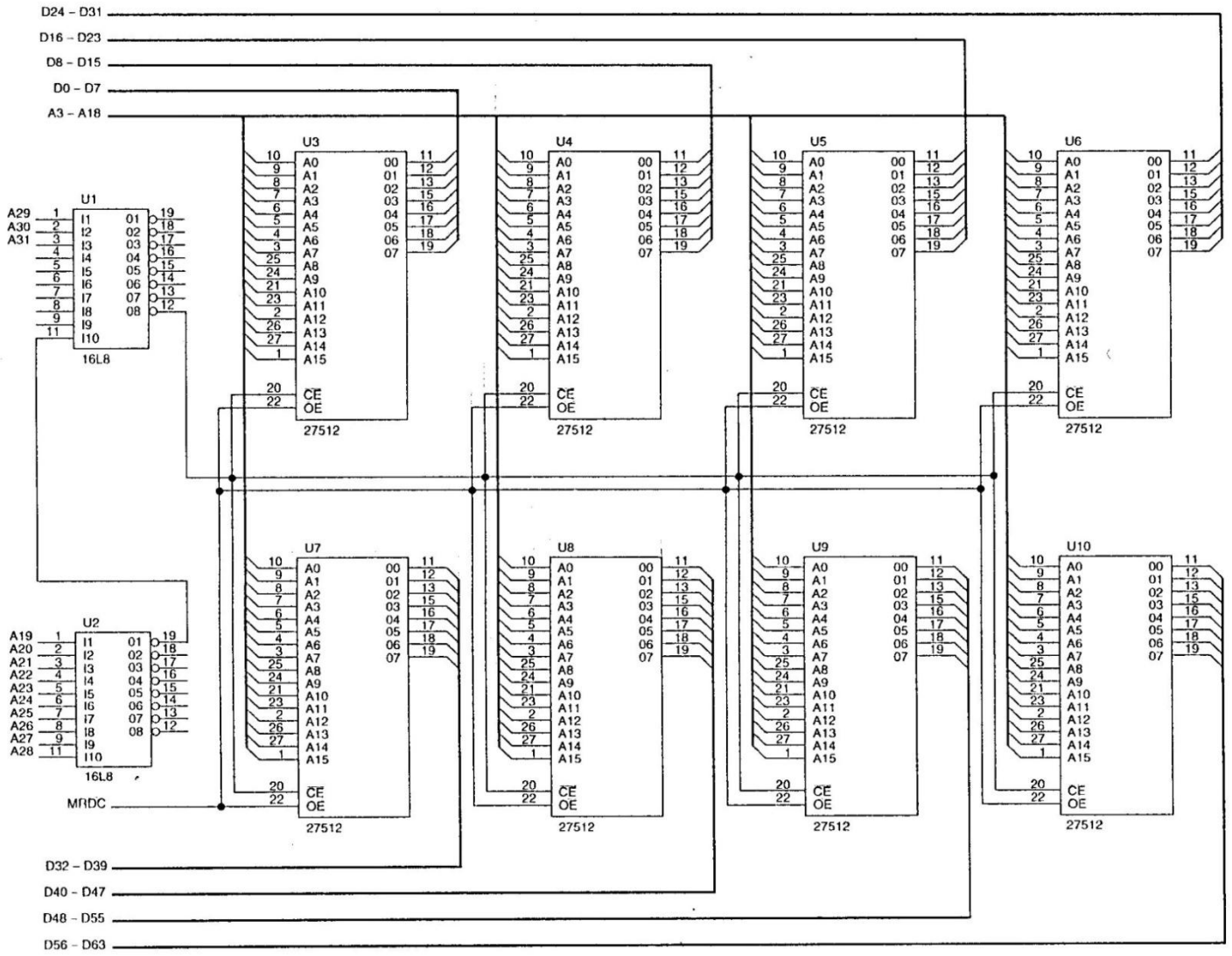


FIGURE 10–38 A small 512K-byte EPROM memory interfaced to the Pentium–Pentium II microprocessors.

Hamming Codes

- By R. W. Hamming; Commonly used in RAM
- k parity bits added to n data bits
- Bit positions numbered 1 to n + k
- Positions numbered with powers of two are for parity
- The k parity bits are generated as follows
  - P1 = XOR (all data bit position numbers with 1 in 1st bit)
  - P2 = XOR (all data bit position numbers with 1 in 2nd bit)
  - P4 = XOR (all data bit position numbers with 1 in 3rd bit)
  - P8 = XOR (all data bit position numbers with 1 in 4th bit)

- When n + k data are read, the parity are evaluated
  - C1 = XOR(all bit position numbers with 1 in 1st bit)
  - C2 = XOR(all bit position numbers with 1 in 2nd bit)
  - C4 = XOR(all bit position numbers with 1 in 3rd bit)
  - C8 = XOR(all bit position numbers with 1 in 4th bit)

- C=C1 C2 C4 C8=0 means no error, else there is error

- Decimal value of C indicates the error bit position
  - Error may be in data or parity bits
  - Hamming code detects and corrects error only in single bit
  - With an additional parity bit, two errors detected but not corrected

Example: Data = 11000100

Bit position	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

P_1 = XOR of bits (3, 5, 7, 9, 11) = $1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$

P_2 = XOR of bits (3, 6, 7, 10, 11) = $1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$

P_4 = XOR of bits (5, 6, 7, 12) = $1 \oplus 0 \oplus 0 \oplus 0 = 1$

P_8 = XOR of bits (9, 10, 11, 12) = $0 \oplus 1 \oplus 0 \oplus 0 = 1$

Bit position	1	2	3	4	5	6	7	8	9	10	11	12
	0	0	1	1	1	0	0	1	0	1	0	0

C_1 = XOR of bits (1, 3, 5, 7, 9, 11)

C_2 = XOR of bits (2, 3, 6, 7, 10, 11)

C_4 = XOR of bits (4, 5, 6, 7, 12)

C_8 = XOR of bits (8, 9, 10, 11, 12)

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	
	0	0	1	1	1	0	0	1	0	1	0	0	No error
	1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
	0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

	C_8	C_4	C_2	C_1
No error	0	0	0	0
Error in bit 1	0	0	0	1
Error in bit 5	0	1	0	1

The Hamming code can be used for data words of any length. In general, for k check bits and n data bits, the total number of bits, $n + k$, that can be in a coded word is at most $2^k - 1$. In other words, the relationship $n + k \leq 2^k - 1$ must hold. This relationship gives $n \leq 2^k - 1 - k$ as the number of bits for the data word. For example, when $k = 3$, the total number of bits in the coded word is $n + k \leq 2^3 - 1 = 7$, giving $n \leq 7 - 3 = 4$. For $k = 4$, we have $n + k \leq 15$, giving $n \leq 11$. Thus, the data word may be less than 11 bits, but must have at least five bits; otherwise, only three check bits will be needed. The relationships for $k = 3$ and $k = 4$ justify the use of four check bits for the eight data bits in the previous example.

Mano and Kime "Logic and Computer Design Fundamentals"

of 2—for example, 1, 2, 4, 8, 16, and so forth. These numbers are also the position numbers for the parity bits.

The basic Hamming code can detect and correct an error in only a single bit. Some multiple-bit errors are detected, but they may be corrected erroneously, as if they were single-bit errors. By adding another parity bit to the coded word, the Hamming code can be used to correct a single error and detect double errors. If we include this additional parity bit, the previous 12-bit coded word becomes $001110010100P_{13}$, where P_{13} is evaluated from the exclusive-OR of the other 12 bits. This produces the 13-bit word 0011100101001 (even parity). When this word is read from memory, the check bits and also the parity bit P are evaluated over the entire 13 bits. If $P = 0$, the parity is correct (even parity), but if $P = 1$, the parity over the 13 bits is incorrect (odd parity). The following four cases can occur:

If $C = 0$ and $P = 0$	No error occurred.
If $C \neq 0$ and $P = 1$	A single error occurred that can be corrected.
If $C \neq 0$ and $P = 0$	A double error occurred that is detected but cannot be corrected.
If $C = 0$ and $P = 1$	An error occurred in the P_{13} bit.

Note that this scheme will detect more than two erroneous bits in many cases, but is not guaranteed to detect all such errors.

A modified Hamming code to generate and check parity bits for a single-error-correction, double-error-detection scheme is most often used in real systems. The modified code uses a different parity check bit scheme that balances the number of inputs to the logic for each check bit and thus the number of inputs to each circuit that does the checking. The balancing minimizes the delay through the error correction and detection circuits. These circuits can be used in a RAM subsystem to add check bits during write operations and to correct single errors and detect double errors during read operations. (See Problem 6–14 at the end of the chapter.)

6-6 Programmable Logic Technologies

The next four sections deal with five types of programmable logic devices (PLDs): the read-only memory (ROM), the programmable logic array (PLA), the programmable array logic (PAL®) device, the complex programmable logic device (CPLD), and the field-programmable gate array (FPGA). Before treating these devices, we deal with the supporting programming technologies. In PLDs, programming technologies are applied to establish or break interconnections, build lookup tables, and control transistor switching. We will relate the technologies to these three applications.

The oldest of the programming technologies is the use of a *fuse*. Each of the programmable points in the PDL consists of a connection formed by a fuse. When a voltage considerably higher than the normal power supply voltage is applied across the fuse, the high current breaks the connection by blowing out the fuse. The

Basic I/O Interface

- Two methods: isolated I/O and memory-mapped I/O
- Isolated I/O
  - I/O locations are isolated from memory system
  - Only instructions IN, INS, OUT and OUTS used
- Memory-Mapped I/O
  - May use any instruction that references memory
  - I/O is treated like a memory location

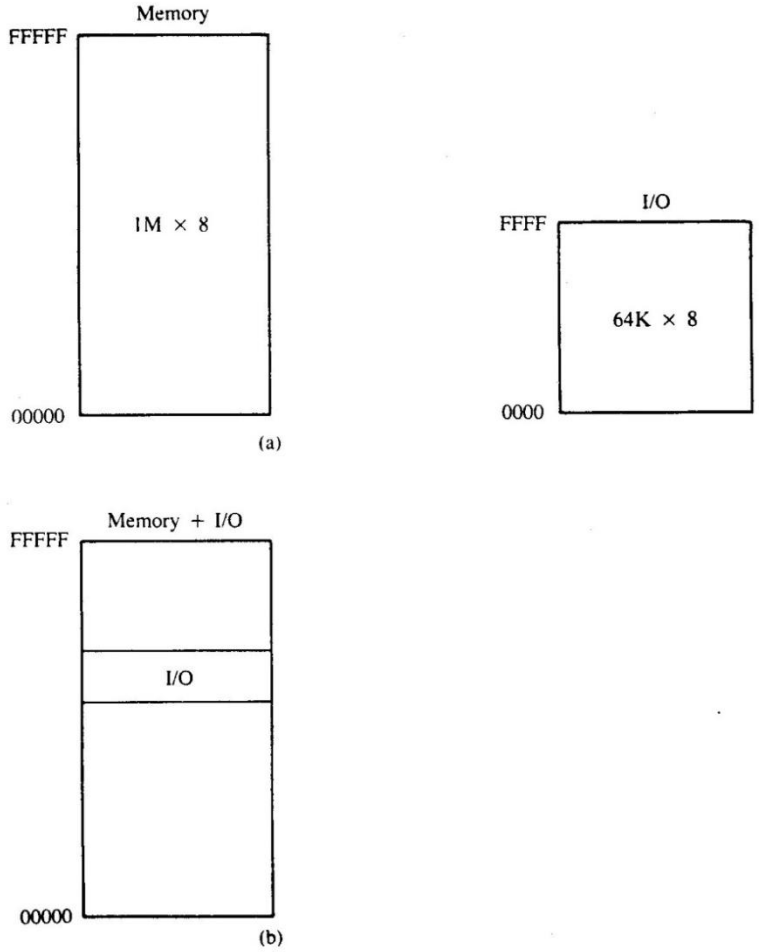


FIGURE 11–1 The memory and I/O maps for the 8086/8088 microprocessors. (a) Isolated I/O (b) Memory-mapped I/O.

Debouncing Switch Contacts

FIGURE 11–6 A single-pole, single-throw switch interfaced as a TTL device.

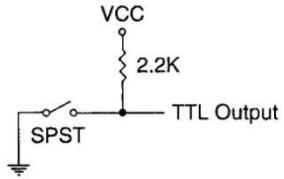
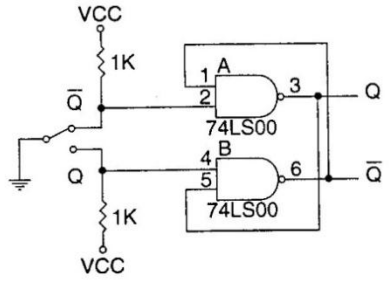
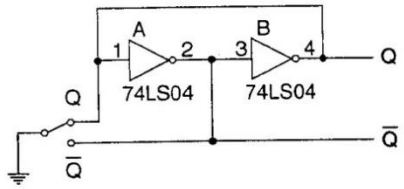


FIGURE 11–7 Debouncing switch contacts: (a) conventional debouncing and (b) practical debouncing.



(a)



(b)

I/O Address Decoding

EXAMPLE 11–2

```
AUTHOR    Barry B. Brey
COMPANY   BreyCo
DATE      7/1/99
CHIP      DECODER8 PAL16L8

;pins  1  2  3  4  5  6  7  8  9  10
       A0 A1 A2 A3 A4 A5 A6 A7 NC GND

;pins 11 12 13 14 15 16 17 18 19  20
       NC F7 F6 F5 F4 F3 F2 F1 F0 VCC

EQUATIONS

/F0 = A7 * A6 * A5 * A4 * A3 * /A2 * /A1 * /A0
/F1 = A7 * A6 * A5 * A4 * A3 * /A2 * /A1 * A0
/F2 = A7 * A6 * A5 * A4 * A3 * /A2 * A1 * /A0
/F3 = A7 * A6 * A5 * A4 * A3 * /A2 * A1 * A0
/F4 = A7 * A6 * A5 * A4 * A3 * A2 * /A1 * /A0
/F5 = A7 * A6 * A5 * A4 * A3 * A2 * /A1 * A0
/F6 = A7 * A6 * A5 * A4 * A3 * A2 * A1 * /A0
/F7 = A7 * A6 * A5 * A4 * A3 * A2 * A1 * A0
```

FIGURE 11–13 The I/O banks found in the 8086, 80186, 80286, and 80386SX.

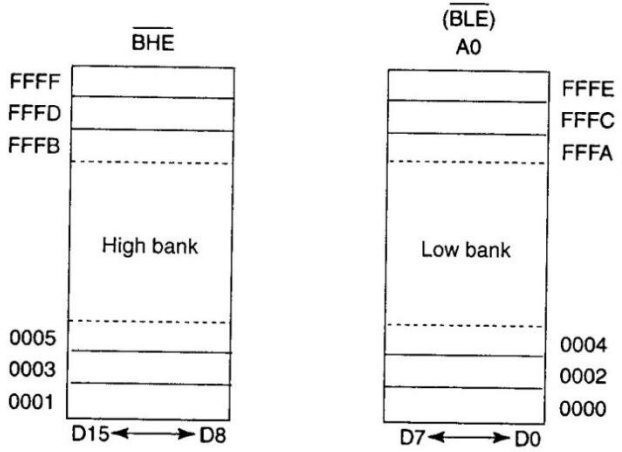
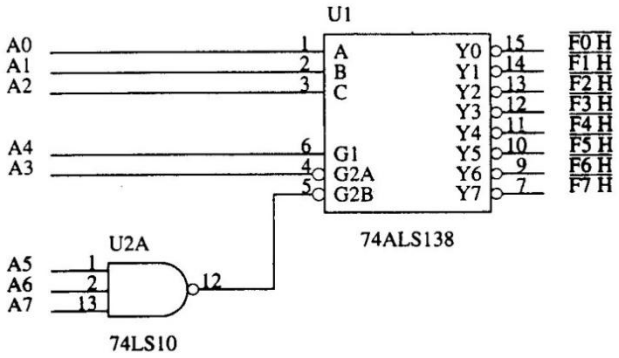


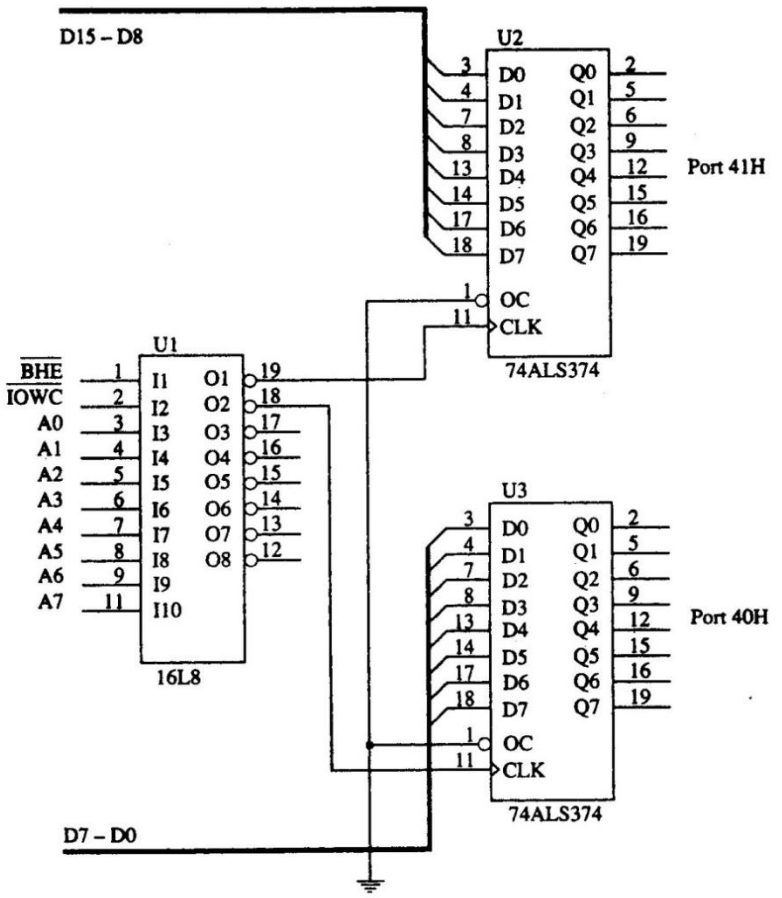
FIGURE 11–10 A port decoder that decodes 8-bit I/O ports. This decoder generates active low outputs for ports F0H–F7H.



EXAMPLE 11–4

```
AUTHOR    Barry B. Brey
COMPANY   BreyCo
DATE      7/3/99
CHIP      DECODERA PAL16L8

;pins  1     2  3  4  5  6  7  8  9  10
       BHE IOWC A0 A1 A2 A3 A4 A5 A6 GND

;pins 11 12 13 14 15 16 17 18 19 20
       A7 NC NC NC NC NC NC 40 41 VCC

EQUATIONS

/40 = /BLE * /IOWC * /A7 * A6 * /A5 * /A4 * /A3 * /A2 * /A1
/41 = /BHE * /IOWC * /A7 * A6 * /A5 * /A4 * /A3 * /A2 * /A1
```

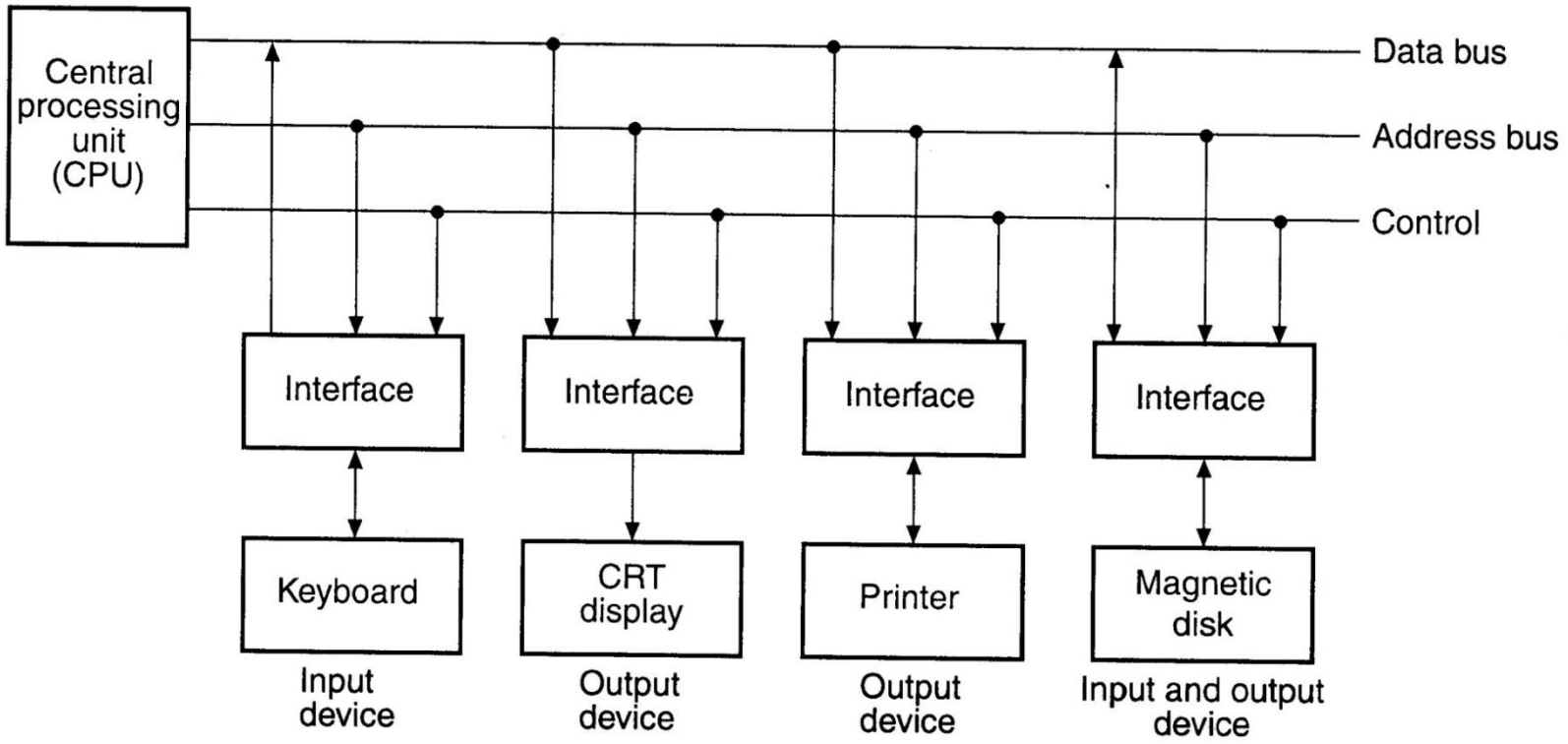
FIGURE 11–14 An I/O port decoder that selects ports 40H and 41H for output data.



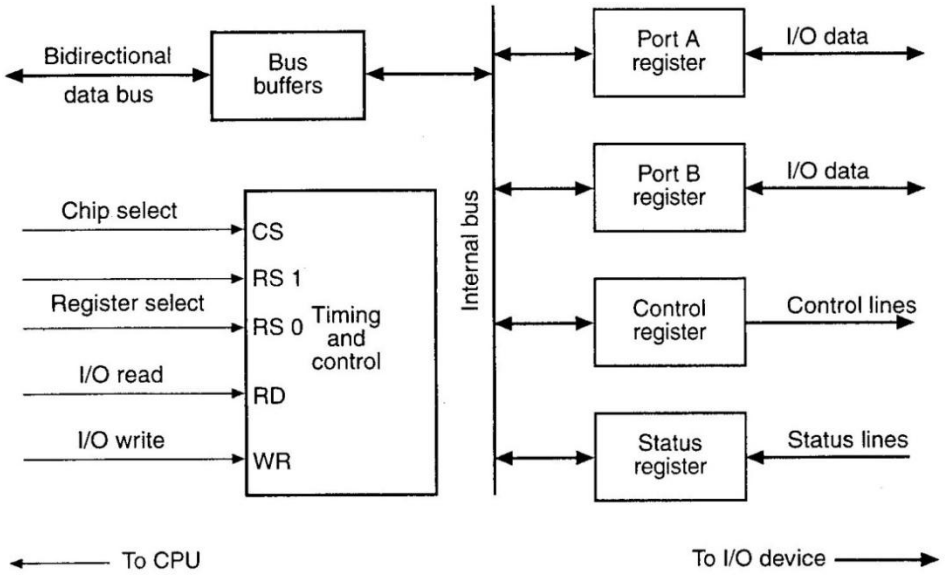
Interface Units

- Links between CPU and I/O
  - Some peripherals are electromechanical devices and need conversion of signal values
  - Data transfer rates of peripherals may differ from CPU clock and need synchronization mechanism
  - Data codes and formats in peripherals may differ from CPU
  - Operation modes of peripherals differ from each other and they need to be controlled so that they do not interfere each others operation

Connection of I/O devices to CPU



Typical I/O Interface



CS	RS1	RS0	Register selected
0	x	x	None: data bus in high-impedance state
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Asynchronous Data Transfer

- CPU and I/O usually have different clocks
- Strobing: simple method with one control signal
  - Transfer may be initiated by the source or destination
  - No indication that data ever captured by destination
  - No indication that source has put the data on bus
  - Speed is as low as that of slowest attached device
- Handshaking: two control signals, one from each side
  - Based on request and acknowledge signals

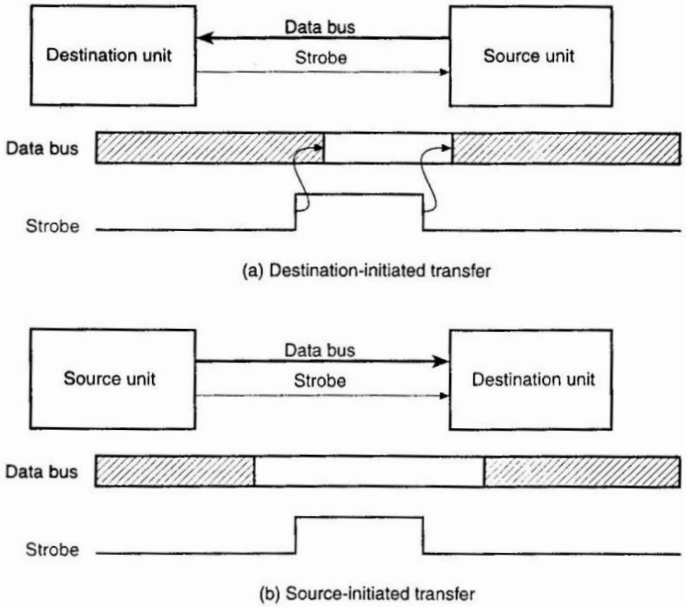


FIGURE 11-6
Asynchronous Transfer Using Strobing

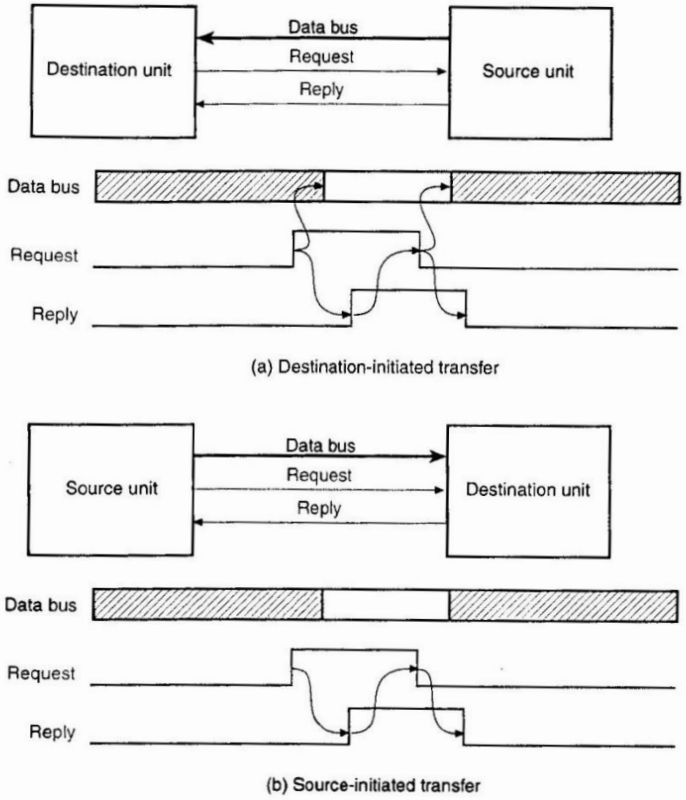


FIGURE 11-7
Asynchronous Transfer Using Handshaking

Impact of I/O on System Performance

- Some applications require high throughput, like Tax Service
- Some require low response time, like Personal Computers
- Many require both, like Automatic Teller Machines
  - Suppose a benchmark executes in 100 seconds of elapsed time, where 90 sec is CPU time and the rest is I/O. If CPU speed improves by 50% per year but I/O performance doesn't, how much faster the program runs at the end of 5 years?

  - Answer: 4.5

Magnetic Disks

- Components
  - One to 15 platters with two recordable surfaces each
  - Stack of platters has diameter of 1 to 8 inches and rotates at 3600 to 7200 RPM
  - Each disk surface divided into 1000 to 5000 concentric circles called tracks
  - Each track divided into 64 to 200 sectors which contain information
- Access time
  - Seek time + rotational latency + transfer time + controller time
  - What is the average rotational latency? 8.3 ms to 4.2 ms

Serial Communication

- Parallel: all bits sent at once
  - Fast, but lots of wires; good for short distance
- Serial: bits sent in sequence one at a time
  - Slow, but less expensive
  - Modems (modulator-demodulator) allow use of telephone lines
  - Simplex: only one way, line radio and television broadcasting
  - Half-duplex: both directions, but one at a time. Modems at both end change roles as transmitter and receiver in a turnaround time
  - Full-duplex: both directions simultaneously

Communication Between I/O and CPU

- CPU to I/O
  - Isolated I/O or memory-mapped I/O
- I/O to CPU
  - Operating system needs to know when I/O finished a task
  - Operating system should be notified of any errors in I/O
  - Two methods: Polling and Interrupt Driven
  - I/O may access memory directly (DMA)

- Polling (Programmed I/O)
  - I/O puts information in a status register
  - The OS periodically checks the status register
  - Busy wait loop is used to implement polling
  - Checks for I/O completion is dispersed among code
  - Advantage: simple, CPU controls all the work
  - Disadvantage: Polling overhead consumes a lot of CPU time

- Interrupt Driven (Exception Strategy)
- I/O interrupts CPU to get its attention
- Step 1: CPU receives interrupt signal from I/O
- Step 2: Current PC or IP is saved
- Step 3: CPU gets address of interrupt service routine
- Step 4: After executing ISR, CPU jumps back
- Advantage: user program is only halted during actual transfer
- Disadvantage: Special hardware needed to cause interrupt (I/O), detect interrupt (CPU), and save proper states to resume after interrupt (CPU)

- Compare I/O Interrupt and Processor Exceptions

Overhead of Polling in I/O Systems

- Polling is only suitable for low bandwidth devices
- Polling should be frequently enough not to lose any data
- Assume and 500 MHz µP with 400 clock cycle polling
  - Mouse must be polled 30 times per second
    - Fraction of processor clock cycle time consumed is 0.002%

    - Polling can be used for mouse without much impact on performance

- Floppy disk transfers data to processor in 16-bit units and has data rate of 50 KB/sec
  - Fraction of processor clock cycle time consumed is 2%
  - This is significant but can be tolerated in low-end systems
- Hard disk transfers data to processor in four 32-bit chunks and has data rate of 4 MB/sec
  - Fraction of processor clock cycle time consumed is 20%
  - This is one-fifth of CPU time and not acceptable to do

Overhead of Interrupt-Driven I/O Systems

- Has overhead for CPU only during actual data transfer
  - For previous hard disk system, assume each interrupt overhead is 500 clock cycles and hard disk transfers data 5% of the time

- Average fraction of CPU time consumed is 1.25%

Direct Memory Access (DMA)

- Allows devices to talk to memory directly
- Less overhead for CPU compared to polling and interrupt
- Suitable for high bandwidth devices like hard disk and transfer of large chunks of data at a time
- Interrupt still used but on completion of transfer or error
- DMA done by special controller device to master the bus
- Many DMA controller are flexible with respect to delays

- Three main steps are involved in DMA
  - Processor sets up DMA by supplying

    1) device identity, 2) operation to perform, 3) Source or destination memory address, and 4) number of bytes to transfer
  - DMA starts and DMA controller arbitrates for the bus
  - DMA transfer completes

    DMA controller interrupts CPU, and CPU checks for any possible errors

- Overhead of DMA I/O Systems
  - For previous hard disk systems

    assume initial setup for DMA takes 1000 clock cycles, handling interrupt at DMA completion takes 500 cycles, and average transfer from disk is 8 kB

    - Average fraction of CPU time consumed is 0.15%

Computer Performance

- Performance is major measure of evaluating computers
- Performance is task dependent and defined differently
  - Single users are interested in Response Time or Executing Time
    time between start and completion of a task improves by faster processors
  - Computer center managers are interested in Throughput
    total amount of work done in a given time improves by faster processor or using multiprocessors

- Performance of a machine: PER = 1 / EXE
  - Relative performance of machine A to machine B is given by PER(A)/PER(B) = EXE(B)/EXE(A)

- Performance improves by reducing the length of clock cycles (TAU) or number of clock cycles required for executing a program (CYC) EXE = CYC * TAU
- Execution time depends on the number of instructions in a program (INS) and the average clock cycles needed for instructions (CPI)

  CYC = INS * CPI

- Basic Performance equations
  - EXE = INS * CPI * TAU = INS * CPI/FRE
  - Check units: Time = $\frac{\text{Instructions}}{\text{Program}} \text{ x } \frac{\text{Clock Cycles}}{\text{Instruction}} \text{ x } \frac{\text{Seconds}}{\text{Clock Cycles}}$
  - CPU clock cycles is measured by looking at different types of instructions (n and i) and their clock cycle counts (COU)

    CYC = $\sum_{i=1}^{n} \text{CPI}_i \text{ x } \text{COU}_i$

- MIPS: Million Instruction Per Second
  - Alternative to execution time for evaluating systems
  - MIPS = $\frac{\text{Instruction Count}}{\text{Execution Time} \times 10^6}$
  - Faster machine means higher MIPS
  - Higher MIPS does not necessarily mean higher or better performance!
  - MIPS is instructions execution rate with no regard to capabilities
  - Cannot use MIPS to compare computers with different instructions
  - MIPS varies for different programs on the same machine

Exercises on Organization

- For a Pentium II processor descriptor that contains a base address of 00280000 H, a limit of 00010 H, and G=1, what starting and ending locations are addressed?

- Code a descriptor that describes a memory segment that begins at location 03000000 H and ends at location 05FFFFF H. This is a data segment that grows upward in the memory system and can be written, for an 80386 Intel processor.

- If the processor sends linear address 00200000 H to the paging mechanism, which paging directory entry is accessed and which page entry is accessed.

- What is wrong with a MOV [BX],[DI] instruction?

- What, if anything, is wrong with MOV AL,[BX][DI] instruction?

- Suppose DS=1100 H, BX=0200 H, LIST=0250 H, and SI=0500 H, determine the address accessed by each of the following instructions.

  a) MOV LIST[SI],EDX

  b) MOV CL,LIST[BX+SI]

  c) MOV CH,[BX+SI]

- Explain what happens when PUSH EAX instruction is executed. Assume SP=0100 H and SS=0200 H.

- Develop a sequence of instructions that copy 12 bytes of data from an area of memory addressed by SOURCE into an area of memory addressed by DEST.

- What is wrong with a MOV CS,AX instruction?

- If AX=1001 H and DX=20FF H, list the sum and the content of each flag register bit (C, A, S, Z and O) after the ADD AX, DX instruction executes.

- What is wrong with the INC[BX] instruction?

- Develop a sequence of instructions that sets (to 1) the rightmost 4 bits of AX, clears (to 0) the leftmost three bits of AX, and inverts bits 7,8, and 9 of AX.

- Why are buffers required in 8086- and 8088-based systems?

- What two 8086 operations occur during a bus cycle?

- Briefly describe the purpose of each T state from T1 to T4.

- Modify the NAND gate decoder in Figure 10-13 to select the memory for address range DF800 H- DFFFF H.

- Modify Figure 10-19 by rewriting the PAL program to address B0000 H- BFFFF H.

- Modify the circuit of Figure 10-20 to select memory locations 68000 H-6FFFF H.

Exercises on Hamming Code

- Given the 11-bit data word 00100111010, generate the corresponding 15-bit Hamming Code word.

- A 12-bit Hamming code word contains 8 bits of data and 4 parity bits is read from the memory. What was the original 8-bit data word that was written into memory if the 12-bit word read out is:
  a)000010101010, b)111110010110, and c) 100111110100.

- How many parity check bits must be included with the data word to achieve single error correction and double error detection when the data word contains: a) 16 bits, b) 32 bits, and c) 64 bits

- It is necessary to formulate the Hamming code for 4 data bits D3, D5, D6, D7 together with 3 parity bits P1, P2 and P4.

  a) Evaluate the 7-bit composite code word for the data word 0101

  b) Evaluate the 3 check bits C1, C2 and C4, assuming no error.

  c) Assume an error in bit D5 during storage into memory. Show how the error in the bit is detected and corrected.

  d)Add a parity bit P to include double error detection in code. Assume that errors occurred in bits P2 and D5. Show how this double error is detected.

Exercises on Computer Performance

- Suppose we have two implementations of the same instruction set architecture. Machine A has a clock cycle time of 1 ns and a CPI of 2.0 for some program, and machine B has a clock cycle time of 2 ns and a CPI of 1.2 for the same program. Which machine is faster for this program, and by how much?

- Our favorite program runs in 10 seconds on computer A, which has a 400 MHz clock. We are trying to help a computer designer build a machine, B, that will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for this program. What clock rate should we tell the designer to target?

- A compiler designer is trying to decide between two code sequences for a particular machine. The hardware designers have supplied the following facts:

Instruction Class	CPI for this Instruction Class
A	1
B	2
C	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

Code Sequence	Instruction Counts for Instruction Class		
	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

- Consider the machine with three instruction classes and CPI measurements from the previous problem. Now suppose we measure the code for the same program from two different compilers and obtain the following data:

Codes From Compiler	Instruction Counts (in billions) for Instruction Class		
	A	B	C
1	5	1	1
2	10	1	1

- Assume that the machine's clock rate is 500 MHz. Which code sequence will execute faster according to MIPS? According to execution time?