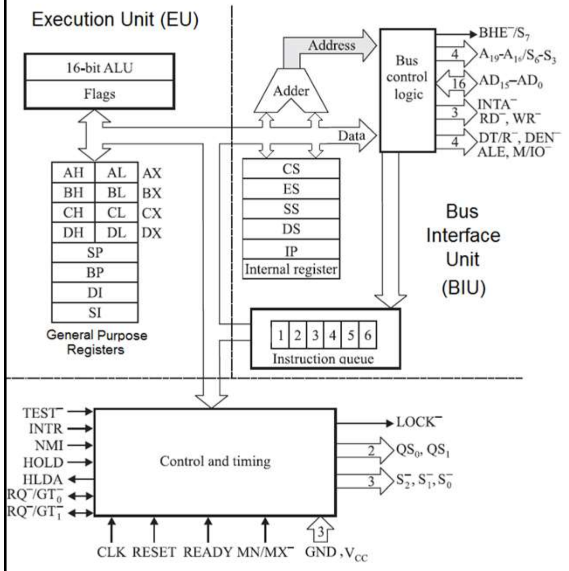
Computer Structure and Language

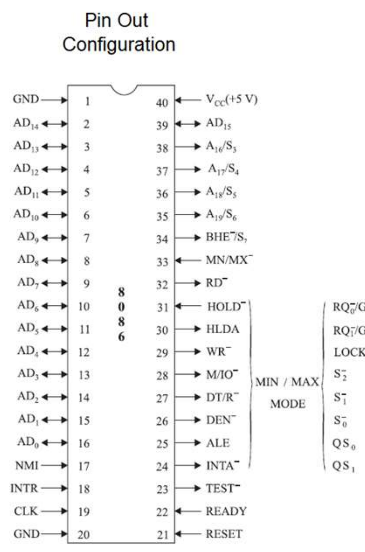
8086/8088 Hardware Design

Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology (SUT)
Tehran, Iran

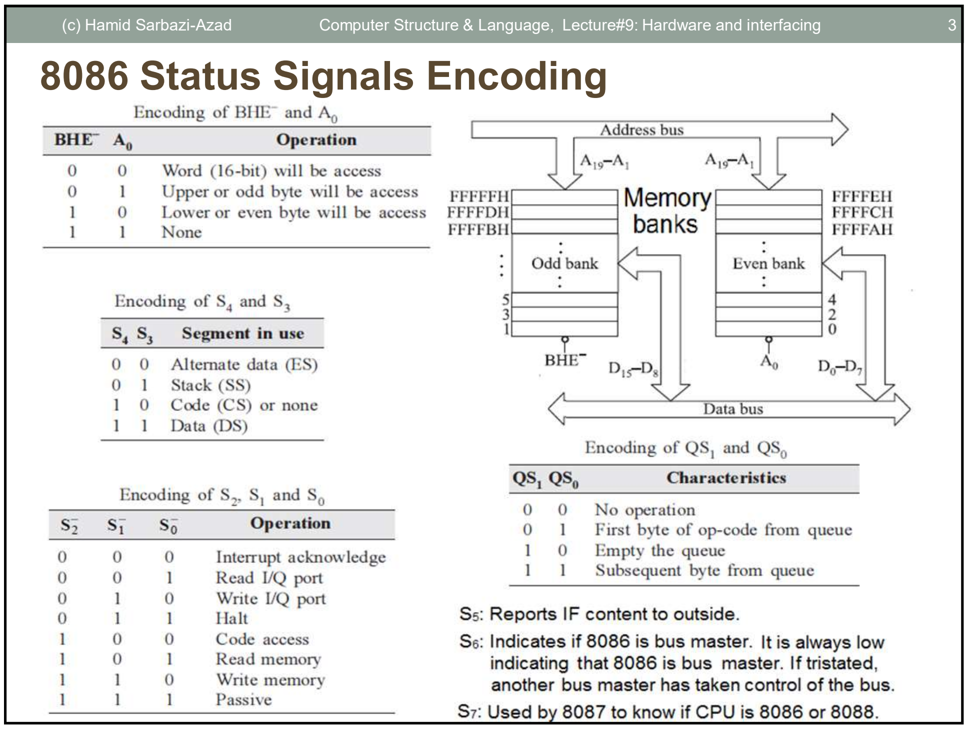
1

Intel’s 8086 Pins/Signals





2



8086 Status Signals Encoding

Encoding of BHE⁻ and A_0

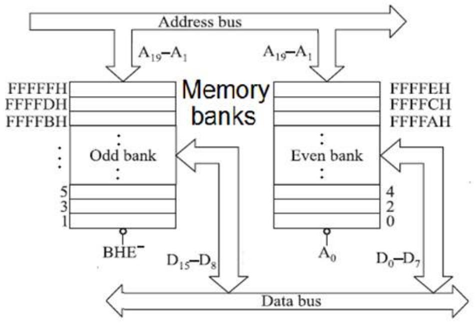
BHE⁻	A_0	Operation
0	0	Word (16-bit) will be access
0	1	Upper or odd byte will be access
1	0	Lower or even byte will be access
1	1	None

Encoding of S_4 and S_3

S_4	S_3	Segment in use
0	0	Alternate data (ES)
0	1	Stack (SS)
1	0	Code (CS) or none
1	1	Data (DS)

Encoding of S_2, S_1 and S_0

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Operation
0	0	0	Interrupt acknowledge
0	0	1	Read I/Q port
0	1	0	Write I/Q port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive



Encoding of QS_1 and QS_0

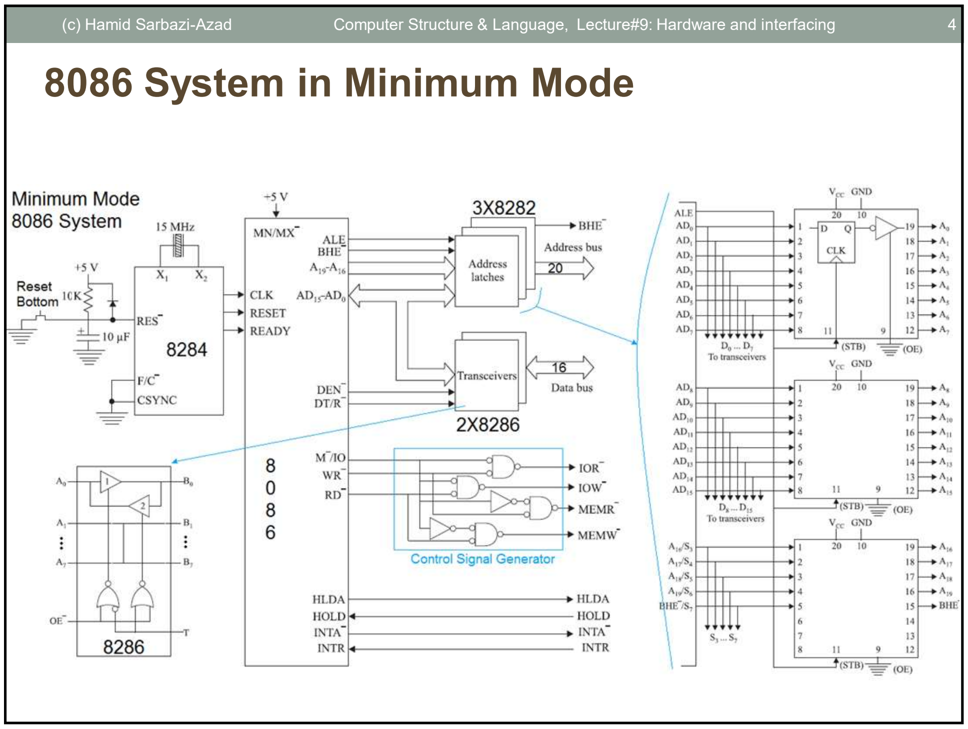
QS_1	QS_0	Characteristics
0	0	No operation
0	1	First byte of op-code from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

S_5: Reports IF content to outside.

S_6: Indicates if 8086 is bus master. It is always low indicating that 8086 is bus master. If tristated, another bus master has taken control of the bus.

S_7: Used by 8087 to know if CPU is 8086 or 8088.

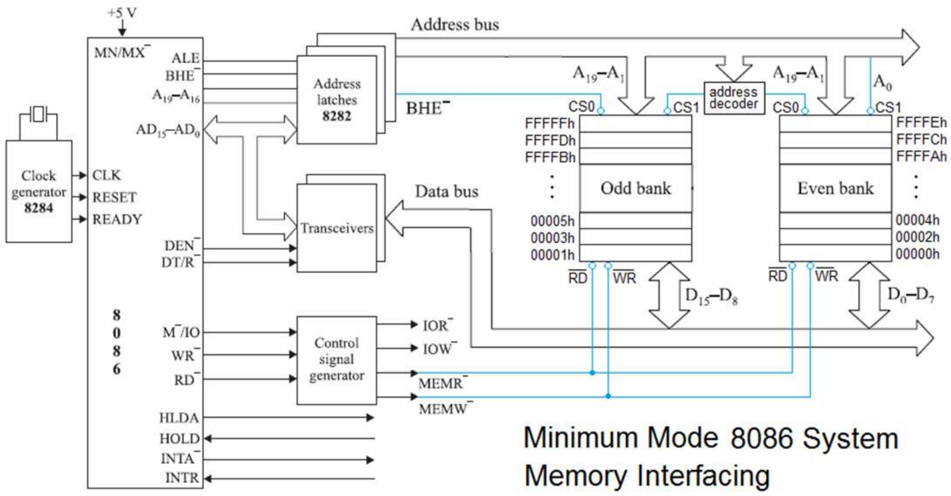
3



8086 System in Minimum Mode

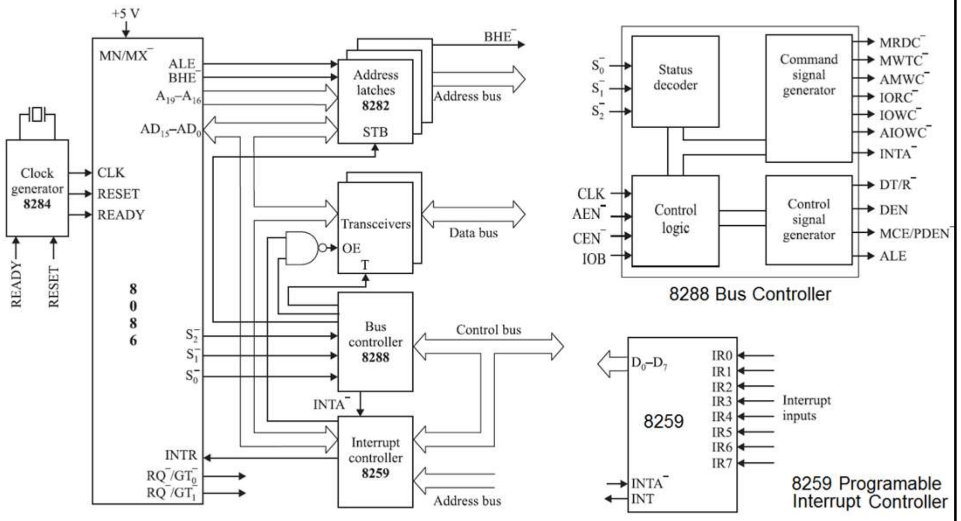
4

Memory Interface in Minimum Mode

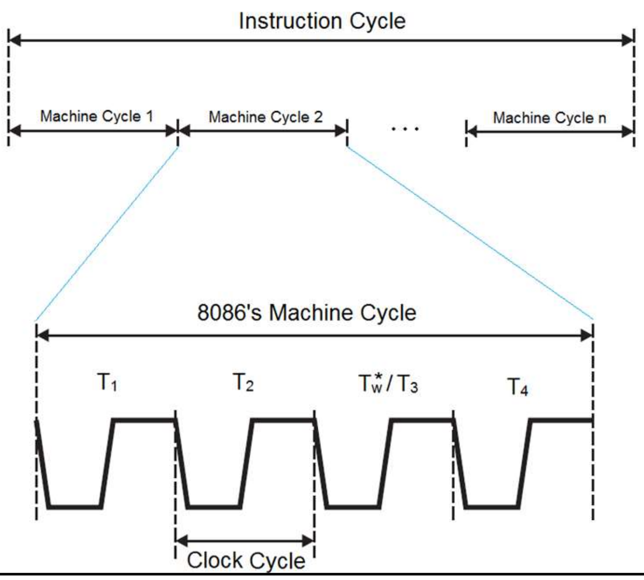


Minimum Mode 8086 System Memory Interfacing

8086 System in Maximum Mode

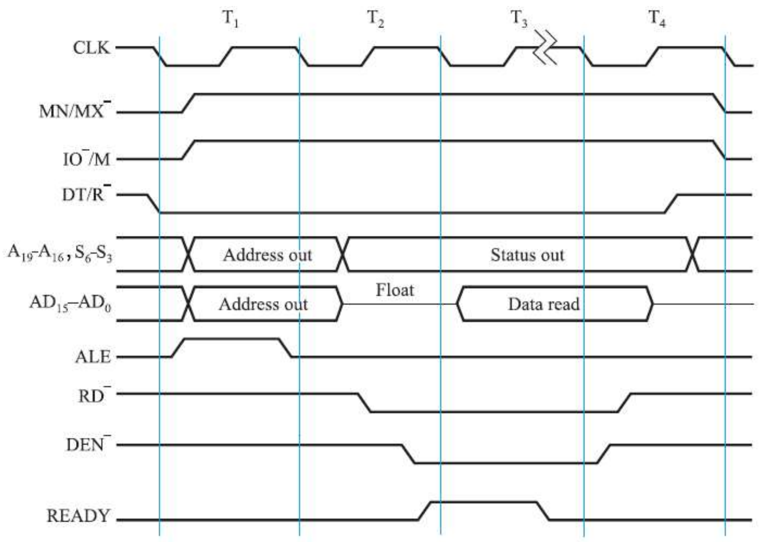


Instruction, Machine (Bus), and Clock Cycles



7

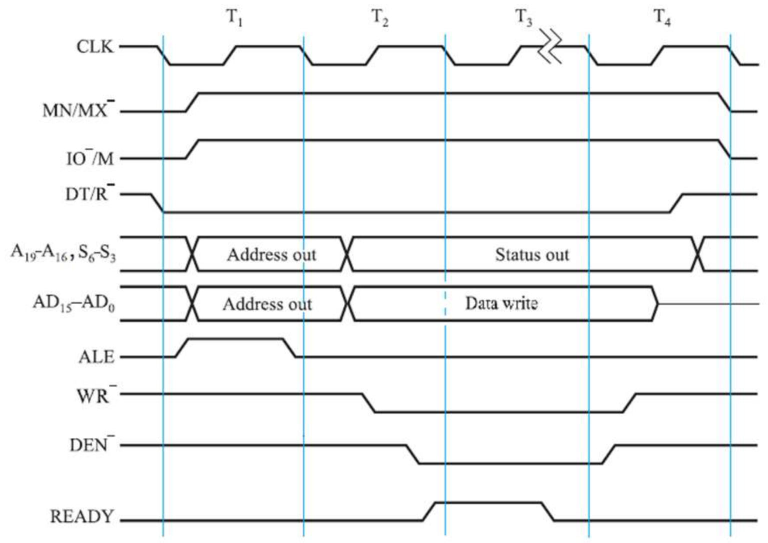
8086’s Memory Read Cycle (Minimum Mode)



Timing diagram for 8086 minimum mode memory read

8

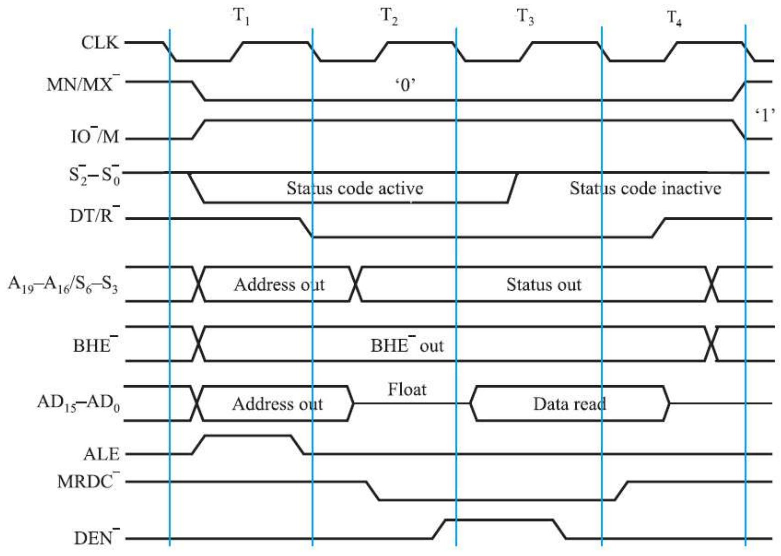
8086's Memory Write Cycle (Minimum Mode)



The timing diagram for 8086 minimum mode memory write

9

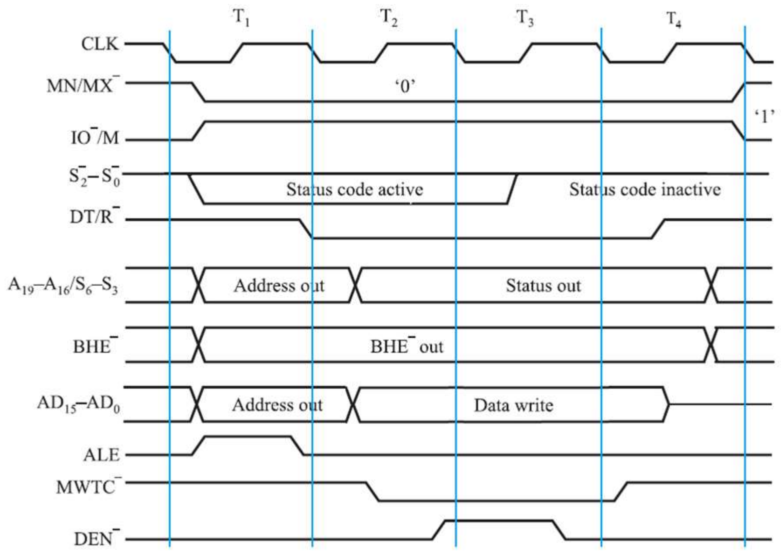
8086's Memory Read Cycle (Maximum Mode)



Maximum mode memory read bus cycle of 8086 system

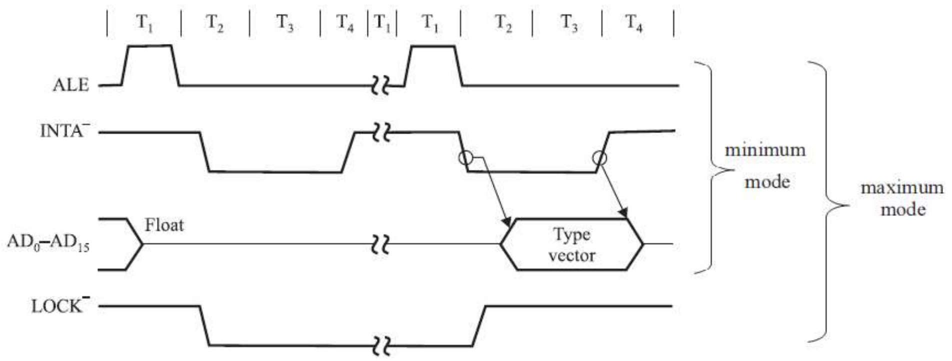
10

8086's Memory Write Cycle (Maximum Mode)



Maximum mode memory write bus cycle of 8086 system

Interrupt Acknowledge Cycle



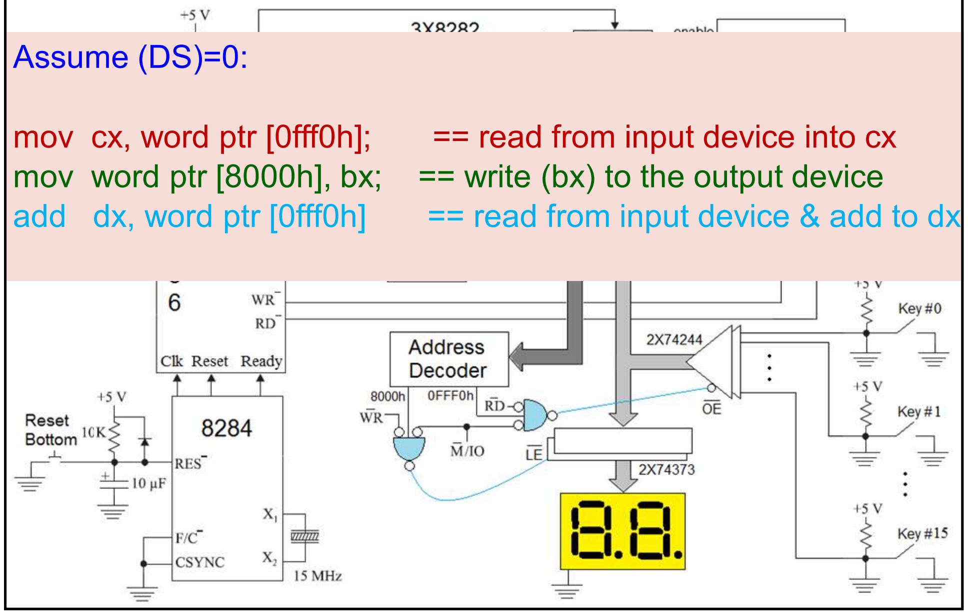
Interrupt acknowledge machine cycle

Input/Output Device Interfacing:

- **Memory Mapped:** I/O device is treated as a location of memory address space.
- **I/O Mapped:** I/O device is accessed in I/O address space.

13

Memory Mapped I/O device interfacing



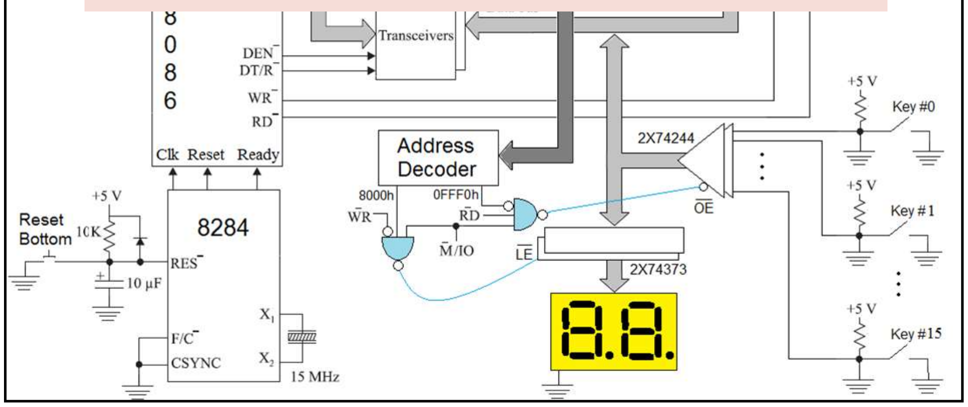
Assume (DS)=0:

```
mov  cx, word ptr [0fff0h];      == read from input device into cx
mov  word ptr [8000h], bx;     == write (bx) to the output device
add  dx, word ptr [0fff0h]      == read from input device & add to dx
```

14

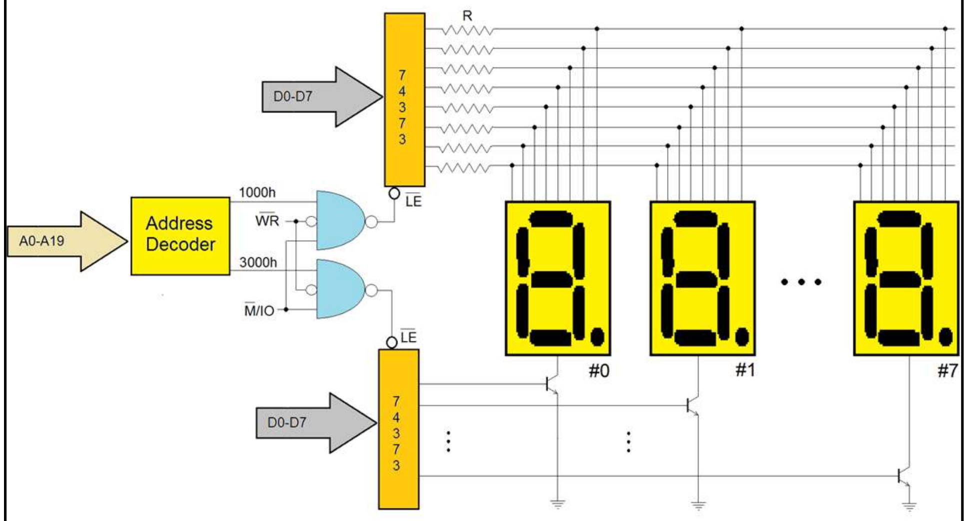
I/O Mapped I/O device interfacing

```
mov   dx,8000h
out   dx,ax ;     write (ax) to output device
mov   dx,0fff0h
in    ax,dx ;     read from input device into ax
```



15

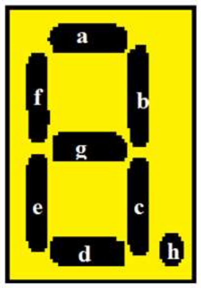
Design Example 1: Build a power-efficient 8-digit 7-segment display.



16

Design Example 1: Build a power-efficient 8-digit 7-segment display. (cont.)

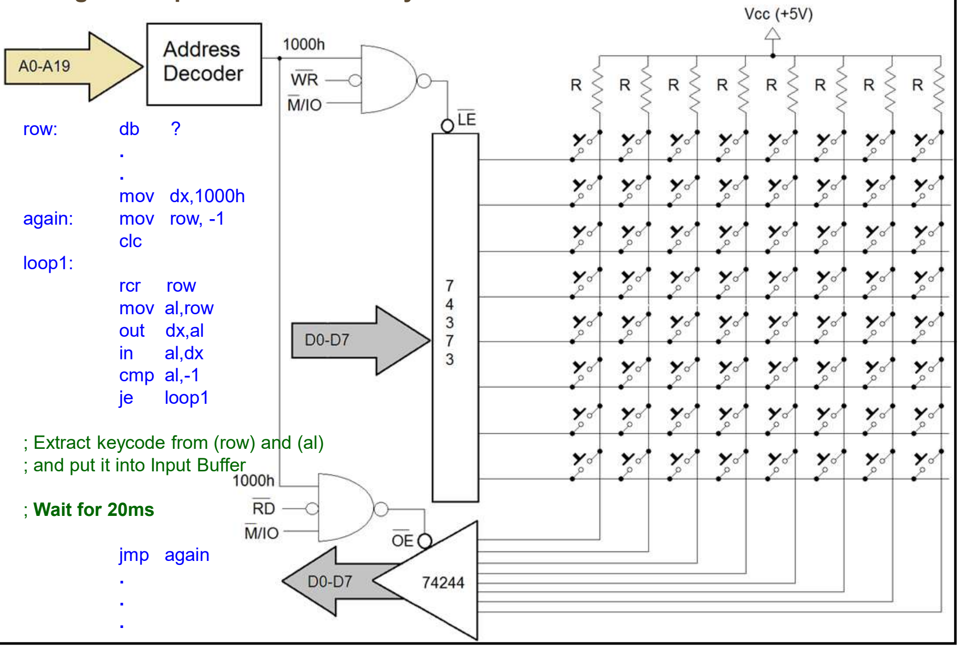
```
display     proc        far

            %pushr      (bp,bx,ax,si,dx)

            mov         bp,sp
            lea         bx,seven_seg
            mov         si,[bp+14]
            mov         cx,8
 next:      mov         al, byte ptr [si+7]
            xlat
            mov         dx,1000h
            out         dx,al
            mov         dx,3000h
            mov         al,order
            out         dx,al
            rol         order,1
            dec         si
            call        delay20ms
            loop        next

            %popr       (dx,si,ax,bx,bp)
            ret         2

  seven_seg:    db      FCh,60h,DAh,F2h,66h,B6h,BEh,E0h,FEh,F6h,1
      order:    db      0FEh

display     endp
```



digit	abcdefgh
0	11111100
1	01100000
2	11011010
3	11110010
4	01100110
5	10110110
6	10111110
7	11100000
8	11111110
9	11110110
.	00000001

17

Design Example 2: Build a 8x8 keyboard.

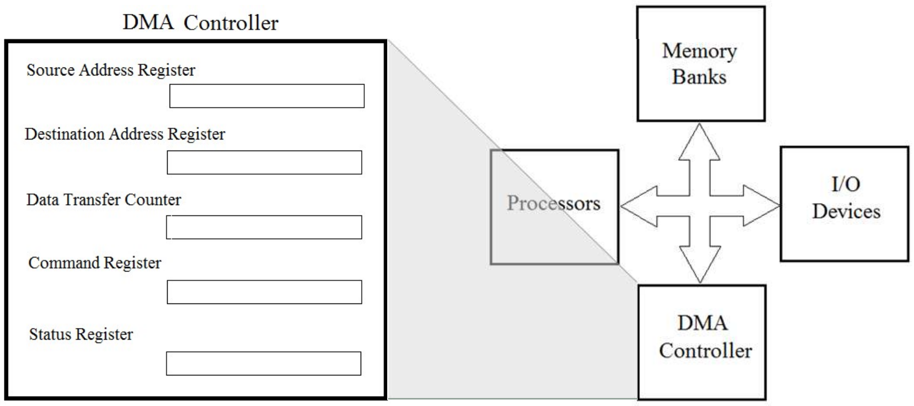


```
row:        db      ?
            .
            .
            mov     dx,1000h
again:      mov     row, -1
            clc
loop1:
            rcr     row
            mov     al,row
            out     dx,al
            in      al,dx
            cmp     al,-1
            je      loop1

; Extract keycode from (row) and (al)
; and put it into Input Buffer

; Wait for 20ms

            jmp     again
            .
            .
            .
```

18

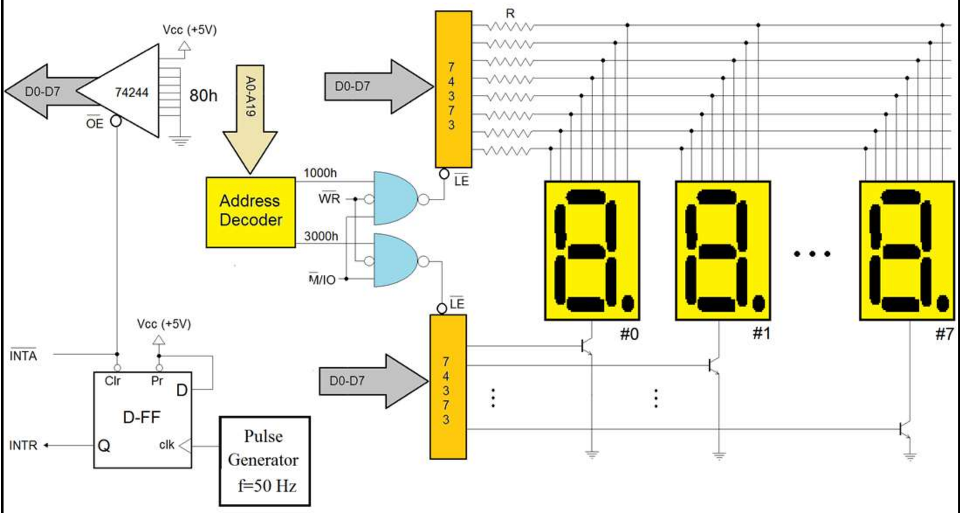
I/O Schemes:

1. **Programmed I/O: CPU is involved in checking for data availability and for data movement.**
2. **Interrupted I/O: CPU is involved for data movement.**
3. **Direct memory Access (DMA): CPU is not involved in I/O operation.**



19

Design Example 1: Build a power-efficient 8-digit 7-segment display (Interrupted).



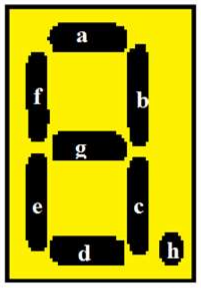
20

Design Example 1: Build a power-efficient 8-digit 7-segment display (cont).

```
display_buf:   db   8 dup (0)    ; a global buffer accessed by producer & displayer


displayer   proc        far      ; this is executed as timer  interrupt service routine

            %push       (flags, dx, ax, bx)  ; push used registers
            mov         si,index
            mov         al, byte ptr display_buf [si]
            lea         bx,seven_seg
            xlat
            mov         dx,1000h
            out         dx,al
            mov         dx,3000h
            mov         al,order
            out         dx,al
            rol         order,1
            dec         index
            and         index,7

            %pop        (bx,ax,dx,flags)    ; pop used registers
            iret

       index:   dw      7
   seven_seg:   db      FCh,60h,DAh,F2h,66h,B6h,BEh,E0h,FEh,F6h,1
       order:   db      0FEh
displayer   endp
```

In main program, at the beginning:

```
mov     ax,0
mov     es,ax
mov     es:[200h], offset displayer
mov     es:[202h], seg displayer
```

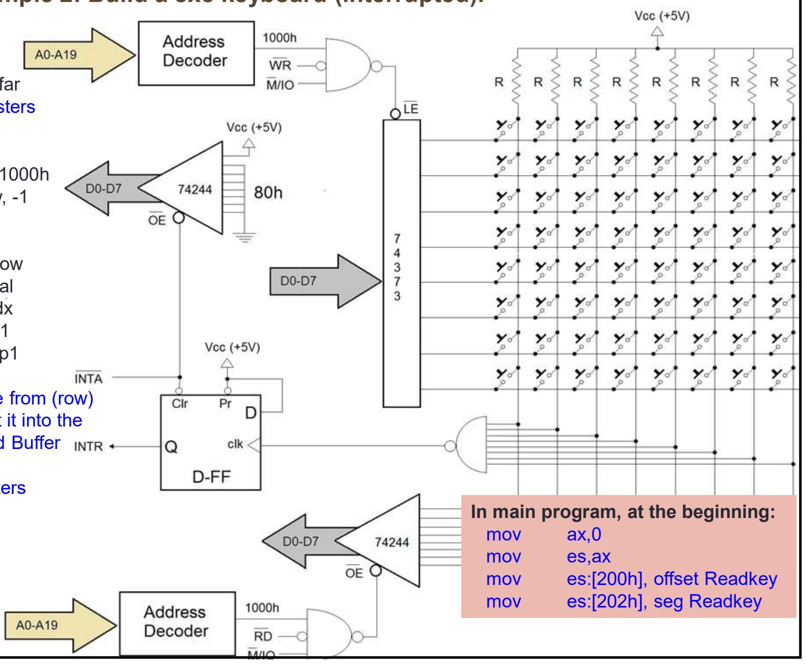


digit	abcdefgh
0	11111100
1	01100000
2	11011010
3	11110010
4	01100110
5	10110110
6	10111110
7	11100000
8	11111110
9	11110110
.	00000001

21

Design Example 2: Build a 8x8 keyboard (interrupted).

```
Readkey    proc  far
;  Push used registers
;  Wait for 20 ms

           mov  dx,1000h
again:     mov  row, -1
           clc
loop1:     rcr
           mov  al,row
           out  dx,al
           in   al,dx
           cmp  al,-1
           je   loop1

;  Extract keycode from (row)
;  and (al) and put it into the
;  global Keyboard Buffer

;  Pop used registers

           iret

Readkey    endp
```

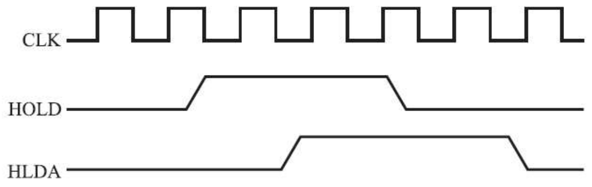


In main program, at the beginning:

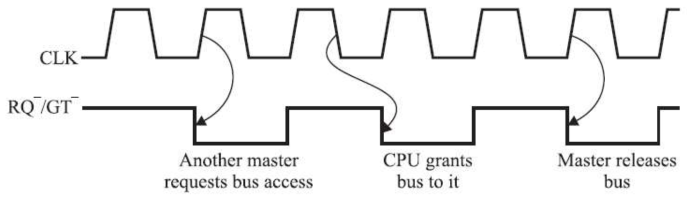
```
mov     ax,0
mov     es,ax
mov     es:[200h], offset Readkey
mov     es:[202h], seg Readkey
```

22

Taking the control of the bus



Bus request and bus grant timings in minimum mode system



Bus request and bus grant timings in maximum mode system

23

8087 Coprocessor

Introduced by Intel in 1980 to speed up computations for floating-point arithmetic, such as addition, subtraction, multiplication, division, and square root. It also realize exponential, logarithmic and trigonometric calculations, and besides floating-point numbers it could also operate on large binary and decimal integers.

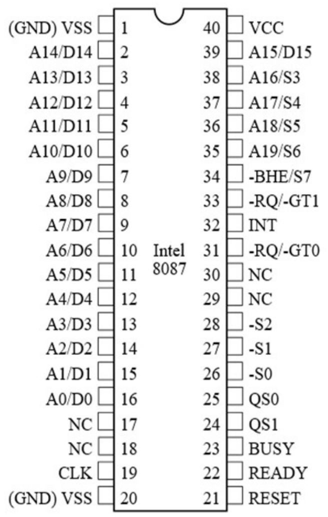
Main Features:

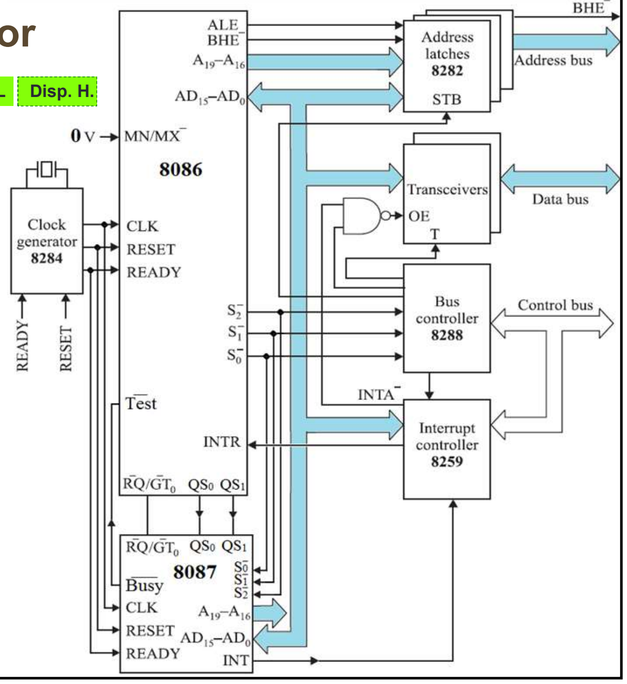
- Floating point (32-bit short, 64-bit long, 80-bit extended) numbers
- 18 digit decimal numbers
- Eight 80-bit internal registers

24

8087 Coprocessor

11011 xxx	Md	yyy	R/M	Disp. L	Disp. H.

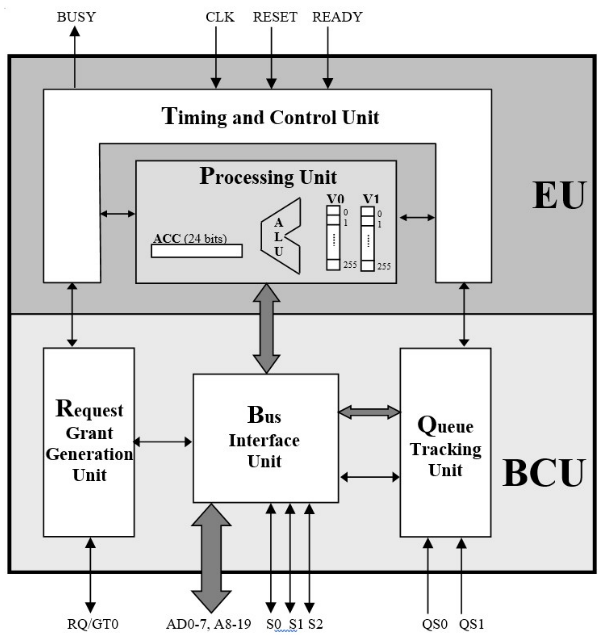




VSCOP:

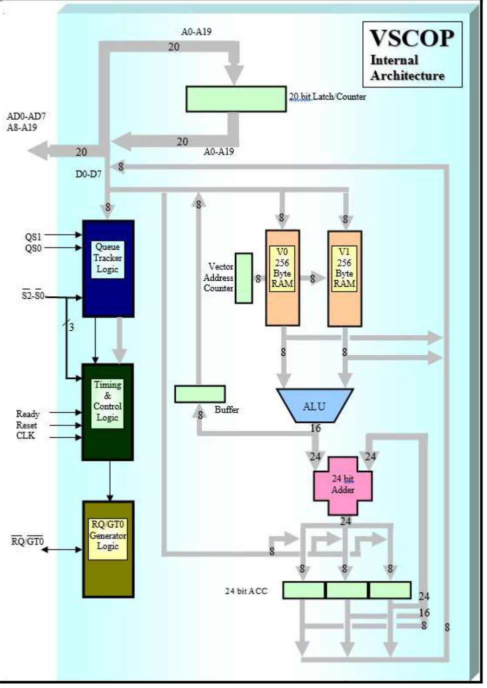
Vector-Signal Coprocessor

Functional block diagram of VSCOP



VSCOP:
Vector-Signal Coprocessor

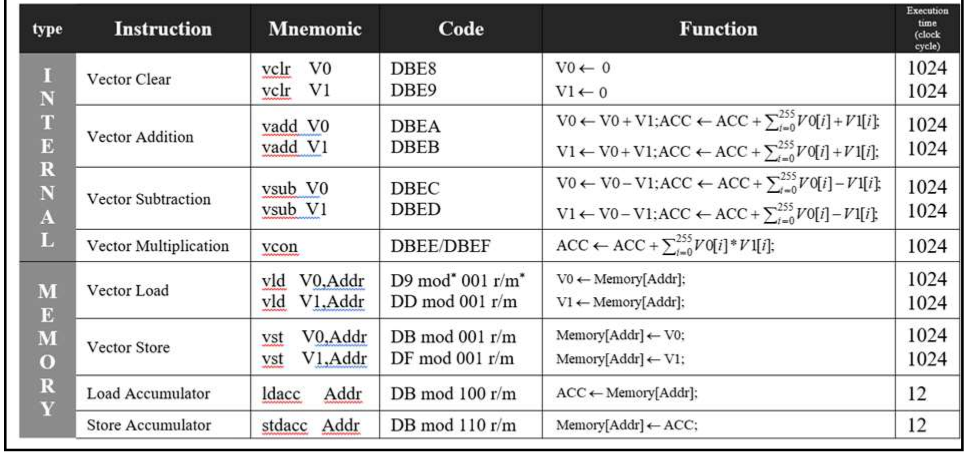
Internal Architecture



27

VSCOP: Vector-Signal Coprocessor

List of instructions



type	Instruction	Mnemonic	Code	Function	Execution time (clock cycle)
INTERNAL	Vector Clear	vclr V0 vclr V1	DBE8 DBE9	$V0 \leftarrow 0$ $V1 \leftarrow 0$	1024 1024
	Vector Addition	vadd V0 vadd V1	DBEA DBEB	$V0 \leftarrow V0+V1; ACC \leftarrow ACC+\sum_{i=0}^{255} V0[i]+V1[i];$ $V1 \leftarrow V0+V1; ACC \leftarrow ACC+\sum_{i=0}^{255} V0[i]+V1[i];$	1024 1024
	Vector Subtraction	vsub V0 vsub V1	DBEC DBED	$V0 \leftarrow V0-V1; ACC \leftarrow ACC+\sum_{i=0}^{255} V0[i]-V1[i];$ $V1 \leftarrow V0-V1; ACC \leftarrow ACC+\sum_{i=0}^{255} V0[i]-V1[i];$	1024 1024
	Vector Multiplication	vcon	DBEE/DBEF	$ACC \leftarrow ACC+\sum_{i=0}^{255} V0[i]*V1[i];$	1024

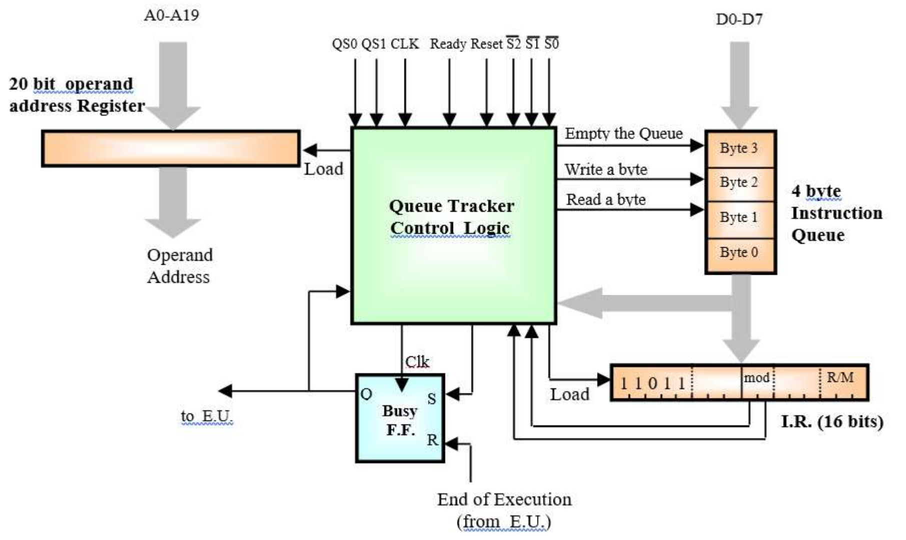
MEMORY	Vector Load	vld V0,Addr
		vld V1,Addr
			D9 mod* 001 r/m*
			DD mod 001 r/m
				V0 ← Memory[Addr];
				V1 ← Memory[Addr];
					1024
					1024

	Vector Store	vst V0,Addr
		vst V1,Addr
			DB mod 001 r/m
			DF mod 001 r/m
				Memory[Addr] ← V0;
				Memory[Addr] ← V1;
					1024
					1024

	Load Accumulator	ldacc Addr	DB mod 100 r/m	ACC ← Memory[Addr];	12
	Store Accumulator	stdacc Addr	DB mod 110 r/m	Memory[Addr] ← ACC;	12

28

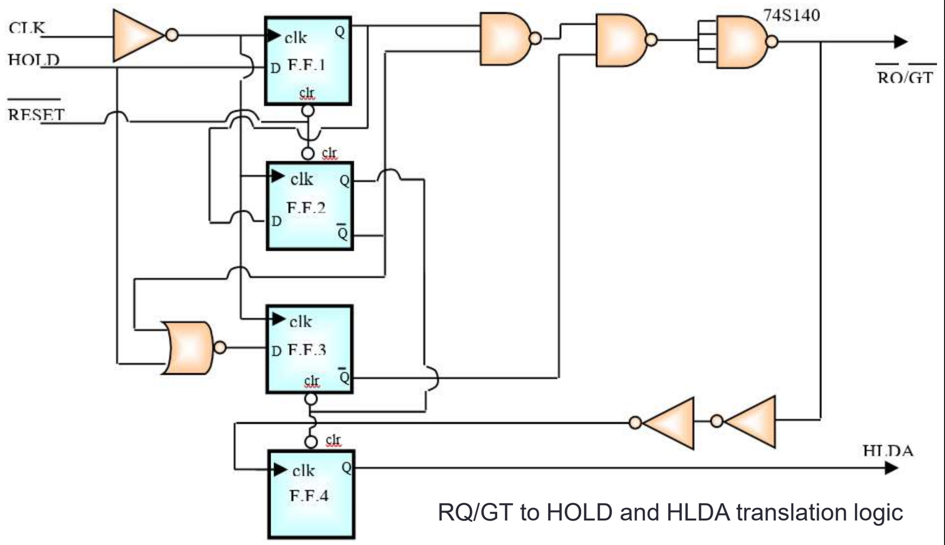
VSCOP: Vector-Signal Coprocessor



A block diagram of the logical circuit for copying the four-byte instruction queue in 8088

29

VSCOP: Vector-Signal Coprocessor



RQ/GT to HOLD and HLDA translation logic

30

(c) Hamid Sarbazi-Azad Computer Structure & Language, Lecture#9: Hardware and interfacing 31

VSCOP:

Vector-Signal Coprocessor

Discrete implementation using TTL MSI/LSI ICs

31

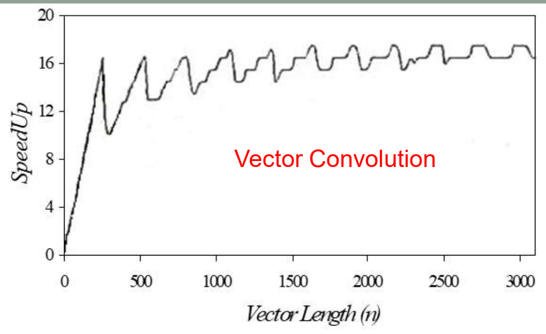
(c) Hamid Sarbazi-Azad Computer Structure & Language, Lecture#9: Hardware and interfacing 32

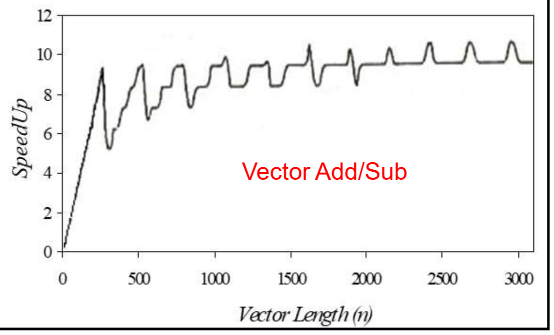
VSCOP:

Vector-Signal Coprocessor

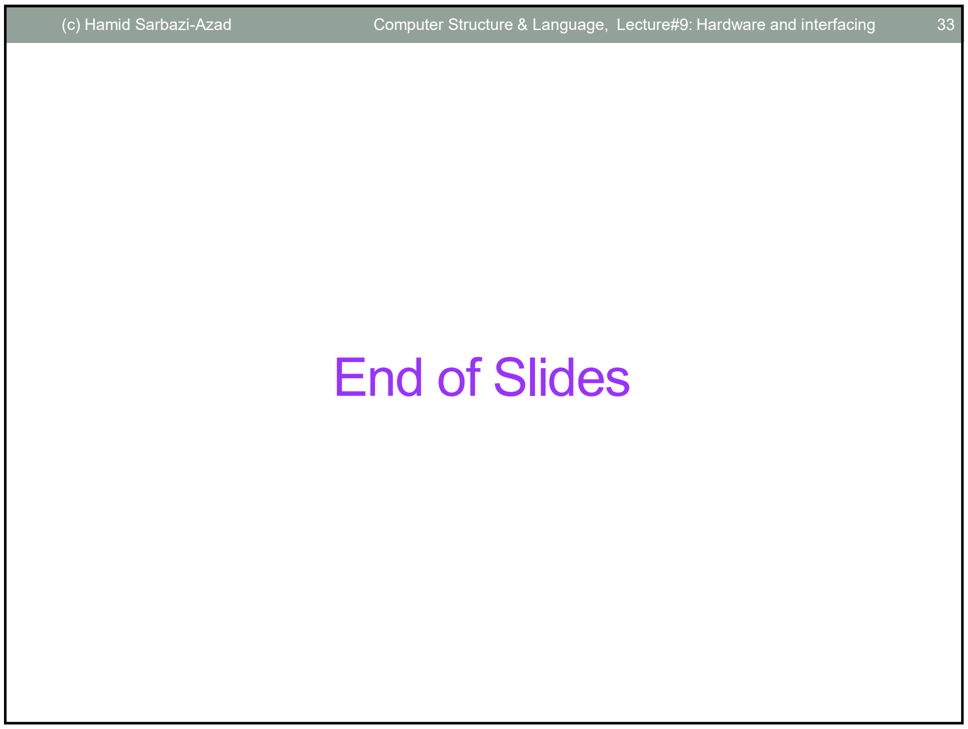
Performance Evaluation

$$SpeedUp(F,n) = \frac{T_{8088}(F,n)}{T_{8088+VSCOP}(F,n)}$$





32



End of Slides

33