3) I/O read
4) I/O write
5) Instruction fetch
6) Interrupt acknowledge
7) Halt / Shut down

M/$\overline{IO}$	D/$\overline{C}$	W/$\overline{R}$	Bus Cycle Type	Locked ?
Low	Low	Low	INTERRUPT ACKNOWLEDGE	Yes
Low	Low	High	Does not occur	–
Low	High	Low	I/O DATA READ	No
Low	High	High	I/O DATA WRITE	No
High	Low	Low	MEMORY CODE READ	No
High	Low	High	HALT : Address=2 (BE0- High BE1 High BE2 Low BE3 High A2-A31 Low SHUT DOWN : Address=0 (BE0 Low BE1 High BE2 High BE3 High A2-A31 Low)	No
High	High	Low	MEMORY DATA READ	Some cycles
High	High	High	MEMORY DATA WRITE	Some cycles

Table 13.6

In each bus cycles corresponding status signals are activated. The Table 13.6 shows the status signals along with the bus cycles. Table 13.6 also shows that memory read and memory write bus cycles can be locked to prevent another bus master from using the bus. Before going to see bus cycles it is necessary to know about bus states in 80386DX and system clock.

13.3.1 System Clock

System clock synchronizes the internal and external bus operations in the 80386DX. The 80386DX can operate on four different clock speeds : 80386DX - 16 (16 MHz), 80386DX - 20 (20 MHz), 80386DX - 25 (25 MHz) and 80386DX - 33 (33 MHz). Operating frequency of the 80386DX is half of the CLK2 frequency. Therefore, CLK2 of an 80386DX - 20 is driven by 40 MHz signal. Fig. 13.20 shows the CLK2 and internal clock signals.

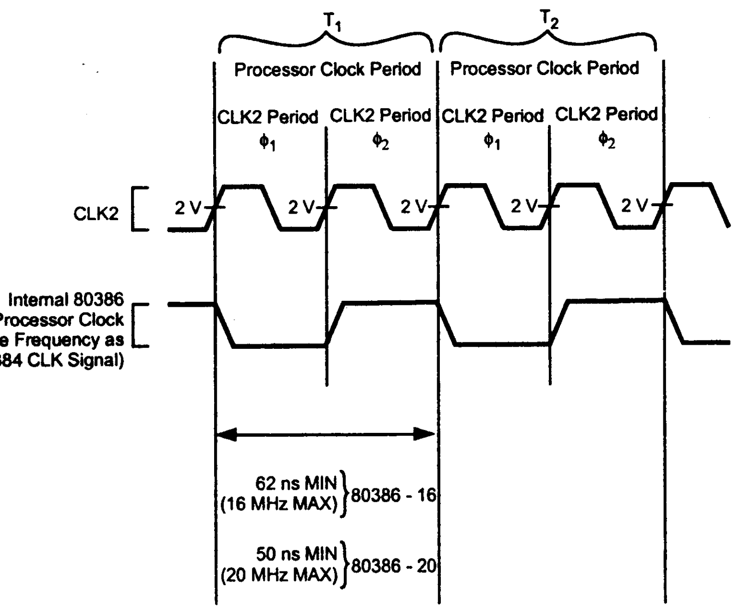

T1
T2
Processor Clock Period
Processor Clock Period
CLK2 Period
CLK2 Period
CLK2 Period
CLK2 Period
ϕ1
ϕ2
ϕ1
ϕ2
CLK2
2 V
Internal 80386 Processor Clock (Same Frequency as 82384 CLK Signal)
62 ns MIN (16 MHz MAX) 80386 - 16
50 ns MIN (20 MHz MAX) 80386 - 20

Fig. 13.20 CLK2 and internal clock signals

13.3.2 Bus States

Each bus cycle consists of atleast two bus states T1 and T2, and each bus state consists of two CLK cycles. During the first bus state (T1), address and bus status pin are active. During the second bus state (T2), actual data transfer takes place. The 30386 DX can perform two types of bus cycles : nonpipelined and pipelined.

13.3.3 Dynamic Bus Sizing

Dynamic data bus sizing is a feature allowing direct processor connection to 32-bit or 16-bit data buses for memory or I/O. A single processor may connect to both size buses. Transfers to or from 32 to 16 bit ports are supported by dynamically determining the bus width during each bus cycle.

The 80386DX microprocessor's bus size 16 ($\overline{\text{BS16}}$) input is used to inform the 80386DX that the currently addressed device is a 16-bit device rather than a 32-bit device. When $\overline{\text{BS16}}$ signal is activated, the 80386DX performs data transfers with the addressed device using only the lower two data bytes, (D7-D0) and (D15-D8). If the data is more than 16-bit microprocessor generates additional bus cycle to fulfill the bus transfer request if $\overline{\text{BS16}}$ is sampled asserted.

Let us consider the actions caused by the execution of the instruction MOV EAX, [4F04] and assume DS = 0000H, the processor is in real mode, and the addressed device is a 16-bit device. The MOV EAX, [4F04] instruction initiates the transfer of a double word (the EAX register is four bytes in size) starting at memory location 00004F04H. At the beginning of the first bus cycle, the 80386DX places address 00004F04H on the address bus

(A31-A0) and asserts all four byte enable outputs. When the currently addressed device's address decoder detects the address, it asserts $\overline{BS16}$, informing the 80386DX that the currently addressed device is only capable of communicating over the lower data bytes.

In the first bus cycle the contents of memory location 00004F04H is placed on D_7 - D_0 and the contents of memory location 00004F05H is placed on D_{15}:D_8 by the addressed 16-bit memory device. At the end of the bus cycle, when $\overline{READY}$ is sampled asserted, the 80386DX reads the two data bytes and loads them in the lower bytes of the EAX register.

The 80386DX then automatically begins a second bus cycle with the same address (00004F04H) on the address bus and $\overline{BE2}$ and $\overline{BE3}$ asserted. The addressed 16-bit device's address decoder once again asserts $\overline{BS16}$, and the contents of 00004F06H and 00004F07H memory locations are transferred to the 80386DX over the two lower data bytes.

At the end of the bus cycle, the 80386DX reads the two data bytes from the two lower data bytes and loads them into the upper two bytes of the EAX register, completing the overall transfer.

13.3.4 Nonpipelined Bus Cycles

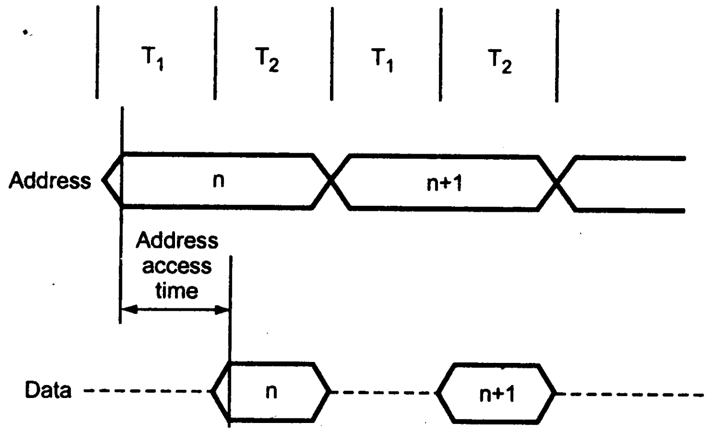


Fig. 13.21 Typical nonpipelined microprocessor bus cycle

Fig. 13.21 shows typical nonpipelined microprocessor bus cycle. During T1, the 80386DX sends the address, bus status signal and control signals. In case of write cycle, data to be output is also send on the data bus, during T1. As shown in the Fig.13.21, after address access time read or write data transfer takes place over the data bus. This activity is carried out in T2.

13.3.5 Pipelined Bus Cycle

Pipelining allows bus cycles to be overlapped. The main advantage of pipelining is that it increases the amount of time required for the memory or I/O device to respond. This time is also referred as access time. The 80386DX implements pipelining by overlapping addressing of the next bus cycle with the data transfer of previous bus cycle. Fig. 13.22 shows the pipelined bus cycles of 80386DX.

Fig. 13.22 shows that address becomes valid in T2 state of the previous bus cycle, and the data transfer for address takes place in T2-state of the current bus cycle. It is important to note that the address An + 1 becomes valid during T2 of the current bus cycle and actual data transfer for address An + 1 takes place in T2 state of the next bus cycle. If the processor is 80386DX-20 then one T-state time is 50ns. In pipelined bus cycle the access time for memory and I/O device is 100ns whereas access time for memory and I/O device in nonpipelined bus cycle is approximately 50ns.

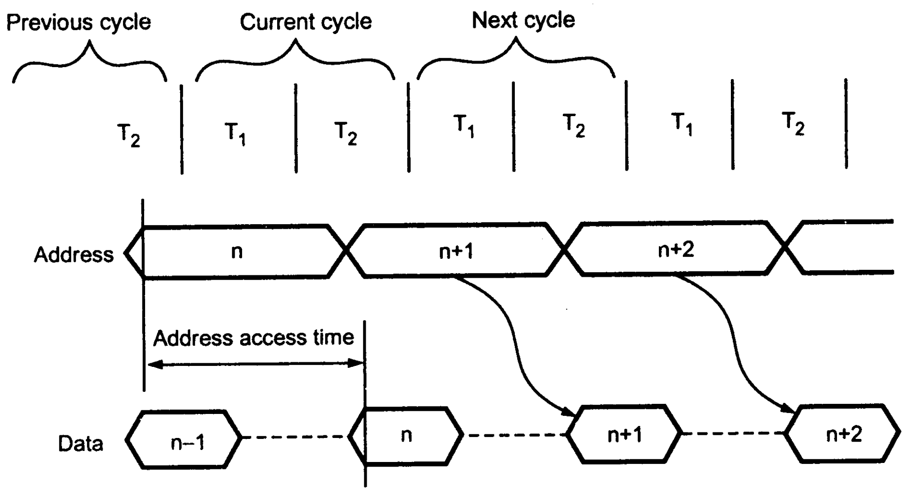


Fig. 13.22 Typical pipelined bus cycle of 80386DX

13.3.6 Idle State in Pipelined Bus Cycle

Fig. 13.23 shows the idle bus state.

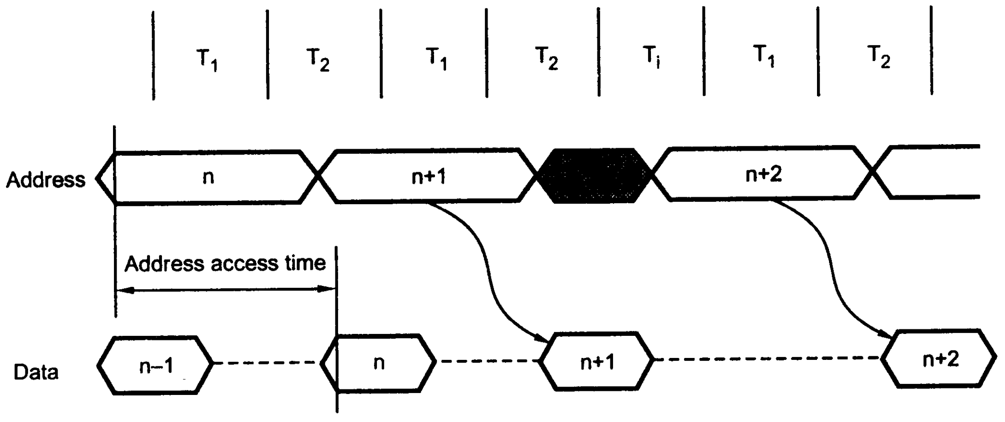


Fig. 13.23 Bus idle state

In the pipelined bus cycle, we have seen that addressing of next cycle is overlapped with the data transfer of the current cycle. But in some situations such as prefetch queue is full and the instruction that is currently being executed does not require to access operands in memory or I/O device, no bus activity will take place. In such situations bus enters into a state called idle state.

13.3.7 READ and WRITE Bus Cycles

In the previous sections, we have seen some basic concepts of memory interfacing, which includes : 80386 DX memory interfacing signals, bus cycle states, and pipelined and non pipelined bus cycles. In this section we will see the sequence of events that take place during the 80386DX memory read and write bus cycles.

Nonpipelined Read Cycle

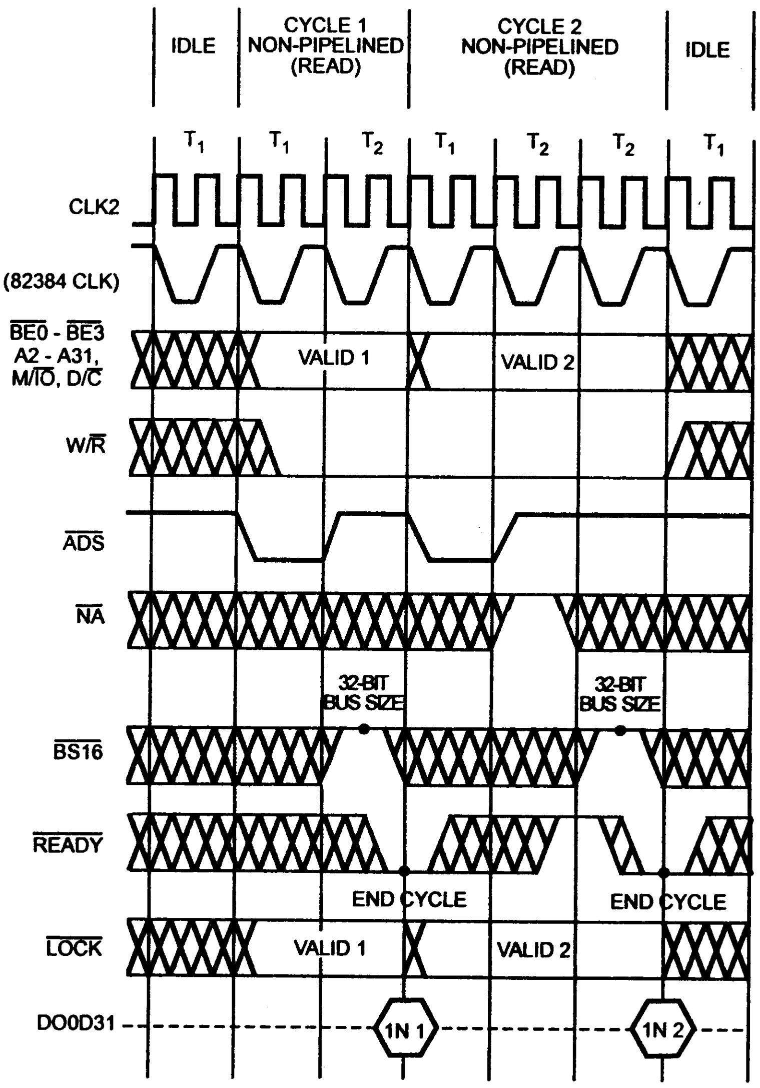


Fig. 13.24 Nonpipelined read cycle

Fig. 13.24 shows the timings for two nonpipelined read cycles (with and without a wait state). First read cycle is without wait state and second cycle is with wait state.

The sequence of events for the nonpipelined read cycle is as follows :

The read operation starts at the beginning of phase in the T1 state of the bus cycle. In this phase, 80386DX sends the address on the address bus and enables signals ($\overline{\text{BE0}}$ - $\overline{\text{BE3}}$) according to data transfer type. After sending the address, in the same phase, 80386 DX activates its $\overline{\text{ADS}}$ signal to indicate valid address is placed on the address bus. In phase 1 of T1 - state 80386DX also activates the bus cycle definition signals : M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$ and W/$\overline{\text{R}}$. For read cycle W/$\overline{\text{R}}$ is low. M/$\overline{\text{IO}}$ is high for memory read and low for an I/O read. D/$\overline{\text{C}}$ signal differentiate between data and instruction code. This signal is high if data is to be read and low if an instruction code is to be read. At the end of phase 2 of T1 - state, $\overline{\text{ADS}}$ is returned to its inactive logic 1 state. The address bus, byte enable pins, and bus status pins remain active through the end of the read cycle.

At the beginning of phase 1 of T2 state external device activates BS16 signal. The 80386DX samples this signal in the middle of phase 1 of T2-state. If this signal is high, 80386DX does the 32-bit data transfer; otherwise 80386 DX performs 16-bit data transfer. The 80386 DX does this data transfer in phase 2 of T2-state.

At the end of phase 2 of T2-state the $\overline{\text{READY}}$ signal is sampled by the 80386DX. The 80386DX. The logic 1 on this signal inserts wait state in the current bus cycle to extend the bus cycle. In wait state (Tw), the signals from T2-state are maintained throughout the wait state period. It just a repetition of T2-state. Thus the period of one wait state (Tw = T2) is equal to 50ns of 20 MHz clock operation. If this signal is low, 80386 DX proceeds with next bus cycle.

The $\overline{\text{LOCK}}$ signal low indicates it is bus locked cycle. If bus cycles are locked the other bus master is not allowed to take control of the bus between two locked bus cycles.

13.3.8 Nonpipelined Write Cycle

Fig. 13.25 (see Fig. 13.25 on next page) shows the timings for two nonpipelined write cycles (with and without a wait state) first write cycle is without wait state and second cycle is with wait state.

The sequence of events for the nonpipelined write cycle is as follows :

- The nonpipelined write cycle is similar to nonpipelined read cycle. The write operation starts at the beginning of phase 1 in the T1 state of the bus cycle. In this phase, 80386DX sends the address on the address bus and enables signals $\overline{\text{BE0}}$ - $\overline{\text{BE3}}$ according to data transfer type. After sending address in the same phase. 80386DX activates its $\overline{\text{ADS}}$ signal to indicate valid address is placed on the address bus. In phase 1 of T1-state 80386DX also activates the bus cycle definition signals : M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$ and W/$\overline{\text{R}}$. For write cycle W/$\overline{\text{R}}$ is high. M/$\overline{\text{IO}}$ is high for memory and low for I/O write. D/$\overline{\text{C}}$ signal is high.

- At the beginning of phase 2 of T1-state, 80386DX sends data on the data bus. This data remains valid until the start of phase 2 of the T1-state of the next bus cycle.

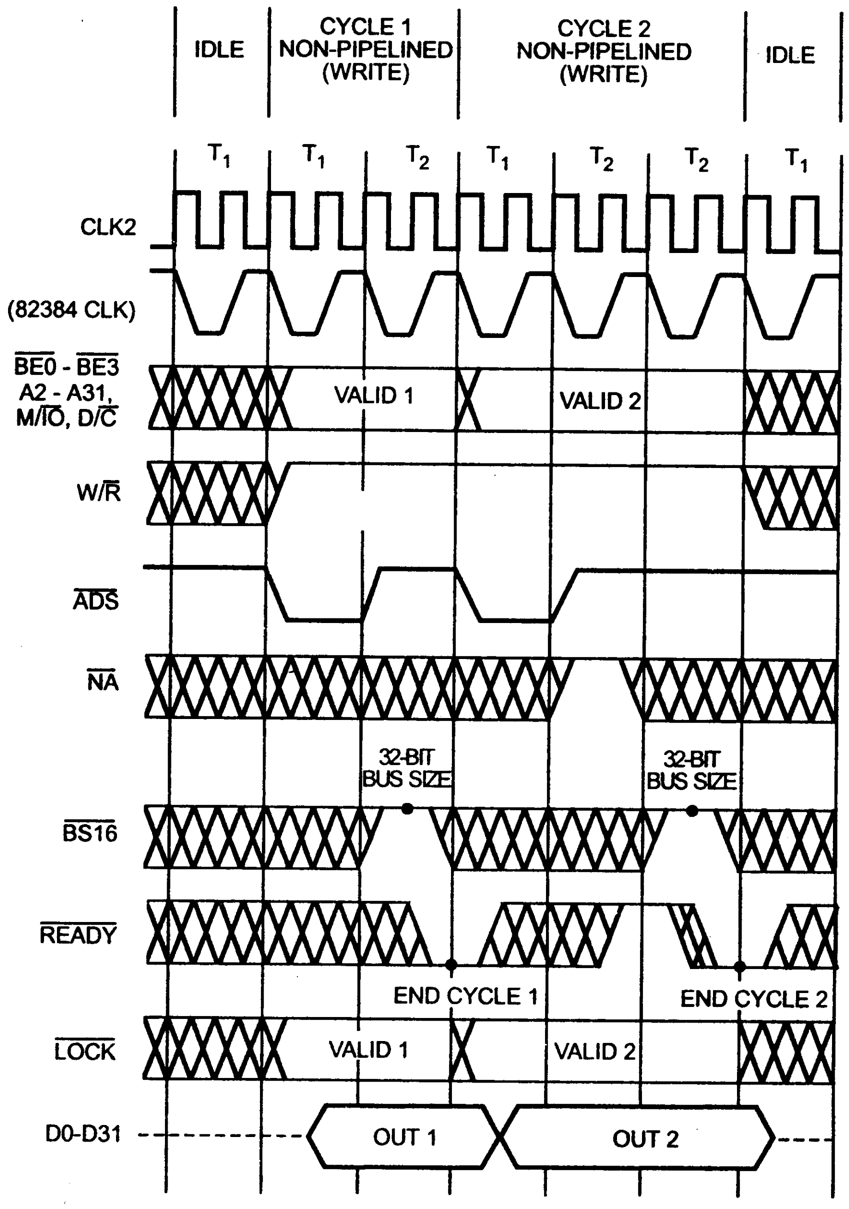


Fig. 13.25 Nonpipelined write cycle

- At the end of phase 2 of T1 - state, $\overline{\text{ADS}}$ is returned to its inactive logic 1 states. The address bus, byte enable pins, and bus status pins remain active through the end of the write cycle.
- In the middle of phase 1 of T2-State, 80386DX samples $\overline{\text{BS16}}$ input. If this signal is high, 80386 DX does the 32-bit data transfer; otherwise 80386DX performs 16-bit data transfer.
- At the end of phase 2 of T2-state the $\overline{\text{READY}}$ signal is sampled by the 80386DX. The logic 1 on this signal inserts wait state in the current bus cycle to extend the bus cycle. In wait state (Tw), the signals from T2-state are maintained throughout the wait state period. It just a repetition of T2-state. Thus the period of one wait state (Tw - T2) is equal to 50ns of 20 MHz clock operation. If this signal is low, 80386DX proceeds with next bus cycle.

13.3.9 Pipelined Read/Write Cycle

As mentioned earlier, address pipelining allows bus cycles to be overlapped, increasing the amount of time available for the memory or I/O device to respond. Fig. 13.26 shows both nonpipelined and pipelined read and write cycles. The cycle 1 and cycle 2 in the

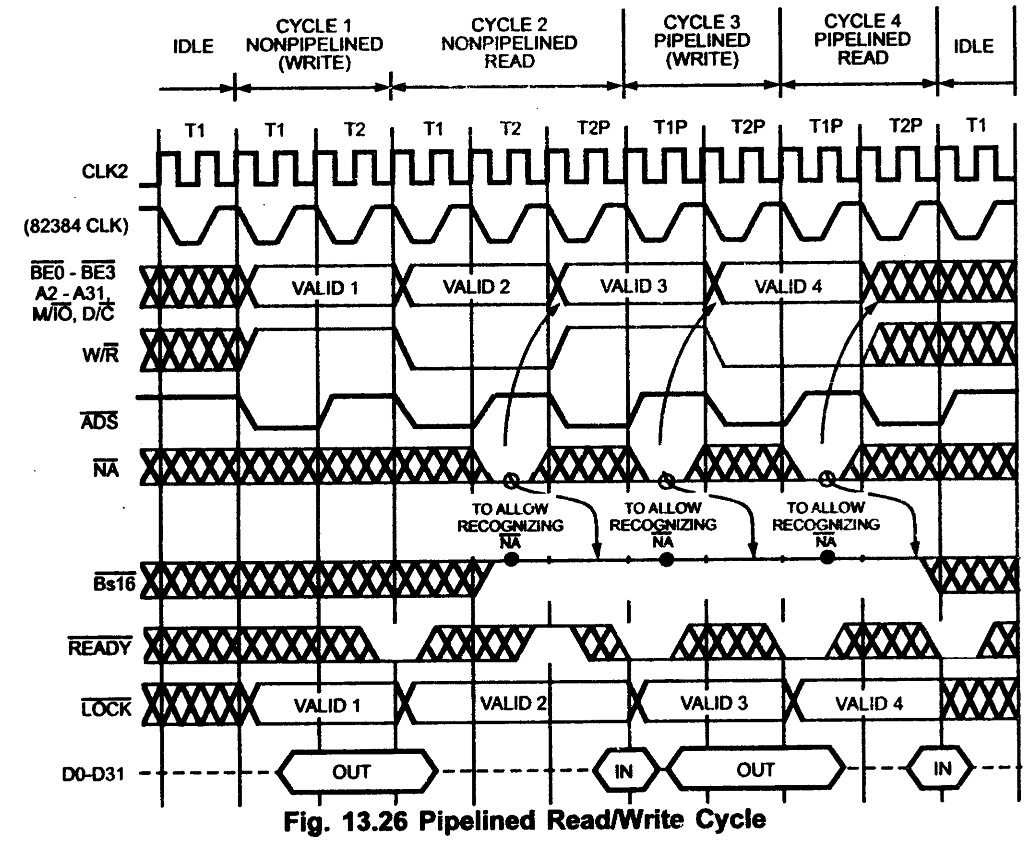


Fig. 13.26 Pipelined Read/Write Cycle

diagram show nonpipelined write and read cycles, respectively, whereas cycle 3 and cycle 4 in the diagram show pipelined write and read cycles, respectively. This diagram also shows how wait state can be avoided using pipelined bus cycle.

In the pipelined bus cycle the address for the next bus cycle is sent during the T2-state of the current cycle. In 80386DX, $\overline{\text{NA}}$ (next address) signal initiates address pipelining. The 80386DX samples $\overline{\text{NA}}$ signal at the beginning of phase 2 of any T state in which $\overline{\text{ADS}}$ is not active, specifically.

- In the second T-state of a nonpipelined address cycle.
- In the first T-state of a pipelined address cycle.
- In any wait state of a nonpipelined address or pipelined address cycle unless $\overline{\text{NA}}$ has already been sampled active.

In Fig. 13.26 $\overline{\text{NA}}$ is tested as 0 (active) during T2 of cycle 2 which ensures that 80386DX has to execute next cycle as pipelined bus cycle. The cycle 2 (nonpipelined read cycle) is also extended with one wait state because $\overline{\text{READY}}$ pin is not active, in wait state, the valid address for the next bus cycle is sent on the address bus as next bus cycle is pipelined bus cycle.

The next cycle (cycle 3) is pipelined write cycle. In this, data is sent on the data bus in phase 2 of T1p-state and remains valid for the rest of the cycle. The $\overline{\text{READY}}$ signal is sampled at the end of T2p - state. As it is low, write cycle is completed without wait state. Fig. 13.26 shows $\overline{\text{NA}}$ is active during T1p of cycle 3, which ensures that 80386DX has to execute next cycle as pipelined bus cycle.

The next cycle (cycle 4) is pipelined read cycle. In this, $\overline{\text{READY}}$ signal is tested 0 at the end of phase 2 of T2p - state. This means that read cycle is completed without wait state. It is important to note that due to pipelined address cycle access time is extended and one state (T-wait) of read cycle is saved.

13.3.10 Interrupt Acknowledge Cycle

In response to INTR signal, 80386 DX executes interrupt acknowledge bus cycle to read an interrupt type from the external device (8259A programmable interrupt controller). The interrupt acknowledge cycle is a special cycle designed to activate $\overline{\text{INTA}}$ signal for 8259A interrupt controller. Fig. 13.27 shows the interrupt acknowledge bus cycle. Looking at Fig. 13.27, we find that there are two extended interrupt acknowledge cycles separated by four idle T-states. The extended cycles and four idle T-states are required to satisfy minimum pulse width requirements of the 8259A.

The interrupt acknowledge cycle starts at the beginning of phase 1 in the T1 of the bus cycle. In this phase, 80386DX sets address lines A3 through A31 and $\overline{\text{BE0}}$ to logic 0 and A2 and BE1 through BE3 to logic 1. In the same phase, 80386 DX also sets M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$ and W/$\overline{\text{R}}$ signal to logic 0. These signals are latched into external circuitry with using $\overline{\text{ADS}}$ signal. The bus control logic decodes code 000 (M/$\overline{\text{IO}}$ = 0 D/$\overline{\text{C}}$ = 0 and W/$\overline{\text{R}}$ = 0) to generate an $\overline{\text{INTA}}$ signal. In the second interrupt acknowledge cycle same signals are

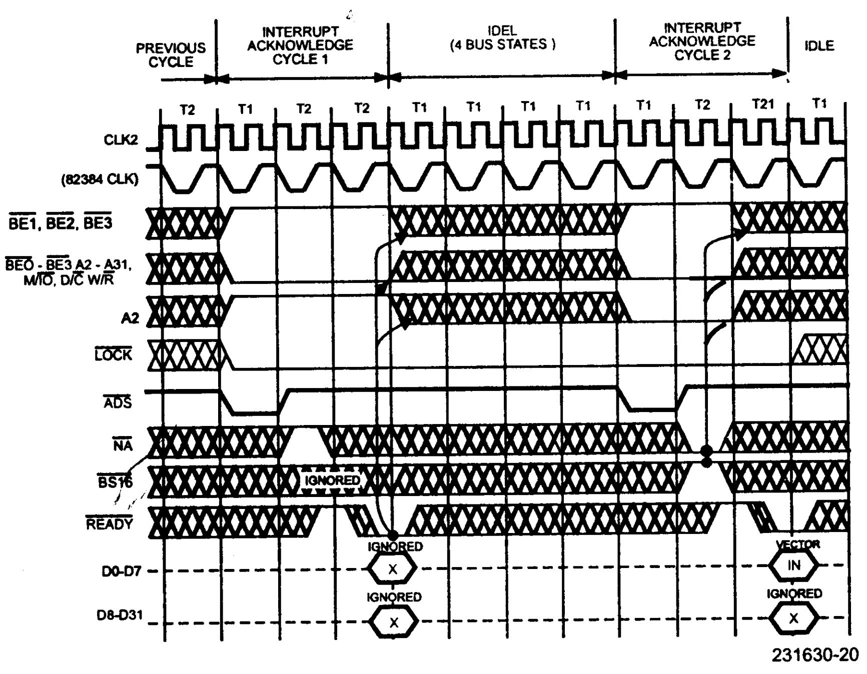


Fig. 13.27 Interrupt acknowledge cycle

activated, excepts A2 address is logic 0 instead of logic 1. The 80386DX activates $\overline{LOCK}$ signal from the beginning of the first cycle to the end of second cycle. This prevents other master to take control of bus before the completion of second interrupt acknowledge cycle.

The 80386DX floats D31 - D0 for both bus cycles; however in the phase 2 of T2-state of the second interrupt acknowledge cycle, 80386 DX reads the interrupt type number from data bus (D7 - D0), placed by the interrupt controller (8259A).

13.3.11 HALT/Shutdown Cycle

In response to a HLT instruction, 80386 goes into halt condition. The shutdown condition occurs when the 80386 is processing a double fault and encounters a protection fault.

Similar to other bus cycles, a halt and shutdown cycles are initiated by activating $\overline{ADS}$ and the bus status pin as given below.

- $M/\overline{IO}$, $W/\overline{R}$ are driven high, and $D/\overline{C}$ is driven low to indicate halt cycle.
- All address lines are set to logic 0.
- For a halt condition, $\overline{BE2}$ is active and for a shutdown condition, $\overline{BE0}$ is active. These signals are used by external devices to identify the halt and shutdown cycles.

EXIT from HALT or Shutdown

The 80386 will remain in the halt or shutdown condition until

1) NMI goes high 2) RESET goes high

This means that when 80386 services NMI interrupt or if 80386 is reinitialized, it is possible to exit from halt or shutdown condition. When 80386 service INTR interrupt, it is possible to exit from halt condition but not from shut down condition.

NOTE : The 80386 can service coprocessor request (PEREQ input) and HOLD (Hold input) request in the halt or shutdown condition.

13.3.12 Control Input BS16

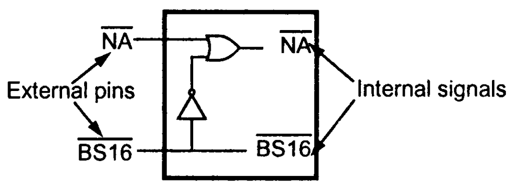


Fig. 13.28 Internal $\overline{\text{NA}}$ and $\overline{\text{BS16}}$ Logic

As mentioned earlier $\overline{\text{BS16}}$ signal specifies bus size for each bus cycle. The 80386DX· samples this signal at the beginning of phase 2 of T-state in which $\overline{\text{ADS}}$ is not active. When $\overline{\text{BS16}}$ is low the 80386 performs 16-bit data transfers using D15-D0 data lines rather than 32-bit data bus. The 80386 automatically performs two or three bus cycles for data transfers more than 16-bits and misaligned (odd-addressed) 16-bit transfers.

Fig. 13.28 shows the internal $\overline{\text{NA}}$ and $\overline{\text{BS16}}$ logic. Looking at Fig. 13.28, we find that eventhough $\overline{\text{NA}}$ and $\overline{\text{BS16}}$ signals are low on the external pins, internally, these signals can't be low at the same time. To implement address pipelining for 16-bit cycles, $\overline{\text{BS16}}$ must be active at the same time as $\overline{\text{NA}}$, which is impractical. Because 80386 samples $\overline{\text{BS16}}$ and $\overline{\text{NA}}$ simultaneously and as said earlier these signals can't be active (low) at the same time.

Fig. 13.29 shows how 80386 performs multiple (in this case two) bus cycles for data transfers larger than 16-bit (in this case 32-bit). (Refer Fig. 13.29 on next page.)

13.3.13 Bus Lock

Because more than one processor shares the system memory and I/O devices through a common system bus, extra logic must be added to ensure that only one processor has access to the system bus at a time. When one processor starts executing the program which uses the common resources, another processor is not allowed to access the used resources. This program region where the common resources are used is called critical program region. One can restrict another processor from accessing the used resources by using a technique, called Mutual exclusion. In this technique, a binary flag called semaphore is stored in the shared memory to indicate where the shared memory is free (semaphore = 1) to be accessed or busy.
(semaphore = 0). Testing and setting the semaphore is a critical operation and it must be performed by one processor at a time. This mutual exclusion technique can be

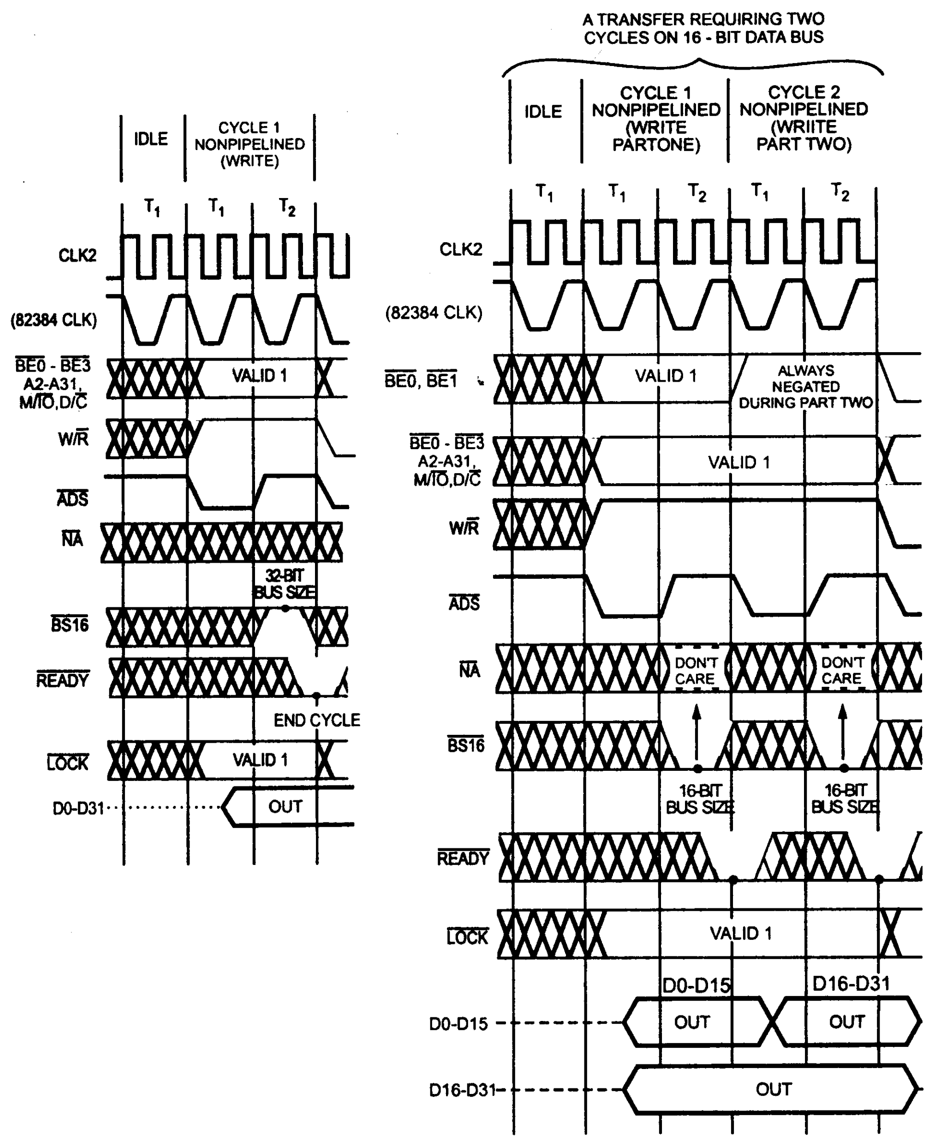

IDLE
CYCLE 1 NONPIPELINED (WRITE)
A TRANSFER REQUIRING TWO CYCLES ON 16 - BIT DATA BUS
CYCLE 1 NONPIPELINED (WRITE PARTONE)
CYCLE 2 NONPIPELINED (WRIITE PART TWO)
T1
T2
CLK2
(82384 CLK)
BE0 - BE3 A2-A31, M/IO, D/C
BE0, BE1
W/R
ADS
NA
BS16
READY
LOCK
D0-D31
D0-D15
D16-D31
VALID 1
ALWAYS NEGATED DURING PART TWO
32-BIT BUS SIZE
16-BIT BUS SIZE
DON'T CARE
END CYCLE
OUT

Fig. 13.29 Timings for 16-bit and 32-bit bus cycles

implemented in 8086, 68000 and many other processors which support the multiprocessor environment.

Semaphore implementation in 80386 :

In 80386 the XCHG instruction along with the LOCK prefix can be used to set or reset semaphore.

Program sequence :

```
                    MOV AL, 0
LOCK checkagain :   XCHG semaphore, AL
                    TEST AL, AL
                    JZ checkagain
```

```
●  | Critical region in which program
●  |  accesses the shared resources
●  |
```

MOV Semaphore, 1

The XCHG semaphore, AL instruction exchanges the contents of the AL register with the contents of the memory location in which semaphore is stored. The XCHG instruction requires two bus cycles. During this XCHG instruction, it is necessary to restrict other processor from accessing semaphore. This is achieved by LOCK prefix in the 80386. LOCK prefix activates the $\overline{\text{LOCK}}$ output pin during the execution of the instruction that follows the prefix. During the execution of XCHG instruction, the $\overline{\text{LOCK}}$ output pin is in the active state which does not allow other processor to get control of the system bus. After execution of the XCHG instruction, the status of the semaphore is available in the AL register. By checking the contents of the AL register, it is possible to decide whether to enter into critical region of program or to check semaphore again.

13.3.14 HOLD/HLDA

As mentioned earlier, in multiprocessor systems more than one processor shares the system resources which are accessed through common bus. To gain the control of this common bus and system resources, the requesting processor activates the 80386 HOLD input. On receiving HOLD signal, 80386 outputs HLDA signal high as an acknowledgment, after completing its current bus cycle (plus a second locked cycle or a second cycle required by $\overline{\text{BS16}}$). At the same time, 80386 tri-states all the outputs except HLDA to effectively remove itself from the bus and then requesting processor takes the control of system bus. A low on HOLD gives the system bus control back to the 80386.

During the HOLD state, the 80386 can continue executing instruction in its Prefetch Queue.

Eventhough HOLD has higher priority over most bus cycles, but HOLD is not recognized :

1) Between two interrupt acknowledge cycles.
2) Between two repeated cycles of a BS16 cycles
3) During locked cycles.
4) During active RESET.

13.4 Memory Interface

The 80386 is a high speed microprocessor. For any microprocessor system, system performance depends on the microprocessor as well as performance of memory system. In 80386 systems, overall system performance depends on the performance of memory system. As you know, most of the bus cycles require access of memory to read program instruction and to read or write data into the memory. If memory system is slow acting then microprocessor has to increase access time by introducing wait states. This reduces the overall system performance.

13.4.1 Basic Memory Interface

Fig. 13.30 shows the basic memory interface. The bus control logic block provides control signals for the address latches (74×373), data transceivers (74×245), and the memory devices. It also sends the active low $\overline{\text{READY}}$ signal to indicate that the memory device is ready to perform new bus cycle. $\overline{\text{NA}}$ signal from bus controller informs 80386 that its next bus cycle should be pipelined bus cycle. The address decoder decodes the address to generate chip select signals, $\overline{\text{BS16}}$ signal, and bus control logic input.

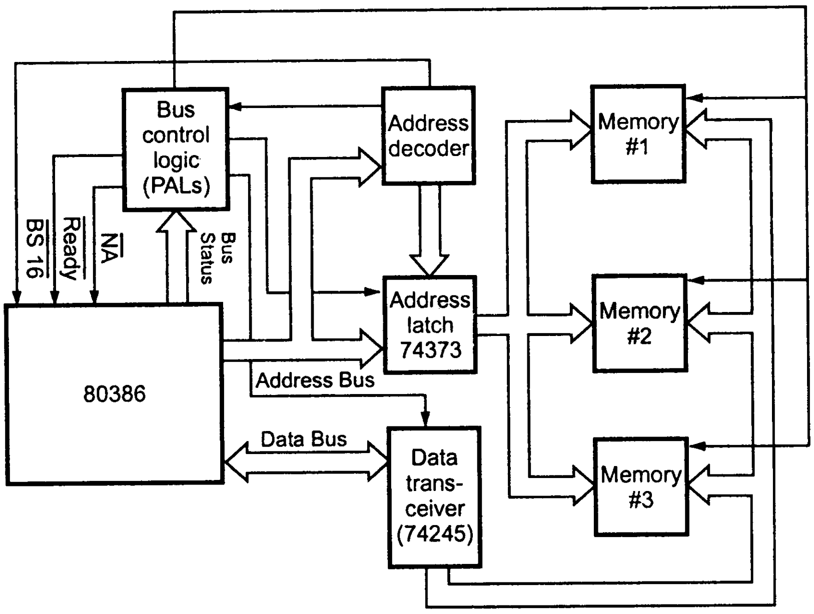


Fig. 13.30 Basic memory interface

To generate these signals bus controller logic decodes the 80386 status outputs (W/$\overline{\text{R}}$, M/$\overline{\text{IO}}$, and D/$\overline{\text{C}}$). The signals generated by Bus Controller for memory device and I/O device are as follows :

$\overline{\text{MRDC}}$ (Memory Read Command) : This signal is a combination of memory data read and memory code read. This signal is used to instruct memory to put the contents of addressed memory location on the data bus.

$\overline{\text{MWTC}}$ (Memory Write Command) : This signal is used to instruct memory to accept the data from the data bus and load the data into the addressed memory location.

$\overline{\text{IORC}}$ (I/O Read Command) : This signal is used to instruct an I/O device to put the data contained in the addressed port on the data bus.

$\overline{\text{IOWC}}$ (I/O Write Command) : This signal is used to instruct an I/O device to accept the data from the data bus and load the data into the addressed port.

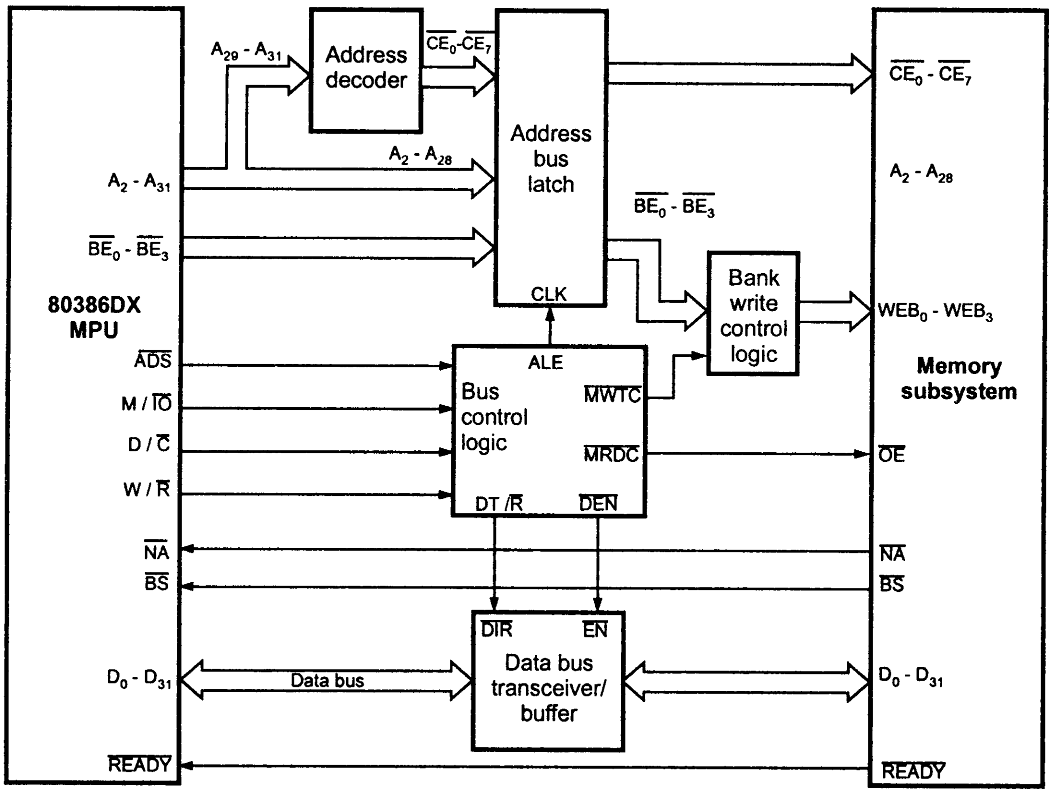


Fig. 13.31 Memory interface with signals

$\overline{\text{INTA}}$ (Interrupt Acknowledge) : In the interrupt acknowledge bus cycle this signal is activated two times. First $\overline{\text{INTA}}$ is used as an acknowledgment whereas second $\overline{\text{INTA}}$ is used to instruct interrupt controller to put the interrupt type on the data bus.

Fig. 13.31 shows the basic memory interfacing with signals generated by the bus controller. Before going to see details of memory interfacing, it is necessary to see the memory organization in 80386 systems, for better understanding.

13.4.2 Memory Organization

The 80386DX has 32-bit address bus so it can access upto 4G-bytes (2^{32}) of memory locations. Fig. 13.32 shows the physical address space. From the software point of view this memory is organized over the address range from 00000000H through FFFFFFFFH and 80386DX can access data in this memory as byte, word or double words. The words are accessed from two consecutive memory locations whereas double words are accessed from four consecutive memory locations. To implement this entire memory is divided into four independent byte-wide memory banks: Bank0-Bank3. Each bank is 1G-byte in size.

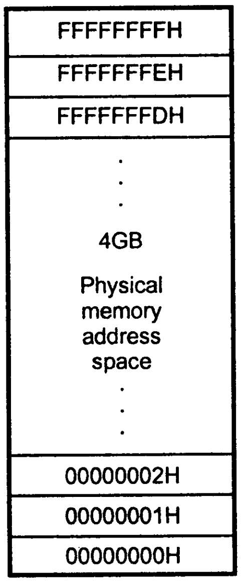


Fig. 13.32 Physical address space

Fig. 13.33 shows the organization of four memory banks. As shown in Fig. 13.32 bank 0, bank 1, bank 2 and bank 3 are selected using byte enable signals $\overline{BE0}$, $\overline{BE1}$, $\overline{BE2}$ and $\overline{BE3}$ signals, respectively. Address lines A31-A2 are connected in parallel to all memory banks which make it possible to access 1 Gbyte of memory. But the 32-bit data bus is distributed over four memory banks, 8-bit each. The 80386 uses byte enable signals instead of the two least significant address bits, because 80386 has problems involved in addressing more than one byte of memory at a time. When 80386 accesses word from even address, it uses two consecutive memory locations for example,

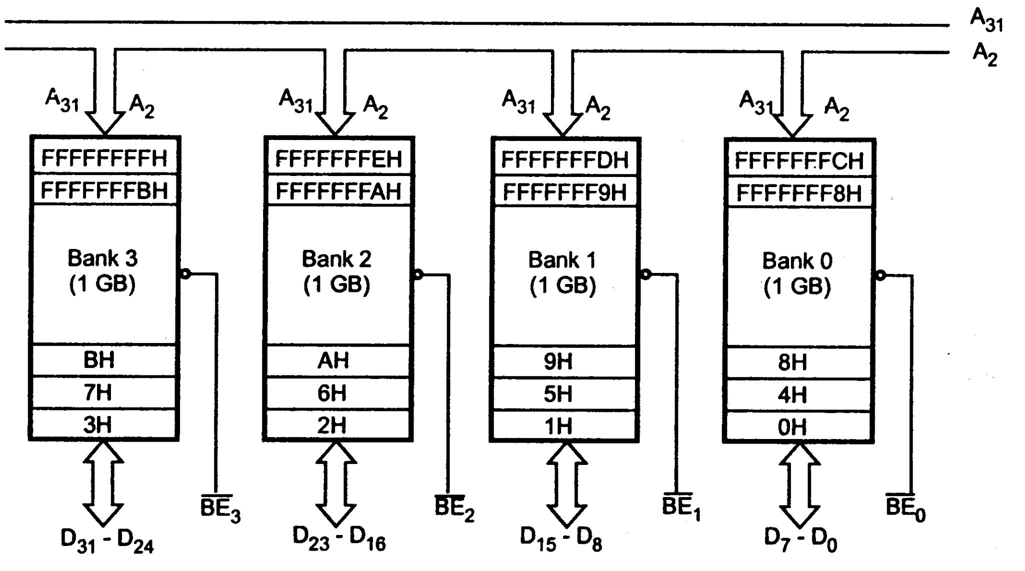


Fig. 13.33 Organization of four memory

MOV WORD PTR DS : [2000H], 5678H This instruction writes 78 to address 2000H and 56 to address 2001H.

Similarly, when 80386 accesses Dword from address divisible by 4, it uses four consecutive memory locations. This works fine when 80386 accesses even byte in case of word access and address divisible by 4 in case of Dword access, since address on the A31–A2 lines is same for word as well as Dword access. The data transfers using such addresses are called Aligned transfers. But to access word from odd address or to access Dwords from address not divisible by 4 unaligned 80386 faces problem in placing the correct address on the address bus. This problem is solved by replacing two address pins A0 and A1 with four byte enable pins. Without the two least significant address pins, the 80386 produces only addresses that are even multiples of 4. To distinguish between four addresses (four banks) byte enable signals are used. Table 13.7 shows the use of the four byte enables pins with the address pins.

Desired Address	A31 through A02	BE0	BE1	BE2	BE3
	Single-Byte Transfers				
00002000	00002000	On	Off	Off	Off
00002001	00002000	Off	On	Off	Off
00002002	00002000	Off	Off	On	Off
00002003	00002000	Off	Off	Off	On
00002004	00002004	On	Off	Off	Off
00002005	00002004	Off	On	Off	Off
	Double-Byte (Word) Transfers				
00002000	00002000	On	On	Off	Off
00002001	00002000	Off	On	On	Off
00002002	00002000	Off	Off	On	On
00002004	00002004	On	On	Off	Off
00002005	00002004	Off	On	On	Off
	Quad-Byte (Dword) Transfers				
00002000	00002000	On	On	On	On
00002004	00002004	On	On	On	On

Table 13.7

Unaligned transfers

Desired Address	A31 through A02	BE0	BE1	BE2	BE3
	Double-Byte (Word) Transfers				
00002003	00002004	On	Off	Off	Off
	00002000	Off	Off	Off	On
	Quad-Byte (Dword) Transfers				
00002001	00002004	On	Off	Off	Off
	00002000	Off	On	On	On
00002002	00002004	On	On	Off	Off
	00002000	Off	Off	On	On
00002003	00002004	On	On	On	Off
	00002000	Off	Off	Off	On
00002005	00002008	On	Off	Off	Off
	00002004	Off	On	On	On

13.4.3 Interfacing with Different Memory Types

In this section, we are going to see interfacing of different memory ICs with 16-bit bus and 32-bit bus.

SRAM Interface

16 bit Bus : For 16-bit memory interface, it is necessary to generate $\overline{BHE}$, $\overline{BLE}$ and A1 signals, since 16-bit physical word begins at an address that is a multiple of 2. To generate $\overline{BHE}$, $\overline{BLE}$ and A1 signals, $\overline{BE0}$-$\overline{BE3}$ are decoded as shown in Table 13.8.

Intel386TM DX Signals				16-Bit Bus Signals			Comments
BE3	BE2	BE1	BE0	A1	BHE	BLE(A0)	
H*	H*	H*	H*	X	X	X	X-no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	X	X	X	X-not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	X	X	X	X-not contiguous bytes
L*	H*	L*	H*	X	X	X	X-not contiguous bytes
L*	H*	L*	L*	X	X	X	X-not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	X	X	X	X-not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE asserted when D0-D7 of 16-bit bus is active.

BHE asserted when D8-D15 of 16-bit bus is active.

A1 low for all even words; A1 high for all odd words.

Key : X = Don't care

H = High voltage level; L = Low voltage level

* = a non-occurring pattern of byte enables; either none are asserted, or the pattern has byte enables asserted for non-contiguous bytes.

Table 13.8 Generating $\overline{BHE}$, $\overline{BLE}$ and A1 for addressing 16-bit devices

Fig. 13.34 shows the logic to generate $\overline{BHE}$, $\overline{BLE}$ and A1.

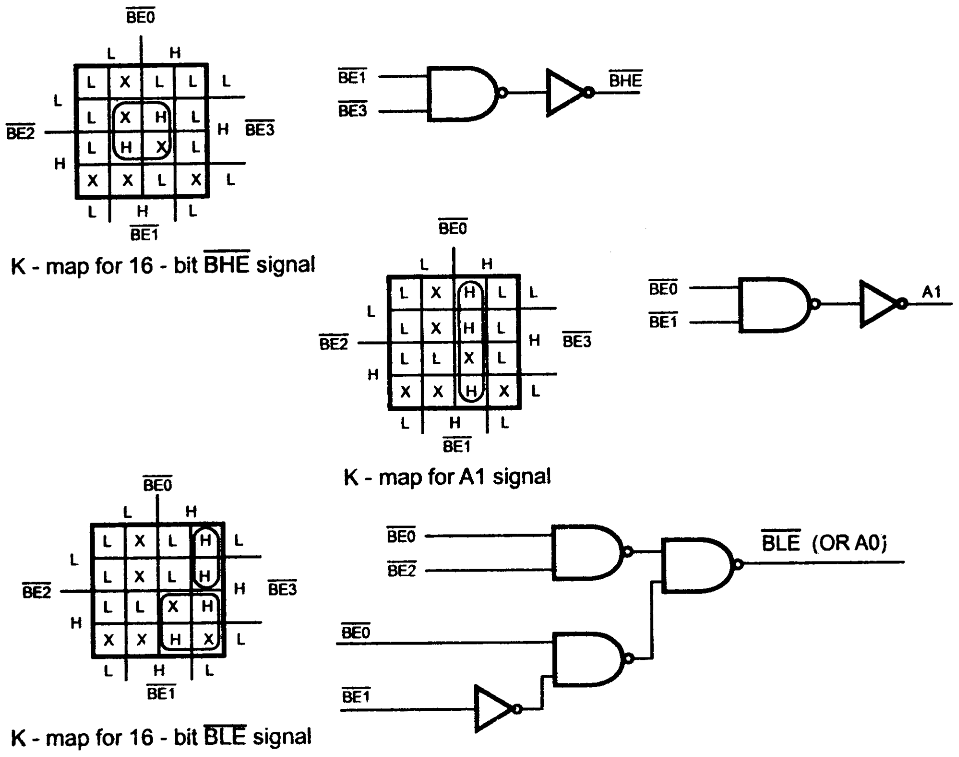


Fig. 13.34 Logic to generate A1, $\overline{BHE}$, and $\overline{BLE}$

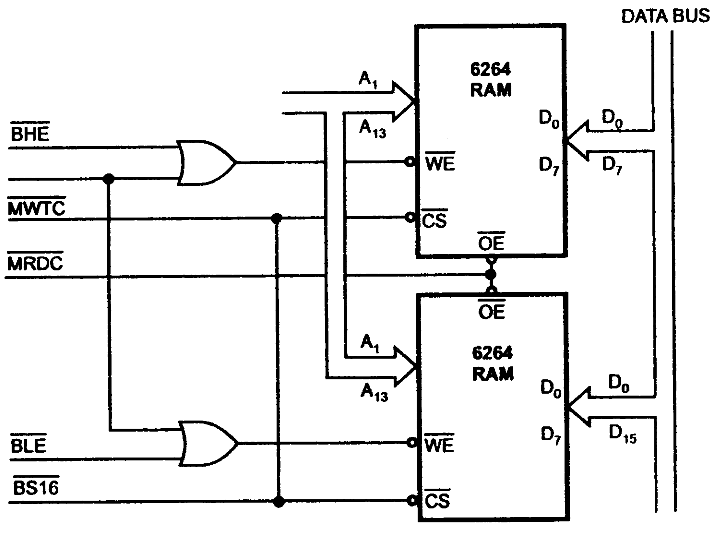


Fig. 13.35 Static RAM interface

In the SRAM interface, the $\overline{OE}$ input of each SRAM IC is connected to the $\overline{MRDC}$ signal from the controller. The $\overline{WE}$ input of both SRAM ICs is driven by the $\overline{MWTC}$ signal. Because it is possible to write only single byte, the $\overline{MWTC}$ signal can't be connected directly to the $\overline{WE}$ inputs. Each $\overline{WE}$ input is activated only when $\overline{MWTC}$ and corresponding byte enable signal are low. Fig. 13.35 shows the static RAM interface. Here, two 8 K SRAMs are connected to the system with 16-bit bus, to provide $8\,K \times 16$-bit of read/write memory. To access 8 kbyte memory (2^{13} = 8196) 13 address lines are required. A1 to A13 lines are connected in parallel to both RAMs. A common chip select signal from the decoder is used to select RAMs. The same signal is used to activate $\overline{BS16}$ input of 80386DX to indicate bus size is 16-bit.

32-bit Bus : Fig. 13.36 shows the SRAM interface with the system. In 32-bit RAM interface, the $\overline{OE}$ inputs of all SRAMS are connected to the $\overline{MRDC}$ signal from the bus controller and $\overline{BE3}$-$\overline{BE0}$ signals are used along with $\overline{MWTC}$ signal to generate $\overline{WE}$ signals for different SRAMs.

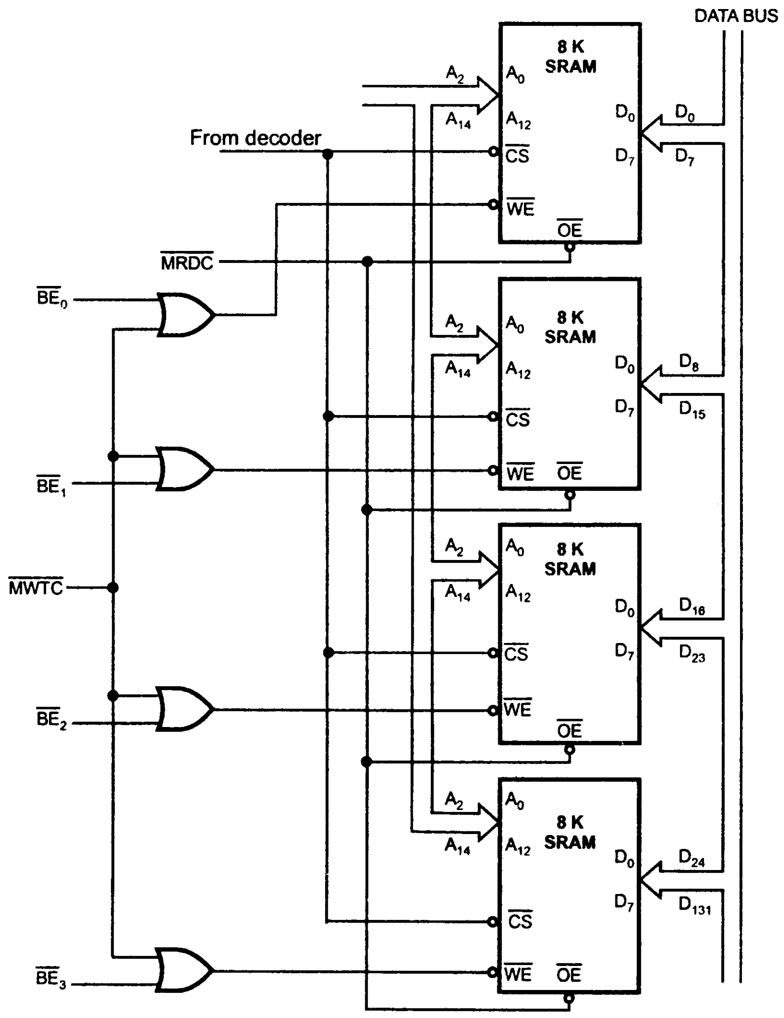


Fig. 13.36 SRAM interface

To access 8 K locations (2^{13}) 13 address lines are required so A2-A14 lines are connected as address lines for each SRAM. It is important to note that in 16-bit or 32-bit read operation byte enable signals are not used, since when 80386 is reading 8-bits (in case of 16-bit) or 16-bit (in case of 32-bit bus), the application of the remaining bits to the data bus has no effect on the operation of the 808386. But in case of write operation, respective bytes must be enabled to update appropriate data.

EPROM interface

16-bit bus : In EPROM interface, the $\overline{OE}$ input of each EPROM device is connected directly to the $\overline{MRDC}$ signal. EPROM is a read only device, so $\overline{WE}$ signal is not present in EPROM. A common chip select signal is used to select EPROMs. The same signal is used to activate $\overline{BS16}$ input of 80386DX to indicate bus size is 16-bit. As it is 16-bit interface, A1 signal is generated using same logic explained earlier.

Fig. 13.37 shows the EPROM interface. Two 27128 EPROMs are used to provide 16 K× 16-bit of program memory. To access 16 kbyte locations (2^{14}) 14 address lines are required.

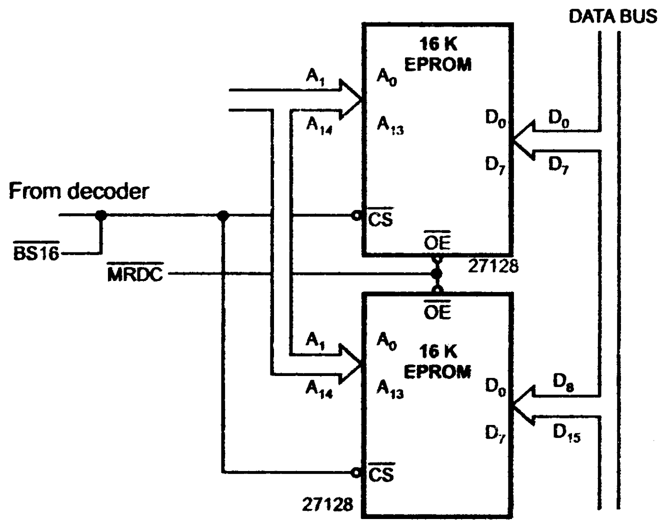


Fig. 13.37 EPROM interface

32-bit Bus : Fig. 13.38 (see Fig. 13.38 on next page) shows the 32-bit EPROM interface with the system. In 32-bit EPROM interface, the $\overline{OE}$ inputs of all EPROMs are connected to the $\overline{MRDC}$ signal from the bus controller. Four 27128 EPROMs are used to provide 16 K× 32 bit program memory. To access 16 kbyte locations (2^{14}) 14 address lines are required. Address lines A2-A15 are connected in parallel to the EPROMs.

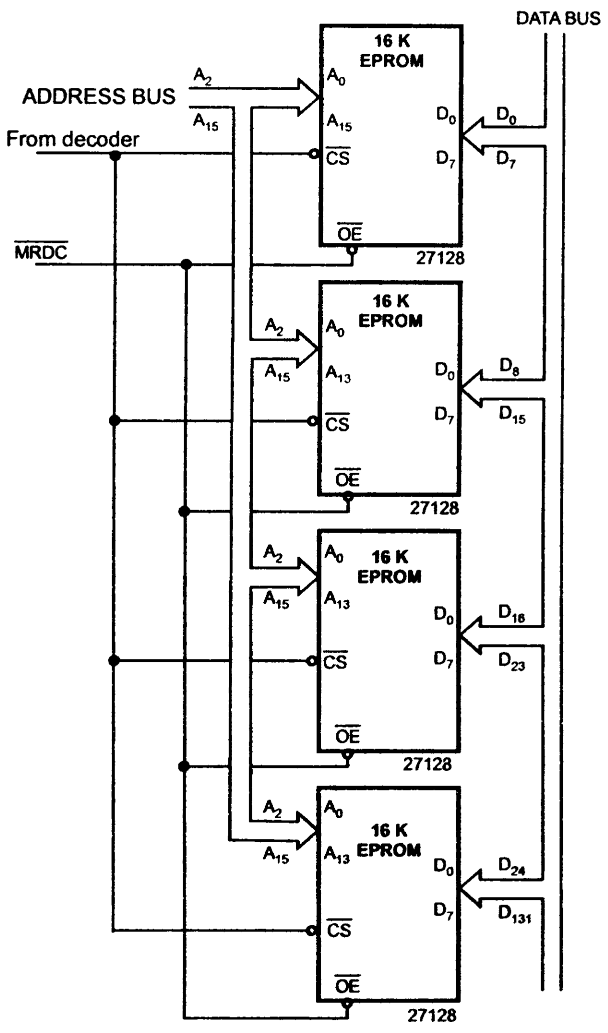
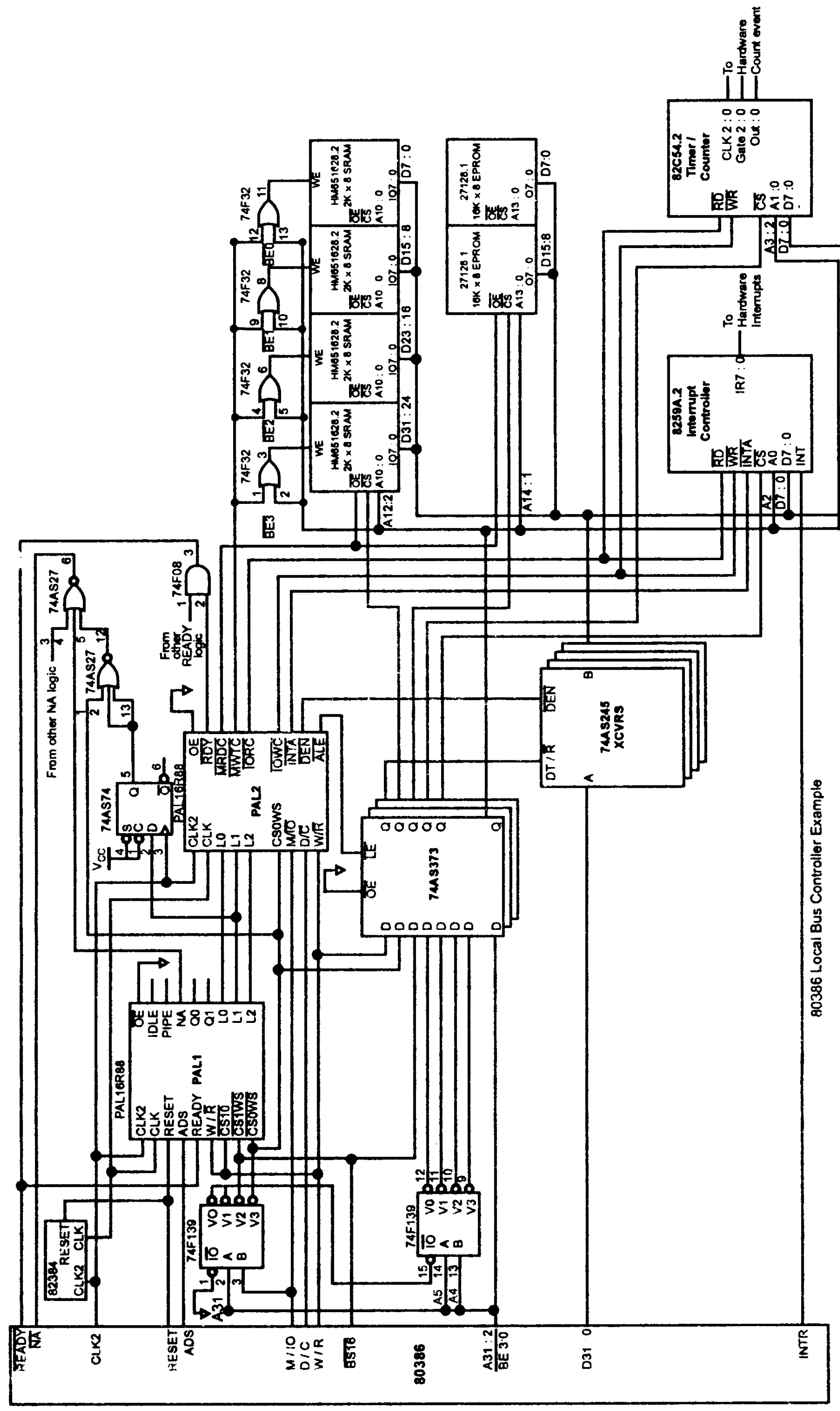


Fig. 13.38 32-bit EPROM interface

Typical memory circuit for 80386DX :

Fig. 13.39 shows (See Fig. 13.39 on next page)the typical memory circuit for 80386DX. It consists of 16 K× 16-bit EPROM, 2 K× 32-bit SRAM. The bus controller logic is implemented using two PALs. PAL1 is used to generate overall bus cycle timing whereas PAL2 is used to generate most of the bus control signals. The timings for these control signals are shown in Fig. 13.40.(See Fig. 13.40 on page 13-59.)

The upper half of 74F139 (2:4) decoder is used to decode address line A31 and control signal M/$\overline{\text{IO}}$. The combination of these two inputs of the decoder activate appropriate decoder output signal. For example, when M/$\overline{\text{IO}}$ is logic 1 and A31 is logic 0, then $\overline{\text{Y2}}$ output of the decoder is activated. This output of decoder is one of the input for PAL1. Two inputs to PAL1 ($\overline{\text{CS0WS}}$ and $\overline{\text{CS1WS}}$) from the address decoder determine the bus control signal timings.



80386 Local Bus Controller Example

Fig. 11.39

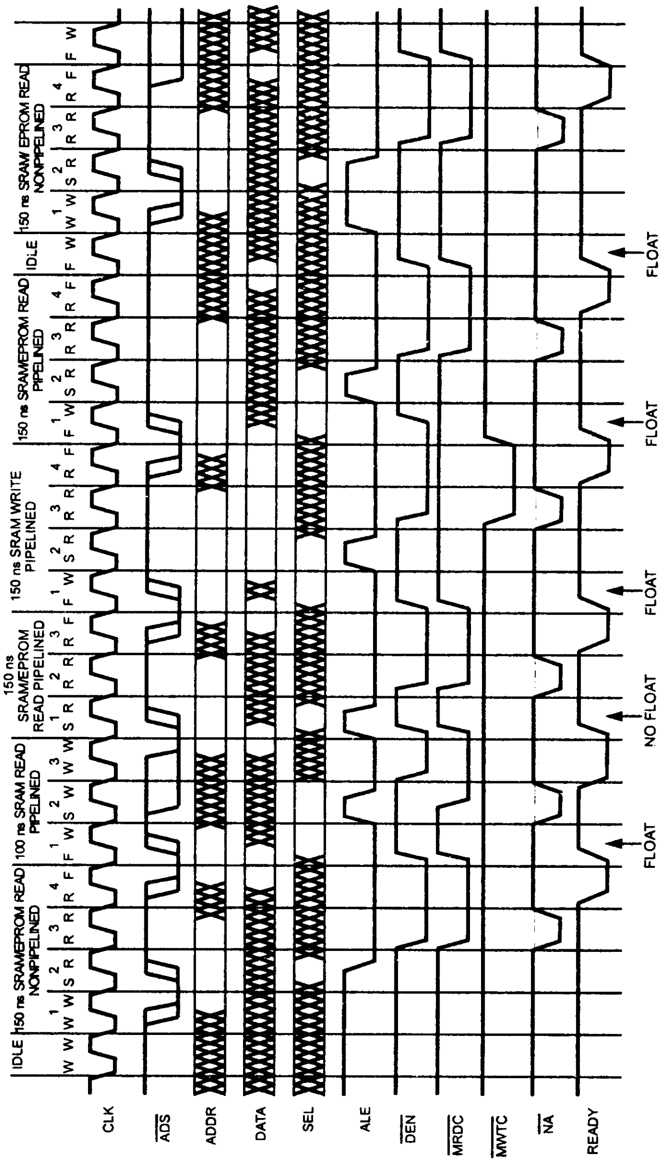

CLK
ADS
ADDR
DATA
SEL
ALE
DEN
MRDC
MWTC
NA
READY
IDLE
150 ns SRAM/EPROM READ NONPIPELINED
100 ns SRAM READ PIPELINED
150 ns SRAM/EPROM READ PIPELINED
150 ns SRAM WRITE PIPELINED
150 ns SRAM/EPROM READ PIPELINED
IDLE
150 ns SRAM/EPROM READ NONPIPELINED
FLOAT
NO FLOAT
FLOAT
FLOAT
FLOAT

Fig. 13.40 Bus control signal timings

Table 13.9 shows the bus cycles generated by Bus Controller. The $\overline{\text{CS0WS}}$ and $\overline{\text{CS1WS}}$ are latched in the 74AS373 to provide $\overline{\text{CS}}$ signals for the 27128 SRAMS and $\overline{\text{CS}}$ signals for EPROMs, respectively.

Chip-select	Cycle Type	WAIT-STATES		Command Delay	Data-float
		Pipelined	Unpipelined		
CS0WS	read	0	1	1 CLK2	2 CLK2
	write	1	2	2 CLK2	—
CS1WS	any	1	2	2 CLK2	4 CLK2

Table 13.9 Bus cycles generated by bus controller

Three octal latches (74AS373) are used to latch address lines (A2-A14), chip select signals, byte enable signals and $W/\overline{R}$ signal. These signals are latched with the help of ALE output of the PAL2 of the bus control logic. Output of latch is permanently enabled by connecting its $\overline{OE}$ signal to the system ground.

The bus cycle definition signals ($M/\overline{IO}$, $W/\overline{R}$, $D/\overline{C}$), $\overline{ADS}$ and chip selects $\overline{CSI0}$, $\overline{CS1WS}$ and $\overline{CS0WS}$ are the inputs for PAL2. The output of PAL2 gives the control signals $\overline{MRDC}$, $\overline{MWTC}$, $\overline{DEN}$, and ALE for the memory interface, along with the $\overline{NA}$ to activate pipelining and $\overline{RDY}$ to insert wait states. The 32-bit data bus is buffered using 74AS245 transceiver. The $\overline{DEN}$ signal from PAL2 is used to enable transceiver and the latched $W/\overline{R}$ signal from 80386DX is used to control the direction of data flow.

The SRAMs and EPROMs are interfaced as explained in previous sections. For SRAM interface 32-bit bus is used whereas for EPROM 16-bit bus is used. Due to 16-bit bus for the EPROM interface chip count for program memory is reduced to two but overall performance is reduced because 32-bit accesses require two bus cycles instead of one in 16-bit bus. However, the system uses EPROMs only for power on initialization and runs programs entirely from SRAMs or DRAMs. Thus with 16-bit bus for EPROM only performance at power-on initialization is affected, but the programs run at the same speed as 32-bit system.

DRAM interface

Use of DRAM is a cost effective solution. The design can be adapted for a wide variety of speed and system requirement to provide high throughput at minimum cost.

DRAMs provide relatively fast access times at a low cost per bit; therefore, large memory systems can be created at low cost. However, DRAMs require a brief idle time between accesses to precharge; if this idle time is not provided, the data in the DRAM can be lost. For two consecutive accesses from the same bank of DRAM, it is necessary to introduce delay equal to precharge time between two accesses. This precharge delay reduces the system performance. To avoid this delay, memory should be arranged such that each subsequent memory access is most likely to be directed to a different bank. Due to such arrangement, most of the times wait time between accesses is not required, because while one bank of DRAMs performs the current access, another bank precharges and will be ready to perform the next access immediately.

13.4.4 Interleaved Memory

In the previous section, we have seen that memory arrangement design is important factor in DRAM interface. One way to design memory interface such that adjacent memory locations should lie in different banks since most programs tend to make subsequent accesses to adjacent memory locations during code fetches, stack operation, and array accesses.

With two banks of DRAMs, one for even 16-bit word addresses and one for odd 16-bit word addresses; all sequential 16-bit accesses can be completed without waiting for the DRAMs to precharge. This memory arrangement is referred to as interleaved memory.

The Fig. 13.41 shows an interleaved memory system. The ALE_0 and ALE_1 signals are used to capture the address for either bank of memory. The memory in each bank is 16-bits wide. As a program executes, the 80386 fetches instruction 16-bits at a time from normally sequential memory locations. Program execution uses interleaving in most cases. However, if a system is going to access mostly 8-bit data, it is doubtful that memory interleaving will reduce the number of wait states.

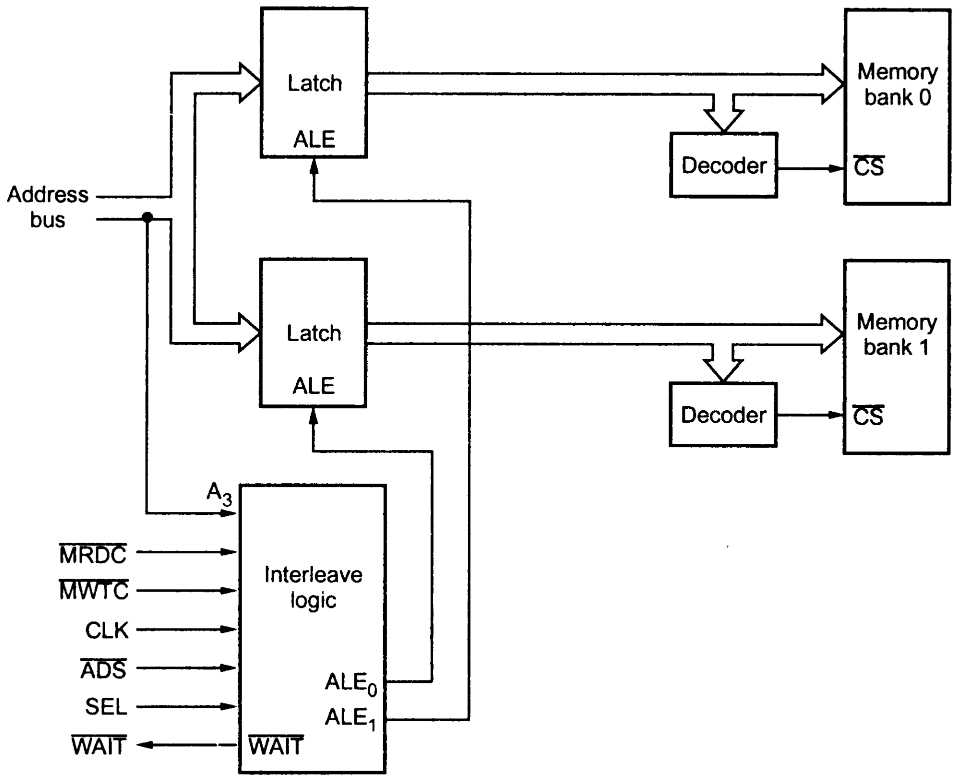


Fig. 13.41 Interleaved memory system

13.5 I/O Interface

Every microprocessor based system requires Input/Output interface to communicate with the outside world. The 80386 supports 8-bit, 16-bit, and 32-bit I/O devices that can be mapped into either the 64 kilobyte I/O address space or the 4 gigabyte physical memory address space.

Fig. 13.42 shows the basic I/O interface. It consists of bus control logic, address decoder, address latch, data transceiver, and wait state generator along with 80386 processor.

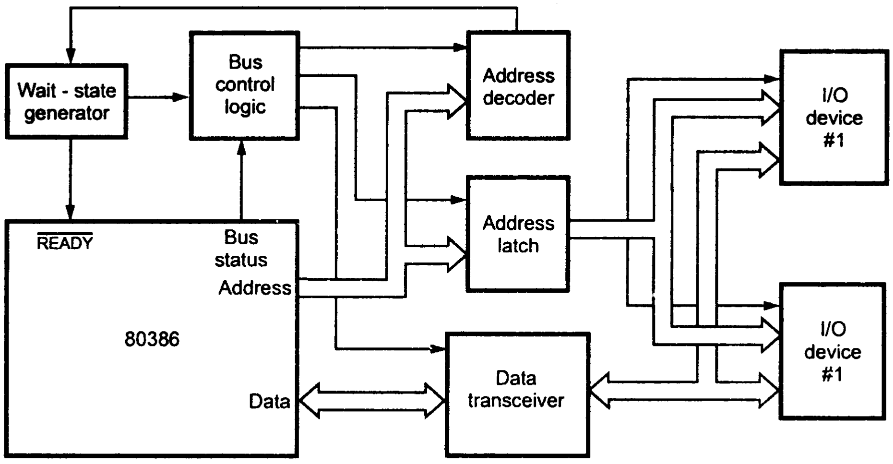


Fig. 13.42 Basic I/O interface block diagram

This chapter describes circuits used in I/O interface and different methods of I/O communication.

Input/Output devices can be interfaced with 80386 systems in two ways.

1. I/O mapped I/O
2. Memory mapped I/O

13.5.1 I/O Mapped I/O

In I/O mapped I/O, the I/O devices are treated separate from memory. The 80386 supports software and hardware features for separate memory and I/O address spaces. Fig. 13.43 shows the memory and I/O spaces in real mode.

The 80386 has four special instructions IN, INS, OUT, and OUTS to transfer data through input/output ports in I/O mapped I/O system. $M/\overline{IO}$ signal is always low when 80386 is executing these instructions. So $M/\overline{IO}$ signal is used to generate separate addresses for Input/Output. Only 256 (2^8) I/O addresses can be generated when direct addressing method is used. By using indirect addressing method this range can be extended upto 65536 (2^{16}) addresses.

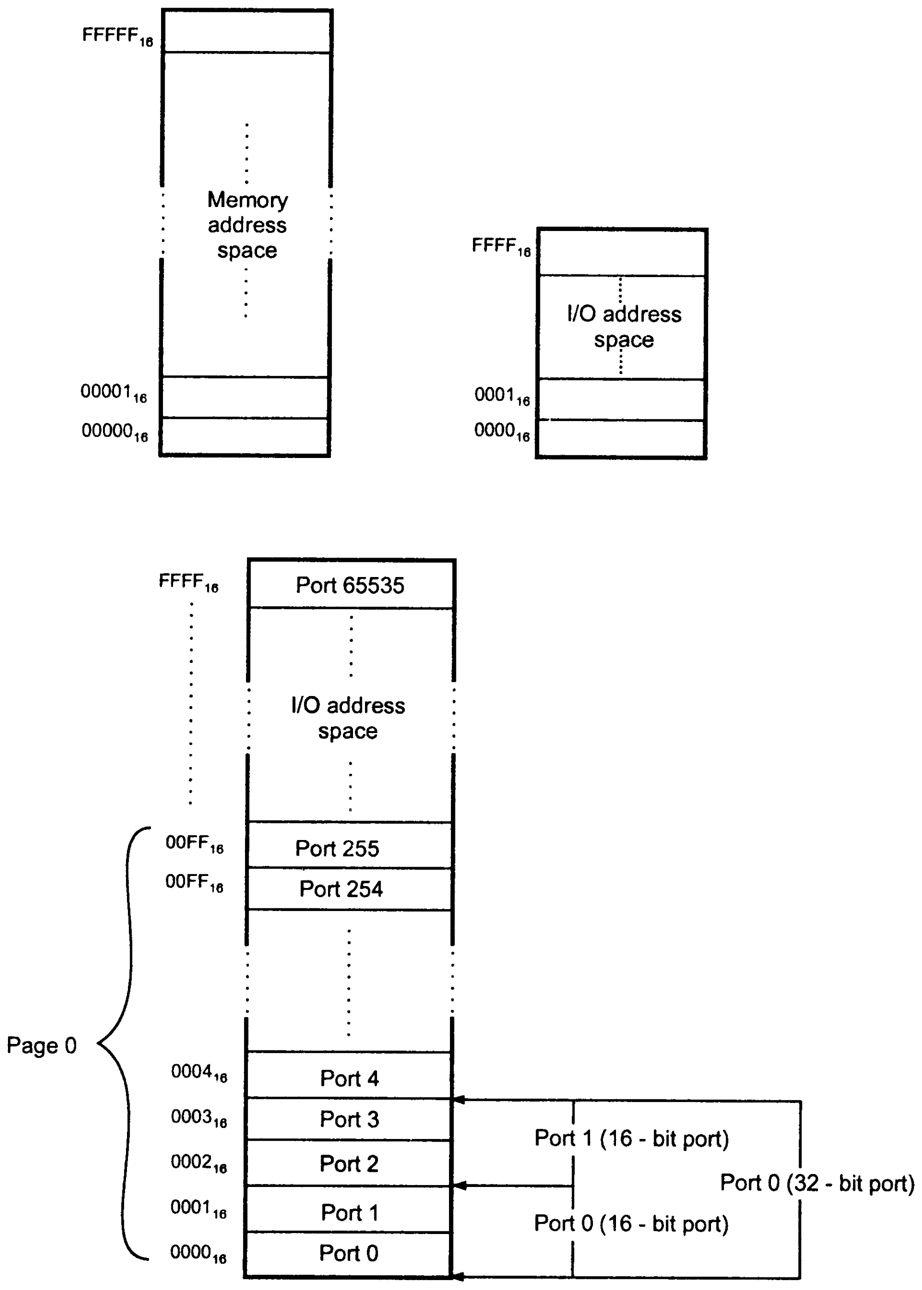


Fig. 13.43 Memory and I/O spaces in real mode

13.5.2 Memory Mapped I/O

In memory mapped I/O, I/O device is placed in the memory address space of the microcomputer system. I/O device is connected as if it is a memory location. For this reason, the method is known as memory mapped I/O.

In a microcomputer system with memory-mapped I/O, some of the memory address spaces are dedicated to the I/O system. Fig. 13.44 shows memory mapped I/O devices in the 80386DX memory address space. Here, 4096 memory addresses from D0000H through D0FFFH are assigned to I/O devices. The contents of D0000H represents byte wide I/O port 0; contents of D0000H and D0001H represents word-wide port 0; and contents of D0000H through D0003 H represents double world wide port 0.

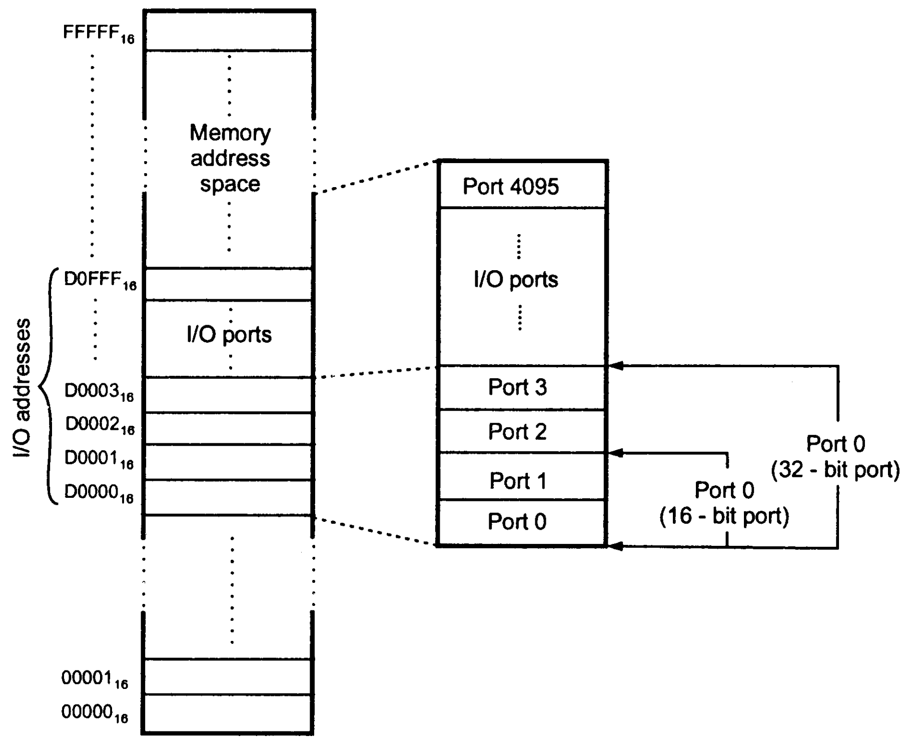


Fig. 13.44 Memory mapped I/O devices

13.5.3 I/O Interface

The 80386 can operate with 8-bit, 16-bit and 32-bit peripherals. The interfacing of I/O devices is not only depend on the signal requirements of the device and its location within the memory space or I/O space, but also depend on the data width. Fig. 13.45 shows a typical I/O mapped I/O interface. Using this I/O circuit it is possible to interface eight I/O devices such as keyboard, printer, mouse display. It consists of address decoder, address latch, data transceiver, bus control logic and write control logic.

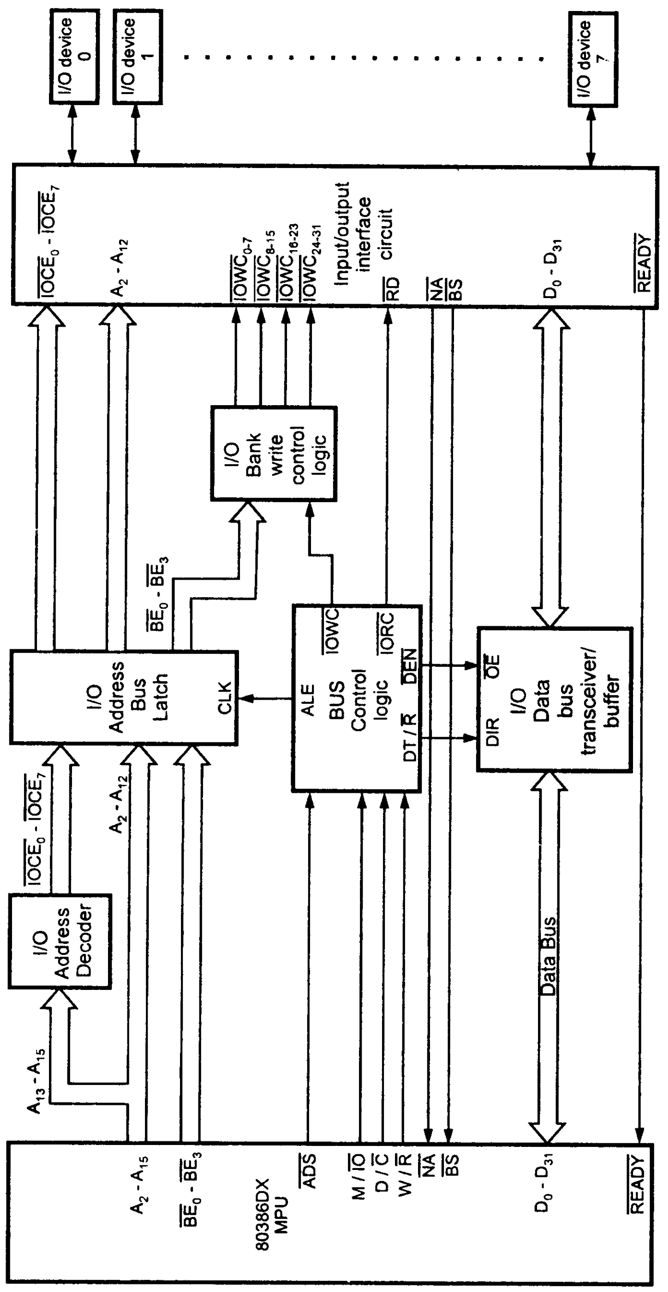


Fig.13.45 Typical I/O mapped I/O interface

Address decoder

Address decoder converts the 80386 address into chip-select signals. It consists of one or more decoders depending on the mapping complexity. I/O address that is an output on address lines A2 through A15 of the 80386DX is decoded by the I/O decoder to generate chip selects signals Fig. 13.45 shows that with three address bits A13 through A15, eight chip select signals are generated. (IOCEO-IOCE7)

Address latch

It is necessary to maintain address, byte enable signals and I/O chip-select signals to be active throughout the bus cycle. These I/O chip enable signals are latched along with the address and byte enable signals in the I/O address latches. This is achieved with a pulse at the ALE output of the bus control logic.

Bus control logic

The bus control logic for the basic I/O interface is the same as the logic for the memory interface described in Fig. 13.45. The bus controller decodes the 80386 status outputs (W/$\overline{R}$, M/$\overline{IO}$, and D/$\overline{C}$) and activates a command signal for the type of bus cycle requested, as explained in section 13.3. The signals generated by Bus controller for memory device and I/O device are as follows :

$\overline{MRDC}$ (Memory Read Command)

This signals is a combination of memory data read and memory code read. This signal is used to instruct memory to put the contents of addressed memory location on the data bus.

$\overline{MWTC}$ (Memory Write Command)

This signal is used to instruct memory to accept the data from the data bus and load the data into the addressed memory location.

$\overline{IORC}$ (I/O Read Command)

This signal is used to instruct an I/O device to put the data contained in the addressed port on the data bus.

$\overline{IOWC}$ (I/O Write Command)

This signal is used to instruct an I/O device to accept the data from the data bus and load the data the into the addressed port.

$\overline{INTA}$ (Interrupt Acknowledge)

In the interrupt acknowledge bus cycle this signal is activated two times. First $\overline{INTA}$ is used as an acknowledgment whereas second $\overline{INTA}$ is used to instruct interrupt controller to put the interrupt type on the data bus.

Write control logic

$\overline{IORC}$ signal is directly applied to the I/O devices, but $\overline{IOWC}$ signal is gated with byte enable signals (BE3 - BE0) to generate separate write enable signals for each byte of the data bus. ($IOWC_{0-7}$, $IOWC_{8-15}$, $IOWC_{16-23}$, and $IOWC_{24-31}$). These signals are required to support writing of 8-bit, 16-bit or 32 bit data through the interface.

Data transceiver

Data transceivers are used to buffer the data bus. The output signals DT/$\overline{R}$ and $\overline{DEN}$ from bus control logic control the direction and output enable signals of the transceivers, respectively.

13.6 Introduction to 80486 Processor

The 32-bit 80486 is the next evolutionary step up from the 80386. It is a highly integrated device containing about 1.2 million transistors. The basic processor unit used in the 80486 is the same as that used in 80386. In this section, we are going to discuss additions and enhancements in 80486 over 80386.

13.6.1 Features

1. It is a highly integrated device containing about 1.2 million transistors.
2. The 80486 operates on 25 MHz, 33 MHz, 50 MHz, 66 MHz or 100 MHz.
3. It has built-in math coprocessor.
4. 80486 is a 32-bit architecture with on-chip memory management and cache memory units.
5. On-chip cache memory allows frequently used data and code to be stored on-chip, thereby reducing accesses to the external bus.
6. MMU consists of segmentation unit and paging unit. Segmentation allows management of the logical address space by providing easy data and code relocatibility and efficient sharing of global resources. Paging allows operating system designers to make physical memory appear to be anywhere in the 4 gigabytes address space. In protected virtual mode it can manage upto 64 Terabytes of virtual memory.
7. The MMU provides four levels of protection for isolating and protecting applications and the operating system from each other.
8. The 80486 has three modes of operation : Read mode, Protected mode and Virtual 8086 mode.
9. It is available in two versions : 80486 DX and 80486SX. The only difference between these two versions is that the 80486SX does not contain the numeric coprocessor.
10. Most of the 80486 instructions require only one clock instead of two clocks required by the 80386.
11. It supports five-stage instruction pipeline scheme that allows it to execute instructions much faster than 80386.
12. It executes conditional JUMP instructions more efficiently. When the 80486 decodes a conditional jump instruction, it automatically prefetches one or more instructions from the jump destination address just in case the jump is taken. Therefore, if the branch is taken, the 80486 does not have to wait through a bus cycle for the first instruction at the branch address.
13. It has built-in parity check/generator unit to implement parity detection and generation for memory reads and writes.
14. It supports burst mode memory reads and writes to implement fast cache fills.
15. It executes a few new instructions that control the internal cache memory and allow addition (XADD) and comparison (CMPXCHG) with an exchange and a byte

swap (BSWAP) operation. Other than these few additional instructions, the 80486 is 100 percent compatible with the 80386 and 80387.

16. It supports built-in-self-test. It tests microprocessor, coprocessor, and cache at reset time. If the 80486 passes the test, EAX contains a zero.
17. It has additional test registers (TR3 - TR5) to test the cache memory.

13.6.2 Architecture of 80486

As mentioned earlier, the internal architecture of the 80486 is an improvement over that of the 80386. For instance, we said that a floating point coprocessor and cache memory are now on-chip. These are not the only changes that have been made to improve the performance of the 80486. Let us see the functional units of 80486 and understand how they have changed from that of the 80386.

The Fig. 13.46 shows the block diagram of the internal architecture of the 80486. (See Fig. 13.46 on next page.) Looking at Fig. 13.46 we find the execution unit, segmentation unit, paging unit, bus interface unit, prefetch unit and decode unit are similar to the 80386 architecture. However, the functions of many of these elements have been enhanced for the 80486. For example, the coding of instructions in the control ROM had been changed to permit instructions to be performed in less clock cycles. Some other changes are that the code queue in the prefetch unit has been doubled in size to 32 bytes. This allows more instructions to be held on chip ready for decode and execution. Also an improved algorithm is now used by the translation lookaside buffer in the paging unit. Finally, the bus interface unit has been modified to give the 80486 architecture a much faster and more versatile processor bus. It is supported with additional parity checker/generator. Parity checker/generator generates parity during each write cycle. Parity is generated as an even parity and parity bit is provided for each byte of memory. On each read cycle, 80486 checks parity and generates a parity check error, if it occurs.

Let us now see the new functional units of the 80486. Traditionally, the floating point operations have been performed by an externally attached floating point coprocessor (Maths coprocessor). The 80486 has a built-in math coprocessor. This coprocessor is essentially the same as the 80387 processor used with 80386, but being integrated on chip allows it to execute math instructions about three times as fast as a 80386 and 80387 combination.

The cache unit of 80486 consists of an 8 Kbyte code and data cache. This additional high speed cache memory provides a way of improving overall system performance while permitting the use of low-cost, slow-speed memory devices in the main memory system. During system operation, the cache memory contains recently used instructions, data, or both. The objective is that the MPU accesses code and data in the cache most of the time, instead of from the main memory. Since less time is required to access the information from the cache memory, the result is a higher level of system performance.

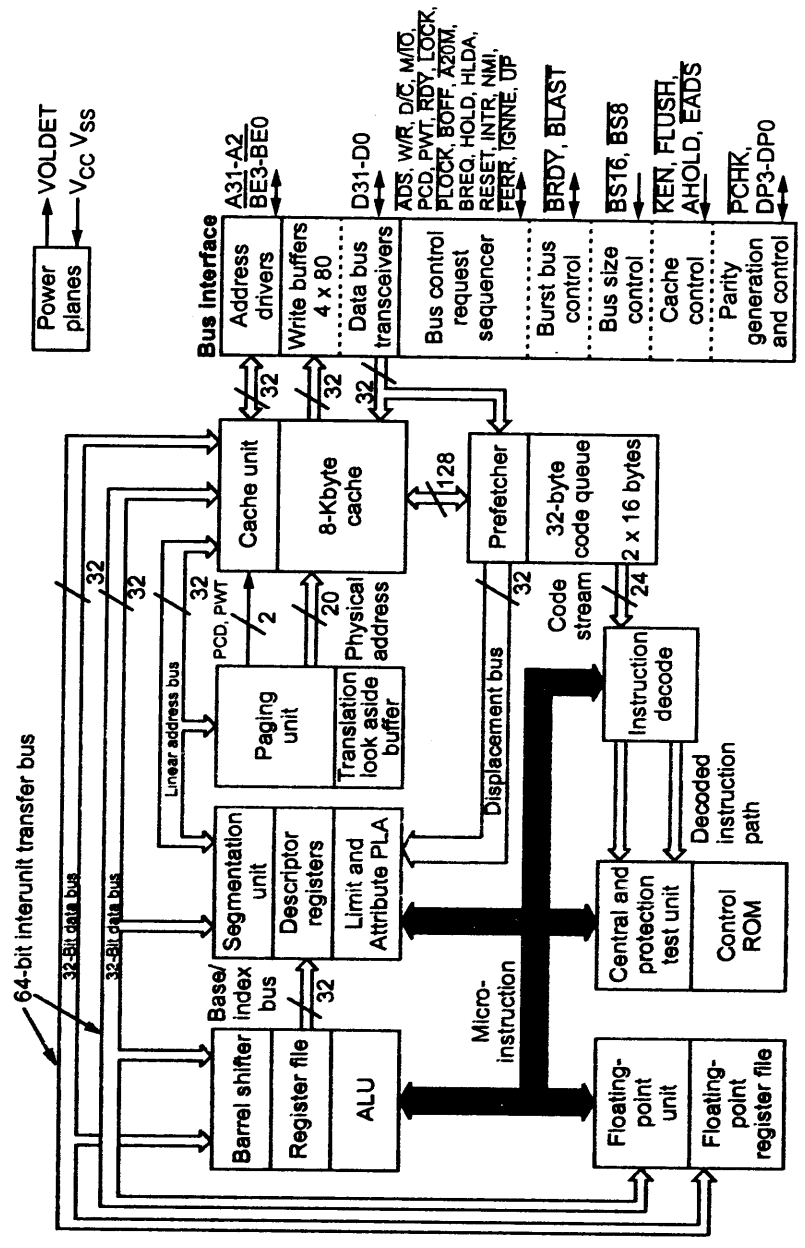

Power planes
VOLDET
V_{CC} V_{SS}
Bus interface
Address drivers
Write buffers 4 x 80
Data bus transceivers
Bus control request sequencer
Burst bus control
Bus size control
Cache control
Parity generation and control
A31-A2, $\overline{BE3}$-$\overline{BE0}$
D31-D0
$\overline{ADS}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, PCD, PWT, $\overline{RDY}$, $\overline{LOCK}$, $\overline{PLOCK}$, $\overline{BOFF}$, $\overline{A20M}$, BREQ, HOLD, HLDA, RESET, INTR, NMI, $\overline{FERR}$, $\overline{IGNNE}$, UP
$\overline{BRDY}$, $\overline{BLAST}$
$\overline{BS16}$, $\overline{BS8}$
$\overline{KEN}$, $\overline{FLUSH}$, AHOLD, $\overline{EADS}$
$\overline{PCHK}$, DP3-DP0
Cache unit
8-Kbyte cache
128
Prefetcher
32-byte code queue
2 x 16 bytes
32
24
Code stream
Instruction decode
Decoded instruction path
Central and protection test unit
Control ROM
Paging unit
Translation look aside buffer
PCD, PWT
2
20
Physical address
Linear address bus
Displacement bus
Segmentation unit
Descriptor registers
Limit and Attribute PLA
Base/ index bus
Micro- instruction
Barrel shifter
Register file
ALU
Floating-point unit
Floating-point register file
64-bit interunit transfer bus
32-Bit data bus
32-Bit data bus

Fig. 13.46 Block diagram of 80486

13.6.3 Programmable Model

The programmable model of 80486 is same as 80386. It contains eight general purpose 32-bit registers : EAX, EBX, ECX, EDX, EBP, EDI, ESI, and ESP. These registers may be used as 8, 16, or 32-bit data registers or to address a location in the memory system. The 80486 also contains same segment registers as the 80386, which are CS, DS, ES, SS, FS, and GS. In addition to this 80486 also contains the global, local, and interrupt descriptor table register and memory management unit, as in the 80386. The only difference in the programmable model is in the extended flag register (EFLAGs) and additional test register. As shown in the Fig. 13.47, EFLAGs contains additional flag bit AC (Alignment Check). It is used to indicate that the microprocessor has accessed a word at an odd address or a doubleword stored at a non-doubleword boundary.

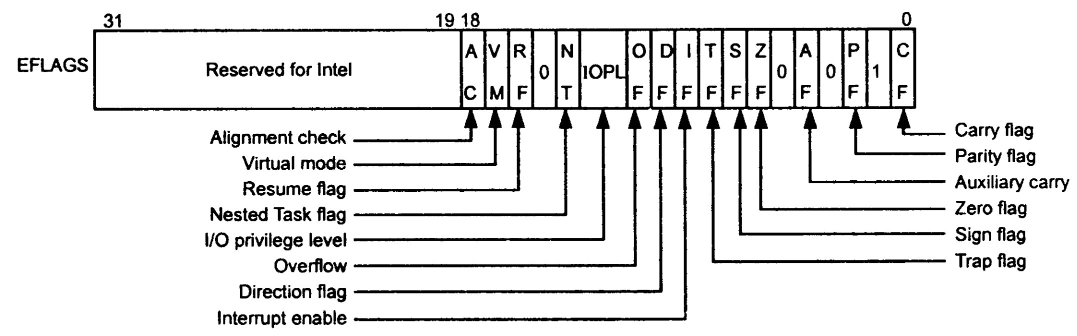


Fig. 13.47 Bit pattern for EFLAGs registers

13.6.4 Pin Description of 80486

Fig. 13.48 shows function pin diagram for 80486. The 80486 data bus, address bus, byte enable, $\overline{ADS}$, $\overline{RDY}$, INTR, RESET, NMI, M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$, $\overline{LOCK}$, HOLD, HLDA, and $\overline{BS16}$ signal function as we described for 80386. New signals are described as follows : (See Fig.13.48 on next page).

Parity signals

This new signal group includes DP_0-DP_3 and $\overline{PCHK}$ signals. These signals allow the 80486 to implement parity detection/generation for memory reads and writes.

DP_0-DP_3 : Data Parity : During each memory write operation, the 80486 generates an even parity bit for each byte and outputs these bits on the DP_0-DP_3 pins. The 80486 checks the parities of the data bytes read and compares them with the DP_0-DP_3 signals. DP_0-DP_3 signals are active high signals and should be connected to V_{CC} through a pull-up resistor in systems not using parity.

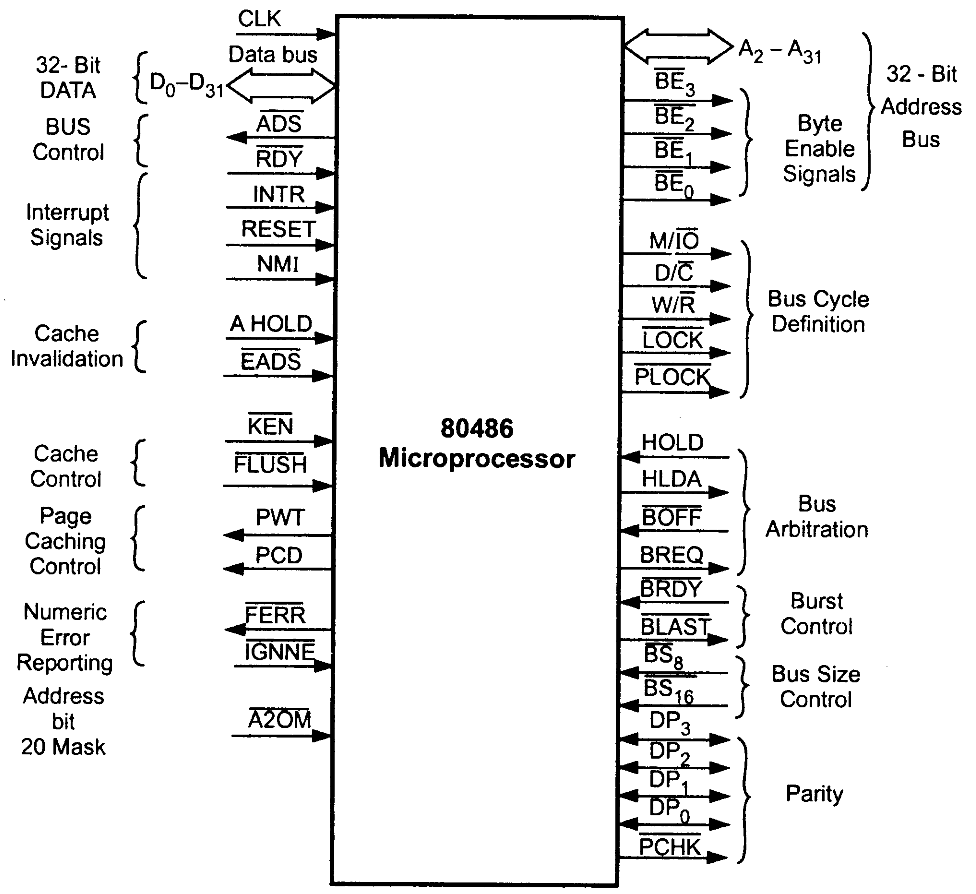


Fig. 13.48 80486 function pin diagram

$\overline{\text{PCHK}}$: Parity check : The 80486 checks the parities for enabled bytes as indicated by the byte enable and bus size signals. If a parity error is found, the 80486 asserts the $\overline{\text{PCHK}}$ signal.

Burst Control ($\overline{\text{BRDY}}$ and $\overline{\text{BLAST}}$ signals)

Another new signal group consists of the brust ready, $\overline{\text{BRDY}}$ signal, and the brust last, $\overline{\text{BLAST}}$ signal. These signals are used to control brust-mode memory read and writes. A normal read a line into cache operation requires 2 clock cycles. However, if a series of reads is being done from successive memory locations, the reads can be done in the brust mode with only 1 clock cycle per read. To activate brust-mode, 80486 sends the starting address and disables $\overline{\text{BLAST}}$ signal. When the external DRAM controller is ready with the data word on the data bus, it asserts the $\overline{\text{BRDY}}$ signal. The 80486 reads the data word and outputs the next address. Since the data words are at successive addresses, only the lower address bits to be changed. After reading the required number of data words 80486 asserts the $\overline{\text{BLAST}}$ signal low to terminate the brust mode.

Bus arbitration

Bus arbitration logic consists of BREQ (Bus Request), $\overline{\text{BOFF}}$ (Backoff), HOLD and HLDA (Hold acknowledge) signals. These signals are used to control sharing the local 80486 bus by multiple processors (bus masters). We know that, the HOLD and HLDA signals are used when another master needs the control of local bus. When a 80486 (master) on the bus needs to use the bus, it internally generates the bus request BREQ signal. An external priority circuit evaluates requests from bus masters and grants bus access to the highest priority master. The bus controller asserts $\overline{\text{BOFF}}$ signal to floats its bus in the next clock. The $\overline{\text{BOFF}}$ has higher priority than $\overline{\text{BRDY}}$; if both are activated in the same clock, $\overline{\text{BOFF}}$ takes effect. The 80486 remains in bus hold until $\overline{\text{BOFF}}$ is active. If a bus cycle is in progress when $\overline{\text{BOFF}}$ is active, the cycle is restarted.

Bus cycle definition

The signals from this group such as M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, W/$\overline{\text{R}}$ and $\overline{\text{LOCK}}$ have same meaning as in 80386. This group has one more signal $\overline{\text{PLOCK}}$ (Pseudo-Lock) which is not supported by 80386.

$\overline{\text{PLOCK}}$ indicates that the current bus transaction requires more than one bus cycle to complete. Examples of such operations are floating-point long reads and writes (64-bits), segment table descriptor reads (64-bits), and cache line fills (128-bits). The 80486 microprocessor drives $\overline{\text{PLOCK}}$ active until the addresses for the last bus cycle of the transaction have been driven, regardless of whether RDY or BRDY has been returned. Normally, $\overline{\text{PLOCK}}$ and $\overline{\text{BLAST}}$ are inverse of each other. However, during the first bus cycle of a 64-bit floating point write, both $\overline{\text{PLOCK}}$ and $\overline{\text{BLAST}}$ will be asserted.

Cache control

This new group consists of cache flush ($\overline{\text{FLUSH}}$) and cache enable ($\overline{\text{KEN}}$) signals.

$\overline{\text{FLUSH}}$: $\overline{\text{FLUSH}}$ forces the 80486 microprocessor to flush its entire internal cache. $\overline{\text{FLUSH}}$ is active Low and need only be asserted for one clock. $\overline{\text{FLUSH}}$ is asynchronous but setup and hold times must be met for recognition in any specific clock. $\overline{\text{FLUSH}}$ being sampled Low in the clock before the falling edge of RESET causes the 80486 microprocessor to enter the three-state test mode.

$\overline{\text{KEN}}$: is used to determine whether the current cycle is cacheable. When the 80486 microprocessor generates a cacheable cycle and $\overline{\text{KEN}}$ is active, the cycle becomes a cache line fill cycle. Returning $\overline{\text{KEN}}$ active one clock before $\overline{\text{RDY}}$ during the last read in the cache line fill causes the line to be placed in the on-chip cache. $\overline{\text{KEN}}$ is active Low and is provided with a small internal pull-up resistor. $\overline{\text{KEN}}$ must satisfy setup and hold times for proper operation.

Cache invalidation : This new group consists of AHOLD and $\overline{\text{EADS}}$ signals.

AHOLD : Address Hold

This request allows another bus master access to the 80486 microprocessor's address bus for a cache invalidation cycle. The 80486 microprocessor stops driving its address bus in the clock following AHOLD going active. Only the address bus is floated during address hold; the remainder of the bus remains active. AHOLD is active High and is provided with a small internal pull-down resistor. For proper operation, AHOLD must meet setup and hold times.

$\overline{\text{EADS}}$: Valid External Address

This signal indicates a valid external address has been driven onto the 80486 microprocessor address pins. This address is used to perform an internal cache invalidation cycle. $\overline{\text{EADS}}$ is active Low and is provided with an internal pull-up resistor. $\overline{\text{EADS}}$ must satisfy setup and hold times for proper operation.

Page caching control

This new group consists of PWT (Page Write Through) and PCD (Page Cable Disable) signals.

PWT and PCD : The outputs reflect the state of the page attribute bits PWT and PCD in the page table or page directory entry. If paging is disabled or unpaged cycles occur, PWT and PCD reflect the state of the PWT and PCD bits in Control Register 3. PWT and PCD have the same timing as the cycle definition pins (M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, and W/$\overline{\text{R}}$). PWT and PCD are active High and are not driven during bus hold. PCD is masked by the Cache Disable Bit (CD) in control register 0.

Numeric Error Control

This new group consists of $\overline{\text{FERR}}$ (Floating Point Error) and $\overline{\text{IGNNE}}$ (Ignore Numeric Error) signals.

$\overline{\text{FERR}}$: **Flating Point Error :** Driven active when a floating point error occurs. $\overline{\text{FERR}}$ is similar to the $\overline{\text{ERROR}}$ pin on a 387 math coprocessor. $\overline{\text{FERR}}$ is included for compatibility with systems using DOS type floating-point error reporting. $\overline{\text{FERR}}$ is active low, and is not floated during bus hold, except during three-state test mode.

$\overline{\text{IGNNE}}$: **Ignore Numeric Error :** When this signal is asserted, the 80486 microprocessor will ignore a numeric error and continue executing non-control floating point instructions. When $\overline{\text{IGNNE}}$ is deasserted, the 80486 microprocessor will freeze on a non-control floating point instruction if a previous floating point instruction caused an error. $\overline{\text{IGNNE}}$ has no effect when the NE bit in control register 0 is set. $\overline{\text{IGNNE}}$ is active low and is provided with a small internal pull-up resistor. $\overline{\text{IGNNE}}$ is asynchronous but must meet setup and hold times to ensure recognition in any specific clock.

Address Bit 20 Mask ($\overline{A20M}$)

This new input signal, when asserted, forces 80486 to mask physical address bit 20 (A_{20}) before performing a look-up to the internal cache or driving a memory cycle on the bus. $\overline{A20M}$ emulates the address wraparound at 1 Mbyte, which occurs on the 8086. $\overline{A20M}$ is active Low and should be asserted only when the processor is in Real mode. This pin is asynchronous but should meet setup and hold times for recognition in any specific clock. For proper operation, $\overline{A20M}$ should be sampled High at the falling edge of RESET.

13.6.5 New Instructions in 80486

The 80486 has six additional instructions beyond those of 80386. These are listed in Table 13.10.

Instruction	Description
INVD	Invalidates the cache
WBINVD	Write back operation
INVLPG	Invalidates the TLB entry
BSWAP	Swaps the order of bytes in 32-bit register
XADD	In this instruction the source operand is a register and the destination is a register or a memory location. The source and destination operates are added together, with the sum replacing the contents of the destination. The original destination is copied into the source register. This instruction affects all flags.
CMPXCHG	This instruction compares the destination operand with the accumulator (AL, AX, or EAX, depending on the size of destination). The flags are set accordingly. If the accumulator and destination operands are different, the accumulator is replaced by the value in the destination.

Table 13.10

13.6.6 80486 Memory System

The memory system for the 80486 is similar to the 80386 memory system. It consists of 4 Gbytes of memory, beginning at location 0000 0000H and ending at location FFFFFFFFH. The major change to the memory system is internal to the 80486 in the form of an 8 kbyte cache memory. This 8 K cache memory increases speed of execution of instructions and the acquisition of data. Another change is the addition of parity checker/generator built into the 80486 microprocessor.

Parity checker/generator

We know that, the parity bit is used along with the data to check correctness of data when it is accessed. The Intel has incorporated facility by providing internal parity checker/generator in 80486. The 80486 generates parity during each write cycle. Parity is generated as an even parity, and parity bit is provided for each memory byte. The parity bits appear on pins DP_0-DP_3, which are parity input/output pins. These parity bits are typically stored in memory during each write cycle and read from memory during each read cycle.

When data is read along with the parity bits, the microprocessor checks parity and generates a parity check error, if it occurs, on the $\overline{PCHK}$ pin. A parity error causes no change in processing unless the $\overline{PCHK}$ signal is connected to interrupt input. The Fig. 13.49 shows the organisation of 80486 memory system. Here, separate storage is provided to store parity bits.

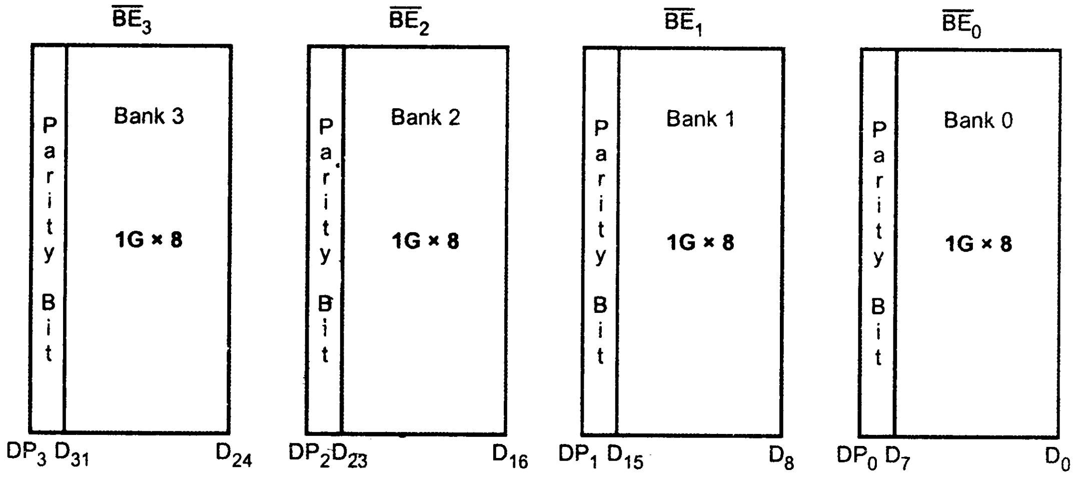


Fig. 13.49 80486 memory system organisation

Cache memory

The cache unit of 80486 consists of 8 kbyte code and data cache. The cache is organised as a 4-way set associative cache, with each location containing 16 byte or four doublewords of data. The cache operates as a write through cache and it changes only if a miss occurs. This means that data written to a memory location not already cached are not written to the cache.

The Control Register 0 (CR_0) is used to control the cache with two new control bits : CD (cache disable) and NW (non-cache write through). When CD bit is logic 1, all cache operations are inhibited. When NW bit is logic 1, the cache write through operations are inhibited. Therefore, for normal operations CD and NW bits are 0. However, they are logic 1 for debugging purposes.

In 80486, the cache is filled with a burst cycle. The 80486 acquires four 32-bit numbers from the memory system to fill a line in the cache. The burst cycle is a special memory cycle where four 32-bit numbers are fetched from the memory system in five clocking periods. If the clock frequency of the 80486 is 33 MHz, it is possible to fill a cache line in 167 ns, which is quite faster than the normal non-brust memory read operations.

Memory read timings

For non-brust memory operation, 80486 requires two clock periods as shown in the Fig. 13.50. Clock period T_1 provides the memory address and control signals and in T_2 actual data transfer takes place between the memory and the microprocessor.

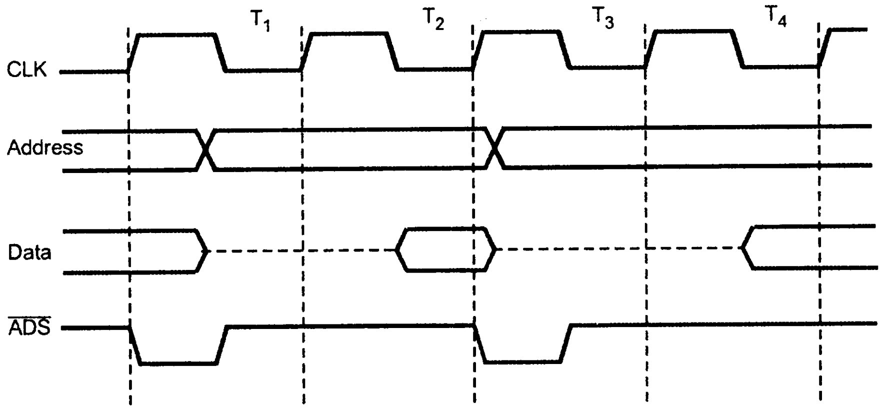


Fig. 13.50 The non-burst read timings for the 80486

The Fig. 13.50 shows the timing diagram for filling a cache line with four 32-bit numbers using a burst cycle. The address A_{31}-A_2 appear during T_1, out of which A_{31}-A_4 remain constant throughout the burst cycle. The address bits A_2 and A_3 change during each T_2 such that the entire address points to the next memory location in the sequence. As shown in the Fig. 13.51, the entire burst cycle is completed in 5 clock periods. Because DRAM memory access is slow (40 ns at best), usually SRAM is used for burst cycle transfer. When SRAM is used for burst cycle transfer, the circuit is called **synchronous burst mode cache** and $\overline{\text{BRDY}}$ pin is used to acknowledge the burst transfer.

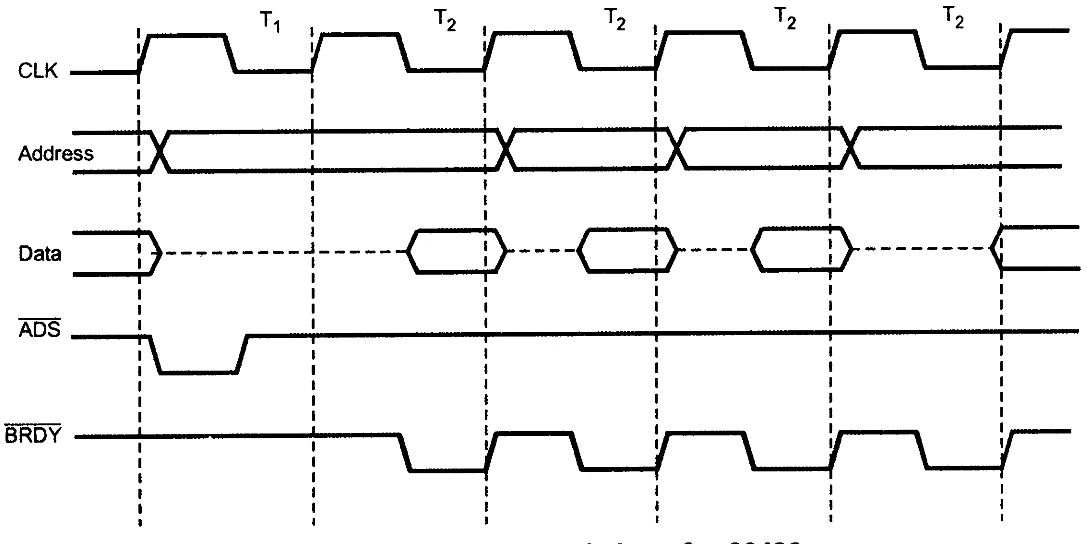


Fig. 13.51 Burst cycle timings for 80486

13.6.7 Memory Management of 80486

The 80486 has similar memory system as for 80386. The 80486 memory system includes a paging unit. It allows any 4 kbyte block of physical memory to be assigned to any 4 kbyte of linear memory. The descriptor types of 80486 are same as that for 80386. The only difference in the 80486 memoiy system to 80386 memory system is in paging mechanism.

The 80486 paging mechanism can be disabled for caching sections of translated memory pages, while the 80386 cannot. The Fig. 13.52 shows the page table directory entry and the page table entry. In these two entries, there are two new control bits : PWT (pag-write through) and PCD (page cache disable). The PWT controls the cache functions for a write operation of the external memory; however it does not control writing to the internal cache. The status of this bit is replicated on the PWT pin of the 80486 microprocessor. This external pin can be used to dictate the write-through policy of the external cache. The PCD bit controls the on-chip cache. If the PCD = 0, the on-chip cache is enabled for the current page of memory. If PCD = 1, the on-chip cache is disabled.

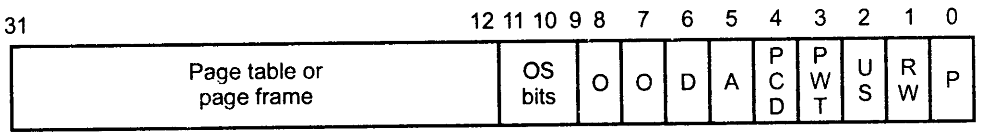


Fig. 13.52 Page directory

13.7 Introduction to Pentium Processor

Pentium processor is one more advance step in Intel's family of processors. It offers compatibility with all previous 80 × 86 machines with many new architectural improvements. In this chapter we are going to discuss the features, architecture, signals and other enhancements of basic pentium processors.

The pentium processor family architecture contains all of the features of the 80486 microprocessor and provides significant additions and enhancements as given below :

- **Wider data bus width :** The Pentium processors have a wider data bus width. The data bus width has been increased from 32-bit to 64-bit to improve the data transfer rate. Burst read and burst write back cycles are supported by the Pentium processors. In addition to 64-bit bus, bus cycle pipelining has been added to allow two bus cycles to be in progress simultaneously.
- **Faster floating point unit :** The floating point unit has been completely redesigned over the 80486 CPU. Faster algorithms provide up to ten times speed-up for common operations including add, multiply, and load.
- **Improved cache structure :** Pentium processors include separate code and data caches integrated on-chip to meet performance goals. Each cache is 8 kbytes in size, with a 32 byte line size and is 2-way set associative. Each cache has a dedicated Translation Lookaside Buffer (TLB) to translate linear addresses to physical addresses. The data cache is configurable to be write back or write

through on a line-by-line basis and follows the MESI protocol. The data cache tags are triple ported to support two data transfers and an inquire cycle in the same clock. The code cache is an inherently write-protected cache. The code cache tags are also triple ported to support snooping and split line accesses. Individual pages can be configured as cacheable or non-cacheable by software or hardware. The caches can be enabled or disabled by software or hardware.

- **Dual integer processor :** Pentium processor has a dual integer processor. It allows execution of two instructions per clock.
- **Branch prediction logic :** The Pentium uses technique called **branch prediction** to check whether a branch will be valid or invalid. To implement branch prediction Pentium processor has two prefetch buffers, one to prefetch code in a linear fashion, and one that prefetches code according to the Branch Target Buffer (BTB). Therefore, the needed code is almost always prefetched before it is required for execution.
- **Data integrity and error detection :** The Pentium processors have added significant data integrity and error detection capability. Data parity checking is still supported on a byte-by-byte basis. Address parity checking, and internal parity checking features have been added along with a new exception, the machine check exception.
- **Functional redundancy checking :** The Pentium processors have implemented functional redundancy checking to provide maximum error detection of the processor and the interface to the processor. When functional redundancy checking is used, a second processor, the "checker" is used to execute in lock step with the "master' processor. The checker samples the master's outputs and compares those values with the values it computes internally, and asserts an error signal if a mismatch occurs.
- **Enhancement virtual 8086 mode :** Enhancements to the virtual 8086 mode have been made to increase performance by reducing the number of times it is necessary to trap to a virtual 8086 monitor.
- **Superscalar processor :** Processors capable of parallel instruction execution of multiple instructions are known as **superscalar processors**. The Pentium is capable, under special circumstances, of executing two integer or two floating point instructions simultaneously and thus it supports superscaler architecture.

The Pentium Pro is a still faster version of the Pentium, and it contains a modified internal architecture that can schedule up to five instructions for execution, and an even faster floating point unit. It also contains a 256 kbyte or 512 kbyte level two cache in addition to the 16 kbyte (8 K for data and 8 K for instruction) level one cache. The Pentium Pro includes error correction circuitry (ECC) to correct a one bit error and indicate a two bit error. It provides four additional address lines which makes it possible to access 64 Gbytes of directly addressably memory space.

13.7.1 Pentium Architecture and Functional Description

Fig. 13.53 shows internal architecture of Pentium processor. As shown in the Fig. 13.53, it is a complex processor with many interlocking parts. At the heart of the processor there are two pipelines, the U pipeline and the V pipeline. The U-pipeline can execute all integers and floating point instructions. The V pipeline can execute simple integer instructions and the FXCH floating-point instructions. Further more, during execution, the U and V pipelines are capable of executing two integer instructions at the same time, under special conditions.

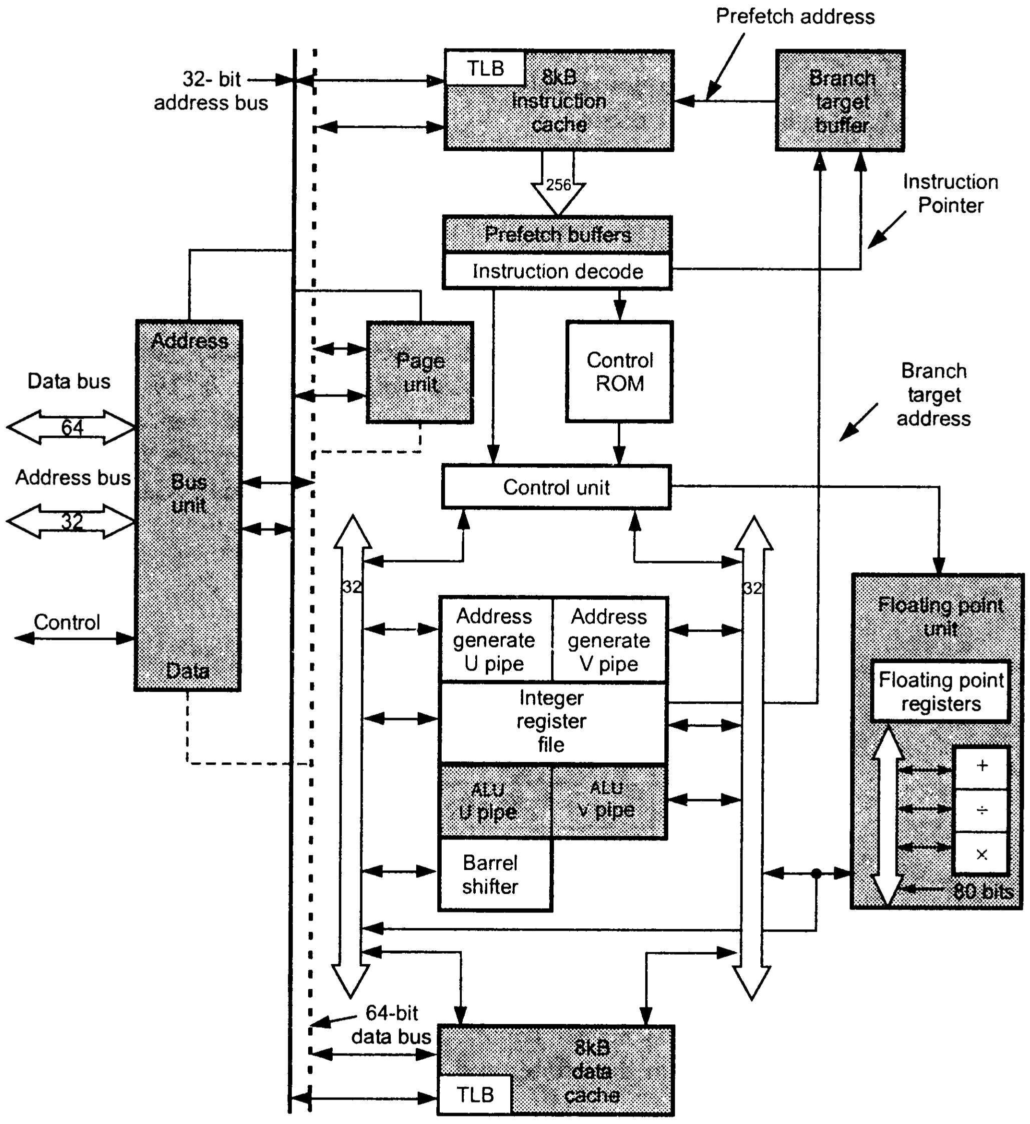


Fig. 13.53 Pentium architecture block diagram

Bus Unit

The Pentium communicates with the outside world via a 32-bit address bus and a 64-bit data bus. The bus unit is capable of performing burst reads and writes of 32 bytes to memory, and through bus cycle pipelining it allows two bus cycles to be in progress simultaneously. It consists of following functional entities :

- **Address drivers and receivers** : During bus cycles the address drivers push the address onto the processor's local address bus (A_{31} : A_3 and BE_7 : BE_0). The address bus transfers addresses back to the Pentium address receivers during cache snoop cycles. Only address lines A_{31} : A_5 are input during cache snoop cycles.
- **Write buffers** : The Pentium processor provides two write buffers, one for each of the two internal execution pipelines. This architecture improves performance when back-to-back writes occur.
- **Data bus transceivers** : The transceivers send data onto the Pentium processors's local data bus during write bus cycles, and receive data into the processor during read bus cycles.
- **Bus control logic** : The Bus Control Logic controls whether a standard or burst bus cycle is to be run. Standard bus cycles are run to access I/O locations and non-cacheable memory locations, as well as cacheable memory write operations. During these bus cycles the transfer size will be either 8, 16 or 32-bits as specified by the instruction. Burst cycles are run by the Pentium processor during cache line fills and during cache write-back bus cycles from the data cache. Four quad-words are transferred during each burst bus cycle.

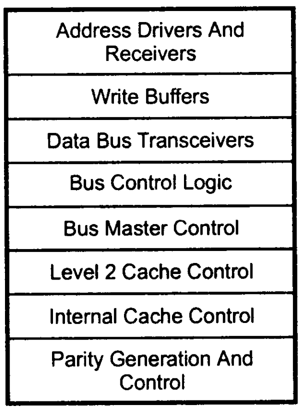


Fig. 13.54 The elements comprising the Pentium processor bus unit

- **Bus master control** : Bus master control signals allow the processor to request the use of the buses from the arbiter and to be preempted by other bus masters in the system.

- **Level two (L2) cache control :** The Pentium processor includes the ability to control a L2 (secondary) external cache operation.
- **Internal cache control :** Internal Cache Control logic monitors input signals to determine when to snoop the address bus and output signals to notify external logic, the results of a snoop operation. It also ensures proper cache coherency.
- **Parity generation and control :** It generates even data parity for each of the eight data paths during write bus cycles and checks parity on read bus cycles. It also generates a parity bit for the address during write bus cycles and checks address parity during external cache snoop operations.

Code cache

An 8 kB instruction cache is used to provide quick access to frequently used instructions. It holds copies of the most frequently used instructions, and it is dedicated to supplying instructions to each of the processor's execution pipelines. The cache is organized as a two-way set associative cache with a line size of 32 bytes. The cache directory is triple ported to allow two simultaneous accesses from the prefetcher and to support snooping. When an instruction is not found in the code (instruction) cache, it is read from the external memory and a copy is placed into the code cache for future references.

Prefetcher

Prefetcher requests for instructions from the code cache. If the requested instruction is not in the cache, a burst bus cycle is run to external memory to perform a cache line fill.

Prefetch buffers

Pentium provides four prefetch buffers. They work as two independent pairs. When instructions are prefetched from the cache, they are placed into one set of prefetch buffers, while the other pair remains idle. When a branch operation is predicted in the Branch Target Buffer (BTB), it requests the predicted branch's target addresses from cache, which are placed in the second pair of buffers that was previously idle. To do this processor gets the new instruction from branch address in no time.

Instruction decode unit

Pentium provides two stage decoding. The instructions are decoded in two stages known as **Decode 1 (D1)** and **Decode 2 (D2)**. During D1, the opcode is decoded in both pipelines to determine whether the two instructions can be paired according to the Pentium processor's pairing rules. If pairing is possible, the two instructions are sent simultaneously to the stage 2 decode. During D2 the address of memory resident operands are calculated.

Control unit

It is also referred to as the Microcode Unit. This control unit consists of the following sub-units :

- Microcode Sequencer
- Microcode Control ROM

This unit interprets the instruction word and microcode entry points fed to it by the Instruction Decode Unit. It handles exceptions, breakpoints and interrupts. In addition, it controls the integer pipelines and floating-point sequences.

Arithmetic/Logic units (ALUs)

Pentium provides two ALUs to perform the arithmetic and logical operations specified by the instructions in their respective pipeline. The ALU for the "U" pipeline can complete and operation prior to the ALU in the "V" pipeline, but the opposite is not true.

Address generators

Pentium provides two Address Generators (one for each pipeline). They generates the address specified by the instructions in their respective pipeline.

Data cache

A separate internal Data Cache holds copies of the most frequently used data requested by the two integer pipelines and the Floating Point Unit. The internal data cache is an 8 kB write-back cache, organized as two-way set associative with 32-byte lines. The Data Cache directory is triple ported to allow simultaneous access from each of the pipelines and to support snooping.

Paging unit

It is enabled by setting the PG bit in CR_0. It translates the linear address (from the address generator) to a physical address. It can handle two linear addresses at the same time to support both pipelines.

Floating-point unit

The floating point unit performs floating point operations. It can accept up to two floating point operations per clock when one of the instruction is an exchange instruction.

13.7.2 Pin Description

The Fig. 13.55 shows the pin diagram of pentium processor and the Fig. 13.56 shows the pin diagram of pentium processor with functional grouping.

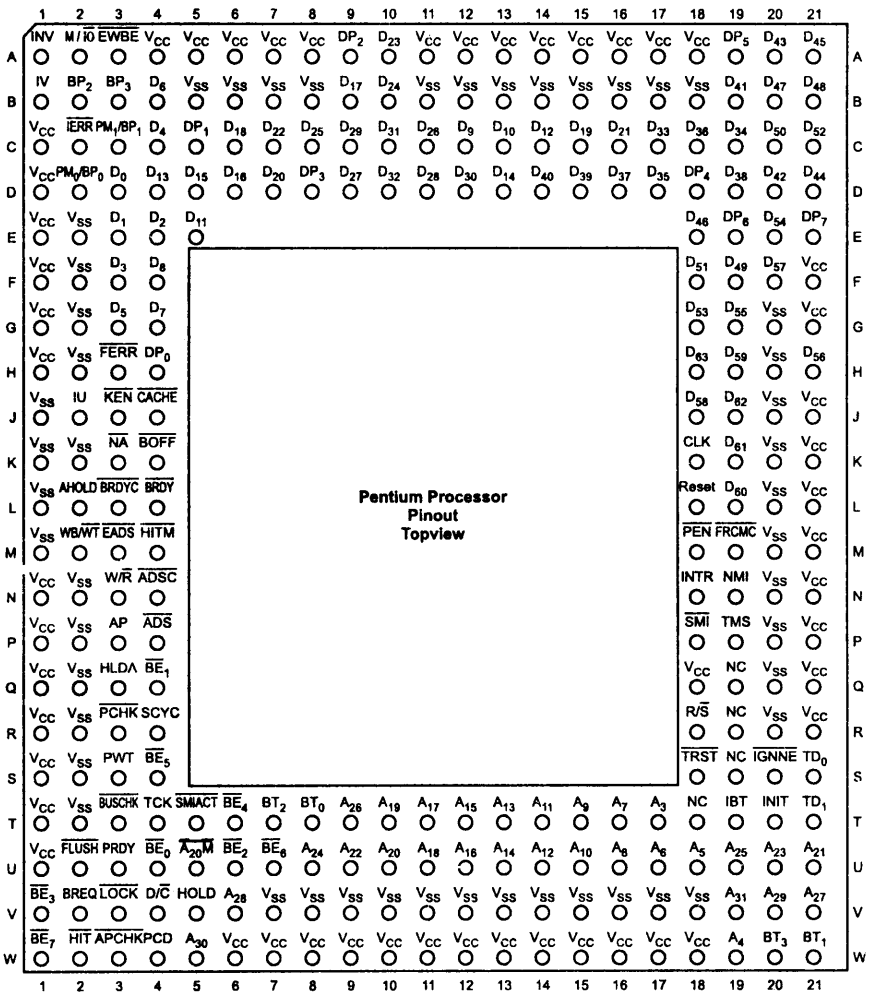

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
A B C D E F G H J K L M N P Q R S T U V W
Pentium Processor
Pinout
Topview
A: INV, M/IO, EWBE, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, DP_2, D_{23}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, DP_5, D_{43}, D_{45}
B: IV, BP_2, BP_3, D_6, V_{SS}, V_{SS}, V_{SS}, V_{SS}, D_{17}, D_{24}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, D_{41}, D_{47}, D_{48}
C: V_{CC}, IERR, PM_1/BP_1, D_4, DP_1, D_{18}, D_{22}, D_{25}, D_{29}, D_{31}, D_{26}, D_9, D_{10}, D_{12}, D_{19}, D_{21}, D_{33}, D_{36}, D_{34}, D_{50}, D_{52}
D: V_{CC}, PM_0/BP_0, D_0, D_{13}, D_{15}, D_{16}, D_{20}, DP_3, D_{27}, D_{32}, D_{28}, D_{30}, D_{14}, D_{40}, D_{39}, D_{37}, D_{35}, DP_4, D_{38}, D_{42}, D_{44}
E: V_{CC}, V_{SS}, D_1, D_2, D_{11} … D_{46}, DP_6, D_{54}, DP_7
F: V_{CC}, V_{SS}, D_3, D_8 … D_{51}, D_{49}, D_{57}, V_{CC}
G: V_{CC}, V_{SS}, D_5, D_7 … D_{53}, D_{55}, V_{SS}, V_{CC}
H: V_{CC}, V_{SS}, FERR, DP_0 … D_{63}, D_{59}, V_{SS}, D_{56}
J: V_{SS}, IU, KEN, CACHE … D_{58}, D_{62}, V_{SS}, V_{CC}
K: V_{SS}, V_{SS}, NA, BOFF … CLK, D_{61}, V_{SS}, V_{CC}
L: V_{SS}, AHOLD, BRDYC, BRDY … Reset, D_{60}, V_{SS}, V_{CC}
M: V_{SS}, WB/WT, EADS, HITM … PEN, FRCMC, V_{SS}, V_{CC}
N: V_{CC}, V_{SS}, W/R, ADSC … INTR, NMI, V_{SS}, V_{CC}
P: V_{CC}, V_{SS}, AP, ADS … SMI, TMS, V_{SS}, V_{CC}
Q: V_{CC}, V_{SS}, HLDA, BE_1 … V_{CC}, NC, V_{SS}, V_{CC}
R: V_{CC}, V_{SS}, PCHK, SCYC … R/S, NC, V_{SS}, V_{CC}
S: V_{CC}, V_{SS}, PWT, BE_5 … TRST, NC, IGNNE, TD_0
T: V_{CC}, V_{SS}, BUSCHK, TCK, SMIACT, BE_4, BT_2, BT_0, A_{26}, A_{19}, A_{17}, A_{15}, A_{13}, A_{11}, A_9, A_7, A_3, NC, IBT, INIT, TD_1
U: V_{CC}, FLUSH, PRDY, BE_0, $A_{20}M$, BE_2, BE_6, A_{24}, A_{22}, A_{20}, A_{18}, A_{16}, A_{14}, A_{12}, A_{10}, A_8, A_6, A_5, A_{25}, A_{23}, A_{21}
V: BE_3, BREQ, LOCK, D/C, HOLD, A_{28}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, V_{SS}, A_{31}, A_{29}, A_{27}
W: BE_7, HIT, APCHK, PCD, A_{30}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, V_{CC}, A_4, BT_3, BT_1

Fig. 13.55 Pin diagram of Pentium processor

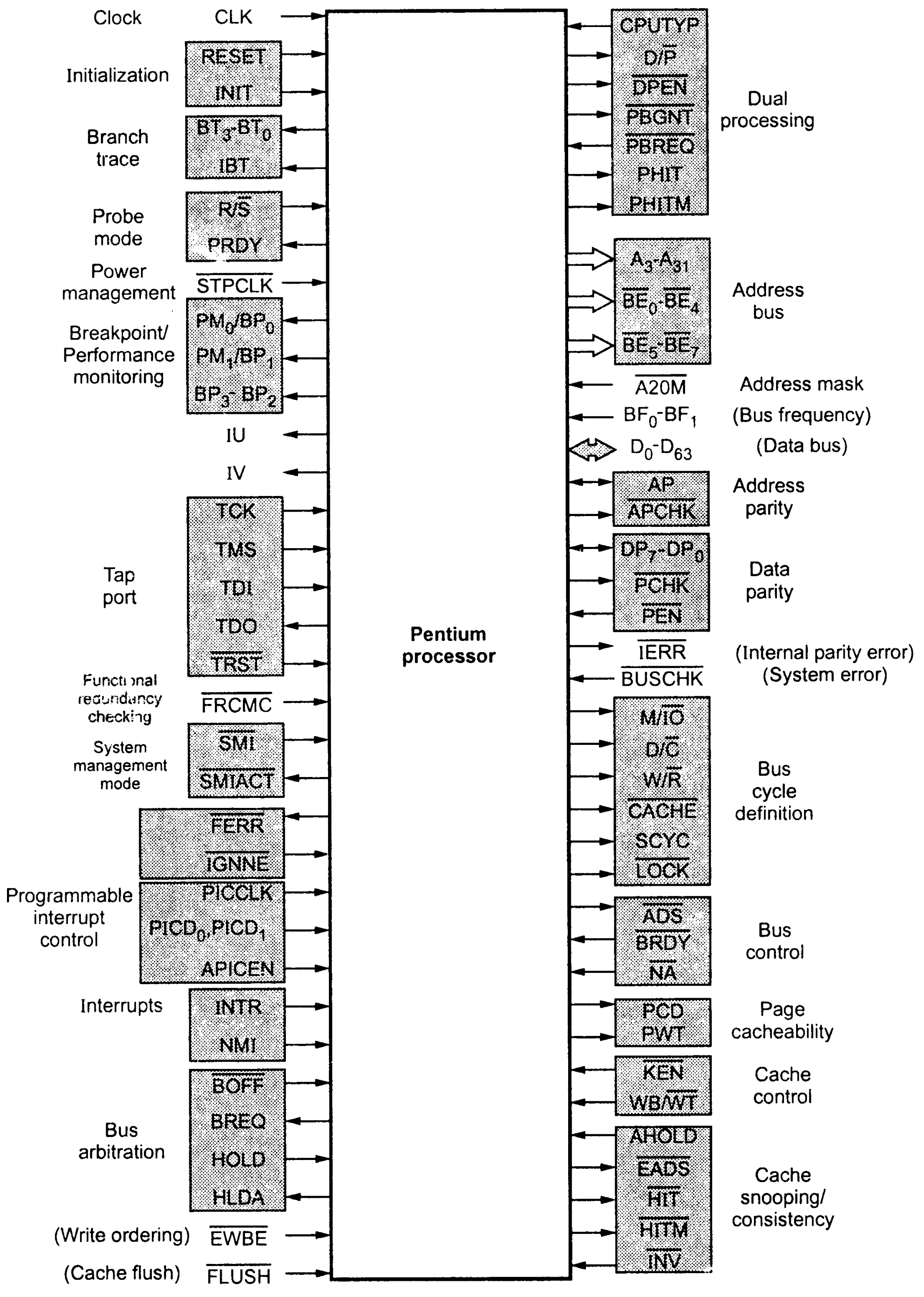

Clock
CLK
Initialization
RESET
INIT
Branch trace
BT_3-BT_0
IBT
Probe mode
R/$\overline{S}$
PRDY
Power management
$\overline{STPCLK}$
Breakpoint/ Performance monitoring
PM_0/BP_0
PM_1/BP_1
BP_3-BP_2
IU
IV
Tap port
TCK
TMS
TDI
TDO
$\overline{TRST}$
Functional redundancy checking
FRCMC
System management mode
$\overline{SMI}$
$\overline{SMIACT}$
$\overline{FERR}$
$\overline{IGNNE}$
Programmable interrupt control
PICCLK
$PICD_0$,$PICD_1$
APICEN
Interrupts
INTR
NMI
Bus arbitration
$\overline{BOFF}$
BREQ
HOLD
HLDA
(Write ordering)
$\overline{EWBE}$
(Cache flush)
$\overline{FLUSH}$
Pentium processor
CPUTYP
D/$\overline{P}$
$\overline{DPEN}$
$\overline{PBGNT}$
$\overline{PBREQ}$
$\overline{PHIT}$
$\overline{PHITM}$
Dual processing
A_3-A_{31}
$\overline{BE_0}$-$\overline{BE_4}$
$\overline{BE_5}$-$\overline{BE_7}$
Address bus
$\overline{A20M}$
Address mask
BF_0-BF_1
(Bus frequency)
D_0-D_{63}
(Data bus)
AP
$\overline{APCHK}$
Address parity
DP_7-DP_0
$\overline{PCHK}$
$\overline{PEN}$
Data parity
$\overline{IERR}$
(Internal parity error)
$\overline{BUSCHK}$
(System error)
M/$\overline{IO}$
D/$\overline{C}$
W/$\overline{R}$
$\overline{CACHE}$
SCYC
$\overline{LOCK}$
Bus cycle definition
$\overline{ADS}$
$\overline{BRDY}$
$\overline{NA}$
Bus control
PCD
PWT
Page cacheability
$\overline{KEN}$
WB/$\overline{WT}$
Cache control
AHOLD
$\overline{EADS}$
$\overline{HIT}$
$\overline{HITM}$
$\overline{INV}$
Cache snooping/ consistency

Fig. 13.56

Pentium hardware signals

Common signals		Changed functionality
$\overline{\text{A20M}}$	I	**Address mask :** When asserted, forces pentium to limit addressable memory to 1 MB to emulate the memory space of the 8086. This signal is active only in the real mode.
A_{31}:A_3	O	These 29 **address lines**, together with the byte enable outputs, form the Pentium's 32-bit address bus. With this 32-bit address a memory space of 4 gigabytes can be accessed.
$\overline{\text{ADS}}$	O	**Address strobe :** When low, indicates the begining of a new bus cycle.
AHOLD	I	**Address hold :** This signal is used to place the Pentium's address bus into a high impedance state so that an inquire cycle can be run.
AP	I/O	**Address parity** is driven by the Pentium processor with even parity information. It is generated in the same clock that the address is driven. Even parity must be driven back to the Pentium processor during inquire cycle on this pin in the same clock.
$\overline{\text{APCHK}}$	O	The **address parity check** status pin is asserted if the Pentium processor has detected a parity error on the address bus during inquire cycles.
APICEN	I	**Advanced Programmable Interrupt Controller (APIC) Enable :** This signal is used to enable or disable the Pentium's internal APIC interrupt controller circuitry.
$\overline{\text{BE}_7}$ - $\overline{\text{BE}_5}$ $\overline{\text{BE}_4}$ - $\overline{\text{BE}_0}$	O I/O	The **byte enable** pins are used to determine which bytes must be written to external memory, or which bytes were requested by the CPU for the current cycle. The byte enables are driven in the same clock as the address lines (A_{31}-A_3). See the purpose of each byte enable **Output** — **Data bus enabled** $\overline{\text{BE}_0}$ — D_0 - D_7 $\overline{\text{BE}_1}$ — D_8 - D_{15} $\overline{\text{BE}_2}$ — D_{16} - D_{23} $\overline{\text{BE}_3}$ — D_{24} - D_{31} $\overline{\text{BE}_4}$ — D_{32} - D_{39} $\overline{\text{BE}_5}$ — D_{40} - D_{47} $\overline{\text{BE}_6}$ — D_{48} - D_{55} $\overline{\text{BE}_7}$ — D_{56} - D_{63}
BF_0, BF_1	I	These inputs are sampled during reset and they control the ratio of bus frequency to CPU core frequency. BF_0 = 1 bus/core ratio = 2/3 BF_0 = 0 bus/core ratio = 1/2
$\overline{\text{BOFF}}$	I	**Back off :** This input causes the processor to terminate any bus cycle currently in process and tri-state its buses. Execution of the interrupted bus cycle is restarted when $\overline{\text{BOFF}}$ goes high.

BP [3 : 2] **PM/BP [1:0]**	O	The **breakpoint** pins (BP_3-P_0) correspond to the debug registers, DR_3-DR_0. These pins externally indicate a breakpoint match when the debug registers are programmed to test for breakpoint matches. BP_1 and BP_0 are multiplexed with the **performance monitoring** pins (PM_1 and PM_0). The PB_1 and PB_0 bits in the Debug Mode Control Register determine if the pins are configured as breakpoint or performance monitoring pins.
$\overline{\text{BRDY}}$		**Burst Ready** : In Pentium $\overline{\text{BRDY}}$ signal is used to indicate that the external device is ready to transfer data.
BREQ	O	**Bus Request** : This signal when active indicates that the pentium has generated a bus request.
BT_3 - BT_0	O	The **branch trace** outputs provide bits 2-0 of the branch target linear address and the default operand size on BT_3. These output become valid during a branch trace special message cycle.
$\overline{\text{BUSCHK}}$	I	The **bus check** input allows the system to signal an unsuccessful completion of a bus cycle. If this pin is sampled active, the Pentium processor will latch the address and control signals in the machine check registers. If, in addition, the MCE bit in CR_4 is set, the Pentium processor will vector to the machine check exception.
$\overline{\text{CACHE}}$	O	The output indicates whether the data associated with the current bus cycle is being read from or written to the data cache.
CLK	I	This is the **clock** signal for the Pentium. It decides the operating frequency of the Pentium. For example, to operate the Pentium at 66 MHz, we apply a 66 MHz clock to this pin.
CPUTYP	I	**CPU type** : It is used to specify the processor type in a dual processor system. CPUTYP = 0 - Primary processor CPUTYP = 1 - Dual processor
D/$\overline{\text{C}}$	O	**Data/Code** : It indicates that the current bus cycle is accessing code (D/$\overline{\text{C}}$ = 0) or data (D/$\overline{\text{C}}$ = 1).
D/$\overline{\text{P}}$	O	**(Dual/Primary)** : This output indicates the processor type in a dual-processing system. D/$\overline{\text{P}}$ = 0 - Primary processor D/$\overline{\text{P}}$ = 1 - Dual processor
D_{63} - D_0	I/O	These are the 64 **data lines** for the processor, Lines D_7-D_0 define the least significant byte of the data bus ; lines D_{63}-D_{56} define the most significant byte of the data bus.
DP_7 - DP_0	I/O	These are the **data parity** pins for the processor. There is one for each byte of the data bus. They are driven by the Pentium processor with even parity information on writes in the same clock as write data. Even parity information must be driven back to the Pentium processor on these pins in the same clock as the data to ensure that the correct parity check status is indicated by the Pentium processor DP_7 applies to D_{63-56}, DP_0 applies to D_7-D_0.
$\overline{\text{DPEN}}$	I/O	**Dual processing enable** : This signal is an output on the dual processor and an input on the primary processor in a dual-processing system.

$\overline{\text{EADS}}$	O	**External Address Strobe** : It is used to indicate that an external address may be read by the address bus during an inquire cycle.
$\overline{\text{EWBE}}$	I	The **external write buffer** empty input, when inactive (high), indicates that a write cycle is pending in the external system.
$\overline{\text{FERR}}$	O	**Floating Point Error** : This output goes low when floating point unit of pentium processor generates an error.
$\overline{\text{FLUSH}}$	I	When asserted, the **cache flush** input forces the Pentium processor to write back all modified lines in the data cache and code cache.
FRCMC	I	The **functional redundancy checking master/checker** mode input is used to determine whether the Pentium processor is configured in master mode or checker mode. When configured as a master, the Pentium processor drives its output pins as required by the bus protocol. When configured as a checker, the Pentium processor tristates all outputs (except $\overline{\text{IERR}}$) and samples the output pins. The configuration as a master/checker is set after RESET and may not be changed other than by a subsequent RESET.
$\overline{\text{HIT}}$	O	The **hit** indication is driven to reflect the outcome of an inquire cycle. If an inquire cycle hits a valid line in either the Pentium processor data or instruction cache, this pin is asserted two clocks after $\overline{\text{EADS}}$ is sampled asserted. If the inquire cycle misses the Pentium processor cache, this pin is negated two clocks after $\overline{\text{EADS}}$.
$\overline{\text{HITM}}$	O	The **hit to a modified line** output is driven to reflect the outcome of an inquire cycle. It is asserted after inquire cycles which resulted in a hit to a modified line in the data cache. It is used to inhibit another bus master from accessing the data until the line is completely written back.
HLDA	O	**Hold Acknowledge** : This output goes high in response to HOLD request to indicate that the pentium has been placed in the hold state.
HOLD	I	When high, the Pentium tri-states its bus signals and activates HLDA.
IBT	O	Instruction branch taken indicates that the Pentium has taken an instruction branch.
$\overline{\text{IERR}}$	O	The **internal error** pin is used to indicate two types of errors, internal parity errors and functional redundancy errors. If a parity error occurs on a read from an internal array, the Pentium processor will assert the $\overline{\text{IERR}}$ pin for one clock and then shutdown. If the Pentium processor is configured as a checker and a mismatch occurs between the value sampled on the pins and the corresponding value computed internally, the Pentium processor will assert $\overline{\text{IERR}}$ two clocks after the mismatched value is returned.
$\overline{\text{IGNNE}}$	I	**Ignore Numeric Exception** : A low on this input allows the processor to continue executing floating-point instructions, even if an error is generated.

INIT	I	The Pentium processor **initialization** input pin forces the Pentium processor to begin execution in a known state. The processor state after INIT is the same as the state after RESET except that the internal caches, write buffers, and floating point registers retain the values they had prior to INIT. If INIT is sampled high when RESET transitions from high to low, the Pentium processor will perform built-in self test prior to the start of program execution.
INV	I	The **invalidation** input determines the final cache line state (shared or invalidated) in case of an inquire cycle hit.
IU	O	This output goes high for one clock cycle each time an instruction completes in U pipeline.
IV	O	This output goes high for one clock cycle each time an instruction completes in V pipeline.
$\overline{\text{KEN}}$	I	**Cache enable** : This signal is used to determine whether current cycle is cacheable or not.
$\overline{\text{LOCK}}$	O	**Bus lock** : This signal goes low to indicate that the current bus cycle is locked and may not be interrupted by any other bus master.
M/$\overline{\text{IO}}$	O	**(Memory/Input-Output)** : This signal indicates the type of current. bus cycle. M/$\overline{\text{IO}}$ = 0 - I/O cycle M/$\overline{\text{IO}}$ = 1 - Memory cycle
$\overline{\text{NA}}$	I	An active next address input indicates that the external memory system is ready to accept a new bus cycle although all data transfers for the current cycle have not yet completed.
NMI	I	This is a non-maskable interrupt signal of pentium.
$\overline{\text{PBGNT}}$	O	**Private bus grant** : This signal is used in a dual-processing system to indicate when private bus arbitration is allowed.
$\overline{\text{PBREQ}}$	O	**Private bus request** : This signal is used to request a private bus operation in a dual-processing system.
PCD	O	The **page cache disable** pin reflects the state of the PCD bit in CR_3, the Page Directory Entry, or the Page Table Entry. The purpose of PCD is to provide an external cacheability indication on a page by page basis.
$\overline{\text{PCHK}}$		**Data parity check** : This output goes low, if the Pentium detects a parity error on the data bus. But in Pentium parity checking has been extended ; if $\overline{\text{PEN}}$ is also asserted low during the same cycle, the Pentium will save a copy of the address and control signals in an internal machine check register. Additionally, if the MCE bit in the new CR4 register is set, a machine check exception is generated.
$\overline{\text{PEN}}$	I	**Parity enable** : If this input is low during the same cycle a parity error detected, the Pentium will save a copy of address and control signals in an internal machine check register.
$\overline{\text{PHIT}}$	O	**Private hit** : It is used to maintain the local cache coherency in a dual processor sytem.

$\overline{\text{PHITM}}$	O	Private modified hit : It is used in conjunction with $\overline{\text{PHIT}}$ to maintain the local cache coherency is a dual-processor system.
PICCHK	I	Programmable interrupt controller clock : This signal controls the serial data rate in the internal APIC interrupt controller.
PIC D_0, PIC D_1	I	Programmable interrupt controller data : These two signals are used to exchange data with the internal APIC interrupt controller.
PRDY	O	The **probe ready** output pin indicates that the processor has stopped normal execution in response to the R/$\overline{\text{S}}$ pin going active, or probe Mode being entered. This output is used for debugging purpose.
PWT	O	The **page write through** pin reflects the state of the PWT bit in CR_3, the page directory entry, or the page table entry. The PWT pin is used to provide an external write back indication on a page-by-page basis.
R/$\overline{\text{S}}$	I	The **run/stop** input is an asynchronous, edge-sensitive interrupt used to stop the normal execution of the processor and place it into an idle state.
RESET	I	This signal forces Pentium to initialize its registers to known state, invalidate code and data cache, and fetch its first instruction from address FFFFFFF0H. This signal must be active for atleast 1 ms after power on.
SCYC	O	The **split cycle** output is asserted during misaligned LOCKed transfers to indicate that more than two cycles will be locked together. This signal is defined for locked cycles only.
$\overline{\text{SMI}}$	I	The **system management interrupt** causes a system management interrupt request to be latched internally. When the latched $\overline{\text{SMI}}$ is recognized on an instruction boundary, the processor enters System Management Mode.
$\overline{\text{SMIACT}}$	O	An active **system management interrupt** active output indicates that the processor is operating in System Management Mode.
$\overline{\text{STPCLK}}$	I	Stop clock : When low, this signal causes the Pentium to stop its internal clock.
TCK	I	The **testability clock** input provides the clocking function for the Pentium processor boundary scan in accordance with the IEEE Boundary Scan interface (Standard 1149.1).
TDI	I	The **test data input** is a serial input for the test logic. TAP instuctions and data are shifted into the Pentium processor on the TD pin on the rising edge of TCK when the TAP controller is in an appropriate state.
TDO	O	**Test Data Output** : This signal is used to send serial test information on the falling edge of TCK.

TMS	I	The value of the **test mode select** input signal sampled at the rising edge of TCK controls the sequence of TAP controller state changes.
$\overline{\text{TRST}}$	I	When asserted, the test reset input allows the TAP controller to be asynchronously initialized.
W/$\overline{\text{R}}$	O	**Write/Read** : This signal indicates whether the current bus cycle is read cycle or write cycle. W/$\overline{\text{R}}$ = 0 - Read cycle W/$\overline{\text{R}}$ = 1 - Write cycle
WB/$\overline{\text{WT}}$	I	The **write back/write through** input allows a data cache line to be defined as write back or write through on a line-by-line basis.

Table 13.11

Note :Non-shaded signals are of Pentium processor (510\60, 567\66) and shaded signals are the additional signal provided in Pentium processor (610\75, 735\90, 815\100, 1000\120, 1110\133).

Pin grouping according to function

Table 13.12 organizes the pins with respect to their function.

Function	Pins
Clock	CLK
Initialization	RESET, INIT
Address bus	A_{31}-A_3, $\overline{BE_7}$-$\overline{BE_0}$
Address mask	$\overline{\text{A20M}}$
Data bus	D_{63}-D_0
Address parity	AP, $\overline{\text{APCHK}}$
Data parity	DP_7-DP_0, $\overline{\text{PCHK}}$, $\overline{\text{PEN}}$
Internal parity error	$\overline{\text{IERR}}$
system error	$\overline{\text{BUSCHK}}$
Bus cycle definition	M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, W/$\overline{\text{R}}$, $\overline{\text{CACHE}}$, SCYC, $\overline{\text{LOCK}}$
Bus control	$\overline{\text{ADS}}$, $\overline{\text{BRDY}}$, $\overline{\text{NA}}$
Page cacheability	PCD, PWT
Cache control	$\overline{\text{KEN}}$, WB/$\overline{\text{WT}}$
Cache snooping/consistency	AHOLD, $\overline{\text{EADS}}$, $\overline{\text{HIT}}$,$\overline{\text{HITM}}$, INV
Cache flush	$\overline{\text{FLUSH}}$

Write ordering	$\overline{\text{EWBE}}$
Bus arbitration	$\overline{\text{BOFF}}$, BREQ, HOLD, HLDA
Interrupts	INTR, NMI
Floating point error reporting	$\overline{\text{FERR}}$, $\overline{\text{IGNNE}}$
System management mode	$\overline{\text{SMI}}$, $\overline{\text{SMIACT}}$
Functional redundancy checking	$\overline{\text{FRCMC}}$ ($\overline{\text{IERR}}$)
TAP Port	TCK, TMS, TD_1, TD_0, $\overline{\text{TRST}}$
Breakpoint/Performance monitoring	PM/BP_0, PM/BP_1, BP_3-BP_2
Power management	$\overline{\text{STPCLK}}$
Probe mode	$R/\overline{S}$, PRDY
Branch trace	BT_3-BT_0, IBT
Dual processing	CPUTYP, D/P, $\overline{\text{DPEN}}$, $\overline{\text{PBGNT}}$, $\overline{\text{PBREQ}}$, PHIT, PHITM
Programmable interrupt control	PICCLK, $PICD_0$, $PICD_1$, APICEN
Bus frequency	BF_0 - BF_1

Table 13.12 Pin functional grouping

13.7.3 The Memory System

The memory system for the Pentium processor is 4 Gbytes in size, same as in the 80386DX and 80486 microprocessors. The only difference is in the width of the memory data bus. The Pentium uses a 64-bit data bus to address memory organised in eight banks that each contain 512 Mbytes of data. This is illustrated in Fig. 13.57. As shown in the Fig. 13.57, the pentium memory system is divided into eight banks that each store a byte of data with a parity bit. The memory system has 4 Gbytes memory, beginning of location 00000000H and ending at location FFFFFFFFH. The Bank selection is accomplished by bank enable signals ($\overline{\text{BF7}}$ - $\overline{\text{BE0}}$), one for each bank. These separate memory banks allow the Pentium to access any single byte, word, double word or quad word with one memory transfer cycle.

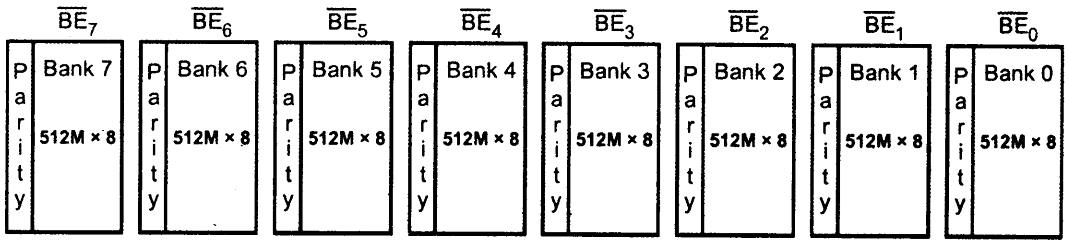


Fig. 13.57 Pentium memory system organisation

In Pentium, the double-precision floating point number can be retrieved in one read cycle because a double-precision floating point number is 64-bit wide and data bus of pentium is also 64-bit wide.

The pentium has an ability to check and generate parity for the address bus (A_{31} - A_5) during certain operations. The AP signal provides the system with parity information and the $\overline{\text{APCHK}}$ indicates a bad parity check for the address bus. The Pentium takes no action when an address parity error is detected. Therefore, in Pentium the error must be assessed by the system and the system must take appropriate action (an interrupt), if so desired.

13.7.4 Input/Output System

The I/O system for Pentium is identical to the 80386 microprocessor and it is completely compatible with earilier Intel microprocessors. In Pentium, the I/O port number appears on address lines A_{15} - A_3 with the bank enable signals used to select the actual memory banks used for the I/O transfer.

Like 80386 microprocessor, the I/O privilege information is added to the TSS segment when the Pentium is operated in the protected mode. This provides I/O protection and allows I/O ports to be selectively inhibited. If the inhibited I/O location is accessed, the Pentium generates a type 13 interrupt to indicate the I/O privilege violation.

13.7.5 System Timings

The system timings are important and must be understood inorder to interface memory or I/O device to the microprocessor. In this section we study the Pentium system timings. Let us consider the non pipelined read cycle for the Pentium processor. The Fig. 13.58(a) shows the timings diagram for it. The basic nonpipelined memory cycle of

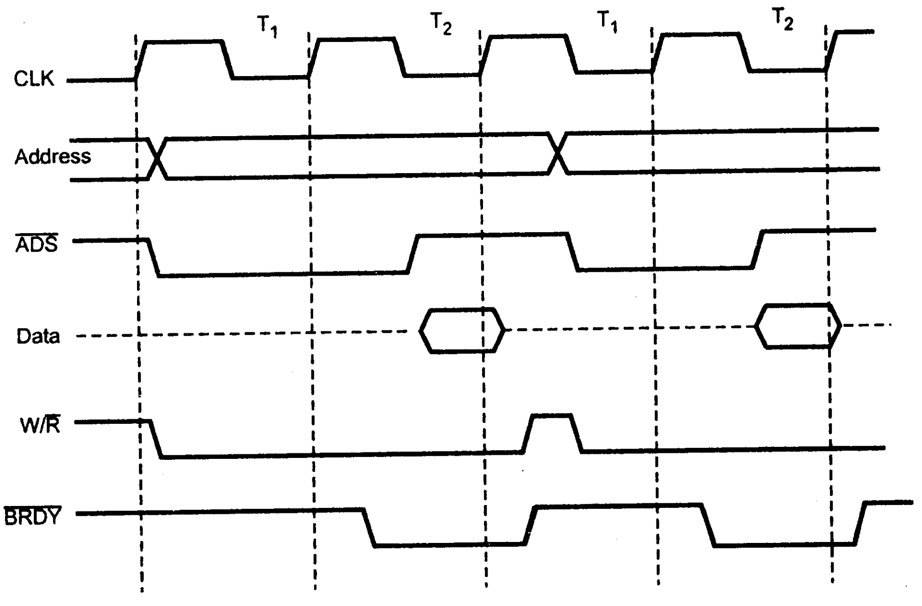


Fig. 13.58 (a) Nonpipelined read cycle for the Pentium processor

Pentium consists of two clocks : T_1 and T_2. During T_1, the Pentium issues the $\overline{ADS}$, $W/\overline{R}$, address and $M/\overline{IO}$ signals. We can generate $\overline{MWTC}$, $\overline{MRDC}$, $\overline{IOWC}$ and $\overline{IORC}$ (memory and I/O control signals) from $\overline{BRDY}$, $W/\overline{R}$, $\overline{ADS}$ and $M/\overline{IO}$ with the help of flip-flop and quad 2 : 1 multiplexer, as shown in the Fig. 13.58 (b).

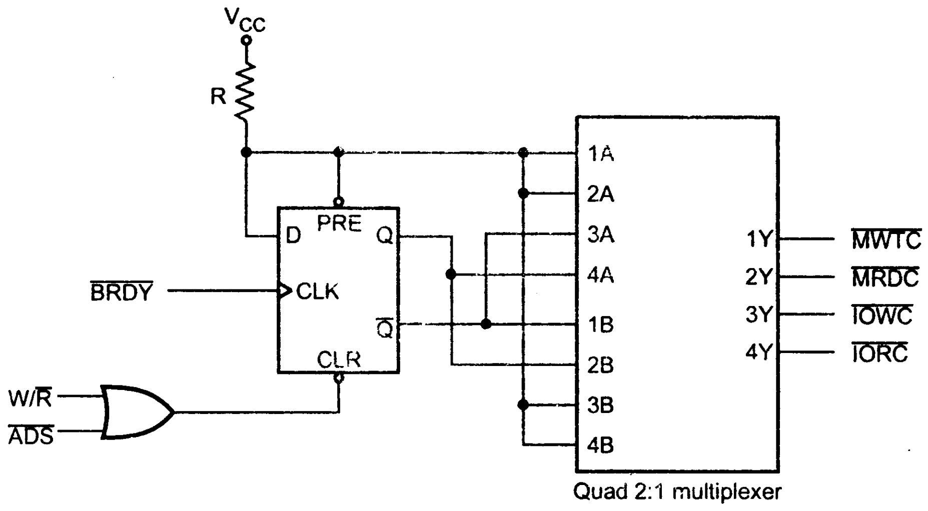


Fig. 13.58 (b) Circuit to generate separate I/O and memory control signals

During T_2, the data bus is sampled in synchronisation with the end of T_2 at the rising edge of the clock pulse. For Pentium the setup time before the clock is given as 3.8 ns, and the hold time after the clock is given as 2 ns. Therefore, the data window around the clock is 5.8 ns. The address appears within maximum 8 ns from the start of T_1. For Pentium operating at 66 MHz has 1 clock period of 15.15 ns. Therefore, the maximum memory access time for Pentium operating at 66 MHz without wait state can be given as

$$\begin{aligned} M_{ACC\ (max)} &= 2 \times \text{Clock period} - \text{Address setup} - \text{Data setup} \\ &= 2 \times 15.15 \text{ ns} - 8 \text{ ns} - 3.8 \text{ ns} \\ &= 18.5 \text{ ns} \end{aligned}$$

This time is enough to access SRAM; however it is not sufficient to access DRAM. To access DRAM wait states must be insertd in the read cycle. Wait states are inserted into the read cycle by controlling the $\overline{BRDY}$ input to the Pentium. If $\overline{BRDY}$ signal is sampled logic 1 at the end of T_2, the wait state is inserted into the cycle. The insertion of wait state/s lengthens the access time, allowing additional time to the memory to access data. This is illustrated in Fig. 13.59. Each wait state increases the memory access time by 15.15 ns.

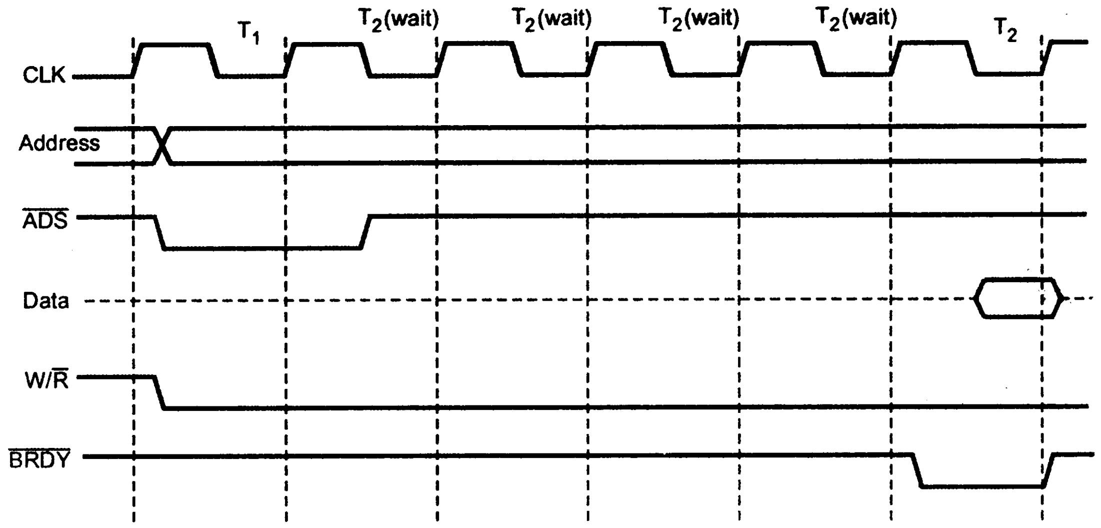


Fig. 13.59 Pentium timing diagram with four wait states

With insertion of four wait states the memory access time becomes

$$M_{ACC\ (max)} \text{ with 4 wait states } = 18.5 \text{ ns} + 4 \times 15.15 \text{ ns}$$

$$= 79.1 \text{ ns}$$

In Pentium, the memory data can be read using burst cycle. It is the most efficient way of accessing data. The burst cycle in the Pentium transfers four 64-bit numbers per

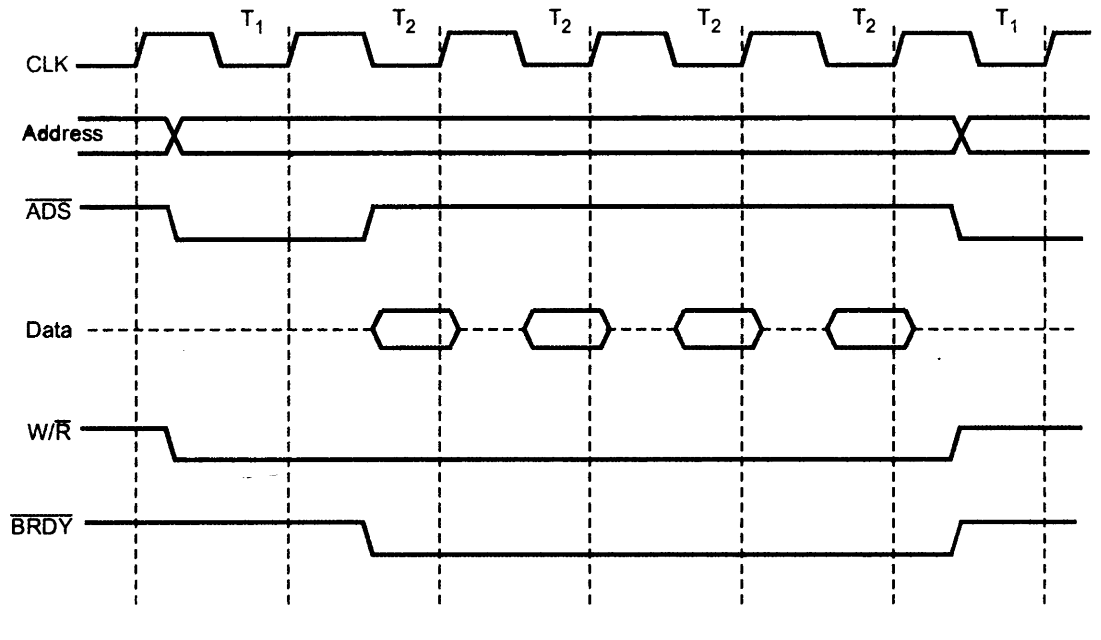


Fig. 13.60 Burst cycle for Pentium processor

burst cycle in five clocking periods. This is illustrated in Fig. 13.60. Therefore, a brust cycle without wait states requires average ([15.15 ns × 5]/4) 18.94 ns for each memory data transfer.

13.7.6 New Pentium Instructions

The Table 13.13 shows new instructions added to the Pentium instruction set.

No.	Instruction	Function
1.	CMPXCHG 8B	Compare and exchange eight bytes
2.	CPUID	Return CPU identification code
3.	RDTSC	Read time stamp counter
4.	RDMSR	Read model specific register
5.	WRMSR	Write model specific register
6.	RSM	Return from system management interrupt

Table 13.13 New Pentium instructions

CMPXCHG8B : This instruction is an extension of CMPXCHG instruction of 80486. It compares the 64-bit number stored in EDX and EAX with the contents of a 64-bit memory location or register pair. This instruction performs as follows :

- Compares the 64-bit value in EDX : EAX with the value specified in memory.
- If equal, the value in ECS : EBX is stored into the specified memory operand and zero flag is set.
- If unequal, the contents of the memory operand is copied into EDX : EAX.

CPUID : This instruction reads the CPU identification code and other information from the Pentium. The Table 13.14 shows different information returned from the CPUID instruction for 0 and 1 input values of EAX.

Value in EAX	Returned information
EAX = 0	EAX = 1, and the ASCII representation of string "Genuine Intel" (vendor identification) is returned in EBX : EDX : ECX
EAX = 1	EAX (bits 3-0) = Stepping ID EAX (bits 7-4) = Model EAX (bits 11-8) = Family EAX (bits 13-12) = Type EAX (bits 31-14) = Reserved

	EDX (bit 0) = CPU returns FPU EDX (bit 1) = Enhanced 8086 virtual mode supported EDX (bit 2) = I/O breakpoint supported EDX (bit 3) = Page size extensions supported EDX (bit 4) = Time stamp counter (TSC) supported EDX (bit 5) = Pentium style MSR supported EDX (bit 6)= Reserved EDX (bit 7) = Machine check exception supported EDX (bit 8) = CMPXCHG 8B supported EDX (bit 9) = 3.3 V microprocessor EDX (bit 31-10) = Reserved
EAX = Anyother value	EAX, EBX, ECX and EDX are all returned with zeros

Table 13.14 Input values and returned information for CPUID instruction

Let us see the simple ALP program for Pentium using CPUID instruction to display the vendor information.

Program :

```
    .MODEL TINY             ; selects the model
    .586                        ; processor is Pentium
    .CODE
    DISP_CH MACRO           ; display character macro
         MOV AH, 02H        ; load function number
         INT 21H
         SHR EDX, 8
         ENDM
.STARTUP
         MOV EAX, 0
         CPUID              ; read CPUID from Pentium
         XCHG EBX, EDX      ; display first from EBX
         DISP_CH
         DISP_CH
         DISP_CH
         DISP_CH
         MOV EDX, EBX       ;  display from EDX
         DISP_CH
         DISP_CH
         DISP_CH
         DISP_CH
         MOV EDX, ECX       ; display from ECX
         DISP_CH
         DISP_CH
         DISP_CH
```

```
        DISP_CH
.EXIT
        END
```

RDTSC : This instruction reads the current value of an internal 64-bit time stamp counter into EDX : EAX that is updated every clock cycle.

Let us the two macros which can be used to save current count i.e. to start the event and find the time elasped in microseconds from start of the event respectively.

Macro to save current count from the time stamp counter

```
START     MACRO     S_COUNT
                PUSH EAX
                PUSH EDX
                RDTSC
                MOV  DWORD PTR S_COUNT, EAX
                MOV  DWORD PTR S_COUNT +4, EDX
                               ; save current count

                POP  EDX
                POP  EAX
                ENDM
```

Macro to find time elapsed from start of event

```
T_FIND    MACRO S_COUNT, FREQ
                PUSH ECX
                PUSH EDX
                RDTSC
                SUB  EAX, DWORD PTR S_COUNT ; [find the difference]
                SBB  EDX, DWORD PTR S_COUNT + 4
                MOV  ECS, FREQ                ; convert to
microseconds
                DIV  ECX
                POP  EDX
                POP  ECX
                ENDM
```

The parameters passed in both macro are quadword wide so they must be defined with the DQ directive. The 64-bit S_COUNT memory location is used to store an initial value of time stamp counter in the memory. The parameter FREQ is used to pass the clock frequency of the Pentium in MHz to the macro. The first macro (START) saves the current count of the time stamp counter and second macro (T_FIND) determines the time elasped in microseconds from start of the event and returns this value in EAX register.

RDMSR and WRMSR

These two instructions are used to access the contents of a selected 64-bit modelspecific register (MSR).

When either the RDMSR or WRMSR instruction is executed, the value in the ECX register specifies access to one of the processor's modelspecific registers. If reading from a specified register, its contents are deposited into the register pair EDX:EAX. When writing to the register, the current contents of EDX:EAX are written to the target MSR. The only MSR registers defined by Intel in their public-domain documentation are the :

- Machine check address register, or MCAR (ECX = 00h)
- Machine check type register, or MCTR (ECX = 01h)
- Test register 12, or TR12 (ECX = 0Eh)

RSM : This instruction returns from a system management interrupt. It causes the processor's state to be restored from system management RAM (SMRAM). The only registers not affected are the model-specific registers. Restoring the register set allows the processor to resume the interrupted application at the point interruption. If, during the register reload, any of the processor state information is found to be faulty, the processor will enter the shutdown state.

13.7.7 Pentium RISC Features

Because of the advances in microelectronic manufacturing technology, a number of changes in the computer architectures are taking place from the last decade. It became possible to cram a large logic into the small space of silicon wafer. The new computers were designed which use processors with complex instructions and addressing modes, which we call as **Complex Instruction Set Computer (CISC)**. But the problem arised with the CISC machines was their instructions required multiple clock cycles to execute because of cramming of large logic into a single package. This degraded the performance of CISC machines. This problem is solved by a new design technique called **Reduced Instruction Set Computer (RISC)**. The important factor considered while designing RISC machines is that it uses fewer instructions and simpler addressing modes. Because of the fewer instructions, the number of operations are reduced and can easily be implemented on silicon wafer which results in increase in the speed and hence improves the performance.

In this section, we will discuss the features of RISC processor, which of them are applied to design Pentium processor.

1. Reduced accesses to main memory

Ideally, computer memory should be fast, large and inexpensive. Unfortunately, it is impossible to meet all the three of these requirements simultaneously. Increased speed and size are achieved at increased cost. Very fast memory of system can be achieved if SRAM chips are used. These chips are expensive and for the cost reason it is impracticable to build a large main memory using SRAM chips. The only alternative is to use DRAM chips for large main memories. Processor fetches the code and data from the main memory to execute the program. The DRAMs which form the main memory are slower devices. So it is necessary to insert wait states in memory read/write cycles. This reduces the speed of execution. Thus, though the great advances are made in memory technology, processors are much faster than memories. Since the speed of operation of processor is much faster than that of memory, the processor has to wait during each memory access.

The RISC design includes a technique which reduces the number of accesses to main memory. Most of the computer programs work with only small sections of code and data at a particular time. In the memory system small section of SRAM is added along with main memory, referred to as **cache memory**. The program which is to be executed is loaded in the main memory, but the part of program (code) and data that work at a particular time is usually accessed from the cache memory. This is accomplished by loading the active part of code and data from main memory to cache memory. Whenever the processor tries to read data from main memory, the cache is examined first. The addresses are stored in the caches. If one of these addresses matches the address being used for the memory read, the cache will supply the data, which is called **cache hit**. Generally, cache is ten times faster than the main memory. When the required data is not found in the cache, it is called **cache miss** and the processor has to access main memory in this situation. After a cache miss, a copy of the new data is written into the cache, so that the data will be obtained whenever needed.

Pentium contains two caches, 8 kB each. An 8 kB instruction cache stores frequently used instructions and an 8 kB data cache stores frequently used data. Initially, each cache is empty and is filled as program executed.

2. Simple instructions and addressing modes (RISC feature, not available in Pentium)

When the processor uses simple and fewer instructions and addressing modes, the implementation of operations on silicon wafer is easier. Also it reduces the complexity of the instruction decoder, the addressing unit and the execution unit. In this case, the machine can be operated at a higher clock rate because work which is to be done in each clock period is less. Practically, it is possible to use simple, fewer instructions and addressing mode because after a research, the computer scientists came to know that the programmers use only a small subset of the instructions available on the processor they are using.

From this point of view, Pentium is not a RISC processor, but it is a CISC processor. The reason is, Pentium should remain compatible with the installed software of entire 80×86 family. Each and every instruction and addressing mode of the previous processor, 80486 should be kept as it is.

3. Large sets of registers and make good use of them

In the first feature, we have seen how the number of accesses to the memory affects the performance of a processor. Similar to this, the number of registers available in processor can affect the performance of it. When a complex calculation is to be performed by a processor, it may require the use of several data values. If all these data values are stored in a memory, then during the calculations, a number of memory accesses are required to use those data values. But when number of registers are available in processor, instead of storing data values in memory, they can be stored in registers. Accessing the internal registers for reading data values during calculations is much faster than accessing memory for the same purpose. Thus, it is always good to have large sets of internal registers for the processor.

Pentium has the following sets of registers.

i) Seven general purpose registers, all of them are 32-bits wide.

ii) Six segment registers, all of them are 16-bits wide.

iii) A 32-bit stack pointer.

iv) Eight floating point registers, all of them are 80-bits wide.

Thus, the pentium has a large set of registers (like a RISC).

4. Pipelining

We know that more than one clock cycles are involved in the instruction cycle. These clock cycles are required to perform various steps in the instruction execution. These steps belong to various processing stages in the instruction cycle. These are

- S_1 - Fetch (F) : Read instruction from the memory.
- S_2 - Decode (D) : Decode the opcode and fetch source operand (s) if necessary.
- S_3 - Execute (E) : Perform the operation specified by the instruction.
- S_4 - Store (S) : Store the result in the destination.

Usually, instruction is executed by performing above mentioned stages one after the other. When these stages for several instructions are performed simultaneously to reduce overall processing time, the processing is called **instruction pipelining.**

Refer Fig. 13.61. Here, instruction processing is divided into four stages hence it is known as **four-stage instruction pipeline.** With this subdivision and assuming equal duration for each stage we can reduce thc execution time for 4 instructions from 16 time units to 7 time units.

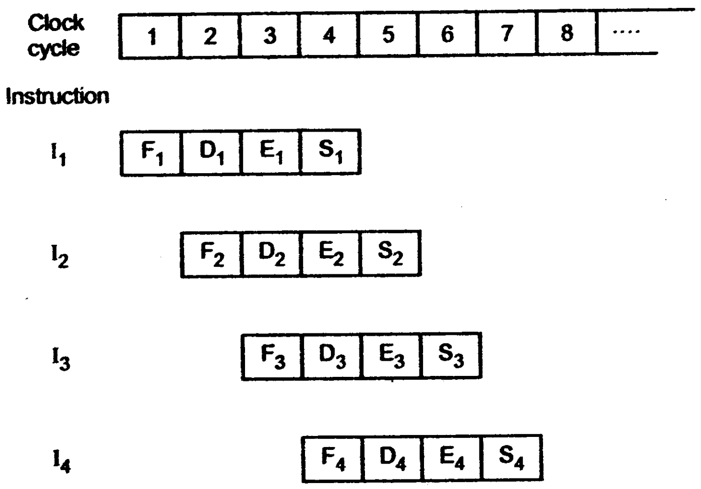


Fig. 13.61 Four stage instruction pipelining

In this instruction pipelining four instructions are in progress at any given time. This means that four distinct hardware units are needed, as shown in Fig. 13.62. These units are implemented such that they are capable of performing their tasks simultaneously and without interfering with one another. Information from the stage is passed to the next stage with the help of buffers.

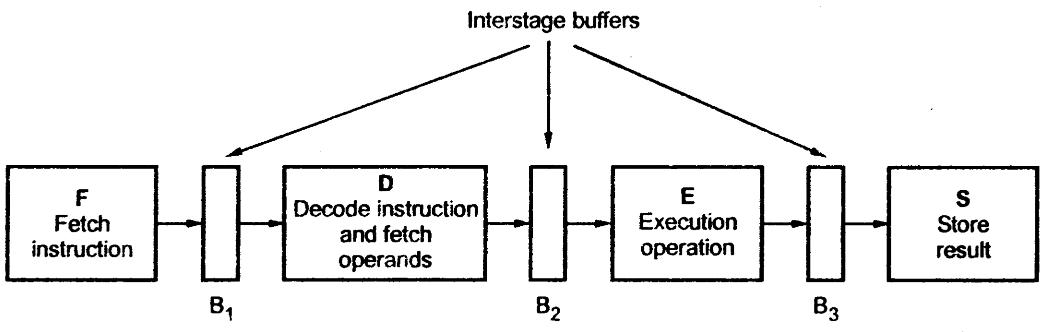


Fig. 13.62 Hardware organisation for four-stage instruction pipeline

Coming to the point of performance analysis we can say that pipelining can reduce effective number of clock cycles required for instruction execution and thus increases the rate of executing instructions significantly. It approaches the ideal value of required clock cycles per instructions as shown in Fig. 13.61. However in practice, this ideal value cannot be attained for a variety of reasons.

The performance of a processor improves tremendously because of pipelining. There are two types of pipelines in Pentium, instruction pipelines (U and V, covered in further section) and bus cycle pipeline that performs special types of bus cycles. The instruction pipelines include five stages. They operate independently.

Also, Pentium employs a branch prediction technique (explained in detail in further section). Normally, there is a flow of instructions through U and V pipelines. With a branch prediction technique, Pentium predicts whether to change normal program flow or not. Thus this technique helps to keep a steady stream of instructions flowing into the pipelines. Of course, this increases the rate of instruction execution, and hence the performance of the Pentium improves. This feature of Pentium is very like a RISC machine.

5. Extensive utilization of the compiler

When a program is written in higher level language (e.g. C language), during compilation, each statement within a program is converted into assembly language instruction. When we use a Pentium compiler, the advances in the Pentium architecture can be utilized with the optimizations on the assembly language code. Some examples of it are given on next page.

a) Arrange some pairs of instructions such that they will execute in parallel in the floating-point unit or dual-integer pipelines.

b) Reorder the instructions such that the Pentium's branch prediction technique is utilized properly.

c) If possible, replace an instruction with an equivalent instruction which requires lesser number of clock cycles or the number of bytes of machine code. For example, MOV EAX, 0 can be replaced by SUB EAX, EAX.

d) Use the instruction/data cache or algorithms to allocate the minimum number of processor registers during parcing of an arithmetic statement.

Thus, a properly written Pentium compiler helps to achieve a high performance like in a RISC or CISC machine.

From all above discussion, it is clear that the Pentium contains both RISC and CISC characteristics.

13.7.8 Pentium Super-Scalar Architecture

Processors capable of parallel instruction execution of multiple instructions are known as **superscalar processors.** The Pentium is capable, under special circumstances, of executing two integer or two floating point instructions simultaneously and thus it support superscaler architecture. **However, there are restrictions placed on a pair of integer instructions attempting parallel execution.**

For floating point instructions there is a restriction of which instructions should execute as a first instruction of a pair and which is the second instruction of the pair.

First instruction in the pair	Second instruction in the pair
FLD <single/double>	FXCH
FLD ST (i), FADD, FSUB	
FMUL, FDIV, FCOM, FUCOM	
FTST, FABS, FCHS	

The modern compilers play the important role in achieving the performance of Pentium processor at superscalar level. They do the ordering of the instructions during code generation to make pair of instructions without any data dependency and make allowable combinations of integer and floating-point instructions for simultaneous execution.

13.7.9 Pipelining

In the previous section we have seen that the rate of instruction execution can be improved with a pipelining. In Pentium, there are two instruction pipelines, U pipeline and V pipeline. These are five-stage pipelines and operate independently. These five stages with their order are as follows :

1. PF	Prefetch
2. D1	Instruction Decode
3. D2	Address Generate
4. EX	Execute, Cache and ALU Access
5. WB	Writeback

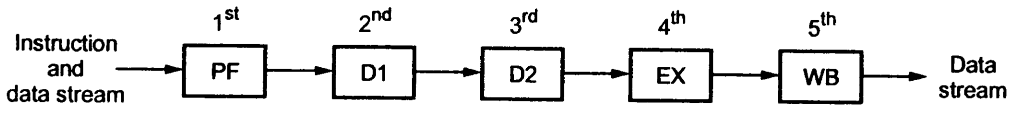


Fig. 13.63 Stages in U and V instruction pipelines

Both pipelines U and V include the above five stages. The U pipeline can execute any processor instruction, but the V pipeline only executes simple instructions. An instruction, which does not require microcode control to execute and generally takes one clock cycle to complete is referred to as a simple instruction. For example, register-to-register MOVs, INC, DEC, near conditional jumps (e.g. JZ, JNZ etc.). It is to be noted that some simple instructions may take two or three clock cycles. These are arithmetic and logical instructions that use both register and memory operands.

Refer Fig. 13.64 which shows the pipelined instruction execution.

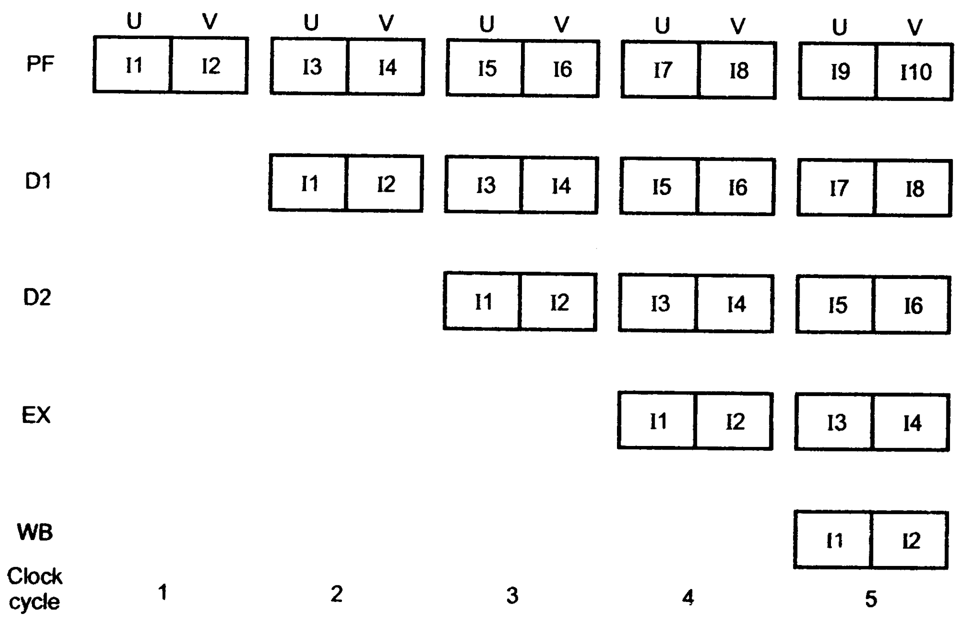


Fig. 13.64 Pipelined instruction execution

The following sequence of steps explains the pipelined instruction execution in Pentium.

1. **Prefetch (PF) stage :** Instructions are prefetched from the instruction cache or memory and fed into the PF stage of both the pipelines U and V.
2. **Instruction Decode (D1) stage :** In this stage, decoder in each pipeline checks if the current pair of instructions can execute together. If the instruction contains a prefix byte, an additional clock cycle is required in this stage. Also, such an instruction may only execute in the U pipeline and may not be paired with any other instruction.
3. **Address Generate (D2) stage :** In this stage, the addresses for the operands that reside in memory are calculated.
4. **Execute (EX) stage :** In this stage, operands are read from the data cache or memory and ALU operations are performed. Also, branch predictions for instructions (except conditional branches in the V pipeline) are verified in this stage.
5. **Writeback (WB) stage :** This is the final stage. In this stage, the results of the completed instructions are written and the conditional branch instruction predictions are verified.

When both the instructions from pipelines U and V reach the EX state, this may happen that one of them will stall and require additional clock cycles for the execution. No work is done during the stall. So the pipeline stall lowers performance. There are various situations when the instructions stall. For example, when the operands required for the operation are not found in the data cache. If the instruction in the U pipeline stalls, the instruction in the V pipeline also stalls. But if the instruction in the V pipeline stalls, the instruction in the U pipeline may continue executing. The instructions in the both pipelines must reach to the last stage, WB before another pair of instructions or the next single instruction may enter the EX stage.

13.7.10 Instruction Pairing Rules

The Pentium processor can issue one or two instructions every clock. In order to issue two instructions simultaneously they must satisfy the following conditions :

- **Both instructions in the pair must be "simple" instructions.**

  Simple instructions are entirely hardwired; they do not require any microcode control and, in general, execute in one clock. Examples of simple instructions are register-to-register MOVs, INC, DEC and near conditional jumps (JZ, JNZ, etc.). There is one more restriction to conditional jump instruction; it must be the second instruction in the pair. The arithmetic and logical instructions are also simple instructions; however, they may take two or three clock cycles because these instructions use both register and memory operands. Sequencing hardware is used to allow them to function as simple instructions. The following integer instructions are considered simple and may be paired :

1. MOV reg, reg/mem/imm
2. MOV mem, reg/imm
3. ALU reg, reg/mem/imm
4. ALU mem, reg/imm
5. INC reg/mem
6. $\overline{\text{DEC}}$ reg/mem
7. PUSH reg/mem
8. POP reg
9. LEA reg, mem
10. JMP/CALL/JCC near
11. NOP

- **Shifts or rotates can only pair in the U pipe.**
  (SHL, SHR, SAL, SAR, ROL, ROR, RCL or RCR)
- **ADC and SBB can only pair in the U pipe.**
- **JMP, CALL and Jcc can only pair in the V pipe. (Jcc = jump on condition code).**
- **Neither instruction can contain BOTH a displacement and an immediate operand.**

  For example :
```
mov  [bx+2], 3 ;  2 is a displacement, 3 is immediate
mov  mem1, 4   ;  mem1 is a displacement, 4 is immediate
```
- **Prefixed instructions can only pair in the U pipe.** Prefixed instructions (such as MOV, AL, ES : [DI]) may only execute in the U pipeline. Therefore, only one prefixed instruction in the pair is allowed.
- **The U pipe instruction must be only 1 byte in length or it will not pair until the second time it executes from the cache.**
- **There should not be any data dependencies between them.**

  Data dependency : The data dependency between two instructions exists if :

  The result of the first instruction is an operand for the second instruction (read-after-write dependency). That is we can not read the operand from register for the second instruction until first instruction writes its result in the register.

There can be no read-after-write or write-after write register dependencies between the instructions except for special cases for the flags register and the stack pointer

```
mov   ebx,2      ; writes to EBX
add   ecx, ebx   ; reads EBX and ECX, writes to ECX
                 ; EBX is read after being written, no pairing
mov   ebx, 1     ; writes to EBX
mov   ebx, 2     ; writes to EBX
                 ; write after write, no pairing
```

The flags register exception allows an ALU instruction to be paired with a Jcc even though the ALU instruction writes the flags and Jcc reads the flags.

For example :

```
cmp   al, 0      ; CMP modifies the flags
je    addr       ; JE reads the flags, but pairs
dec   cx         ; DEC modifies the flags
jnz   loopl      ; JNZ reads the flags, but pairs
```

The stack pointer exception allows two PUSHes or two POPs to be paired even though they both read and write to the SP (or ESP) register.

```
push eax    ;   ESP is read and modified
push ebx    ;   ESP is read and modified, but still pairs
```

13.7.11 Branch Prediction

We have seen that pipelined instruction execution is a valid technique for improving instruction execution rate and hence the performance of a processor. But it reduces when there is a presence of program transfer instructions such as JMP, CALL, RET, the conditional branch instructions etc. in the instruction stream. When the pipelined instruction execution technique is used, the instruction pipeline is always filled with a group of instructions stored in sequential memory locations. But when program transfer instruction is present, it changes the normal sequence of execution. So all the instructions that entered the pipeline after this instructions become incorrect. In this case, the instructions which come in the sequence because of the execution of branch instruction should be loaded in the pipeline. The incorrect instructions that loaded wrongly, must be discarded. This is called **'flushing'** of the pipeline. After flushing, a new sequence of instructions which is correct because of execution of branch instruction, is loaded in the pipeline. No work is done when the pipeline stages are reloaded. These disturbances in the pipelined instruction execution are called **'bubbles'**.

Pentium overcomes this problem by using a technique called **'dynamic branch prediction'**. The branch is to be taken or not taken, is decided by prediction. If the prediction is true, the pipeline will not be flushed, no cycles will be lost and no bubbles in the pipeline. If the prediction is false, the pipeline is flushed. So the cycles will be wasted and this causes bubbles in the pipeline. The pipeline is loaded with the correct group of instructions. Naturally, it is best if the predictions are true most of the time. Pentium uses a branch target buffer (BTB) for dynamic branch prediction. BTB is a special cache which stores the branch instruction that occurs in the instruction stream and target addresses of it. BTB also stores two history bits which indicate the execution history of the last two branch instructions. BTB uses the history bits to predict whether the branch is taken or not taken. When a new target address is placed into the BTB, these history bits are set to 11. When the corresponding branch instruction is present, the history bits are updated. The history bits become 00 if there are repeated failures to take a branch. Here, the prediction becomes 'not taken'. Fig. 13.65 shows the operations take place in dynamic branch prediction technique.

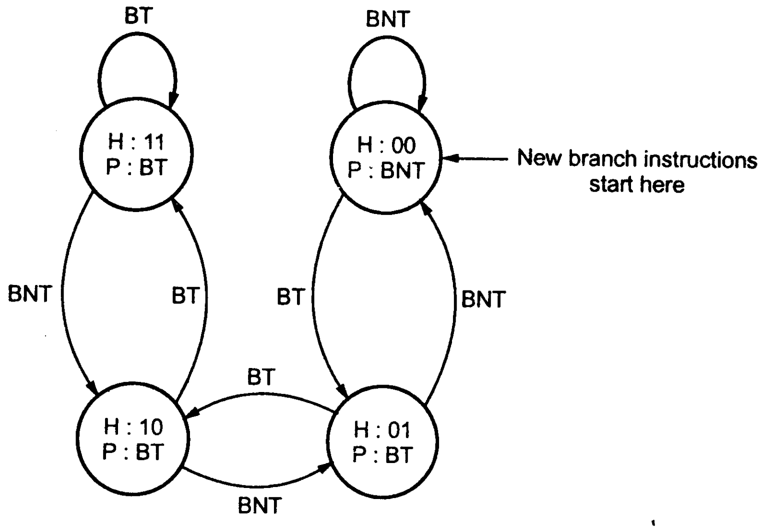


Fig. 13.65 Operations in dynamic branch prediction technique

Note :
1. Each state is represented by history bits, H and prection, P.
2. Prediction is either 'branch is taken' indicated by BT or 'branch is not taken' indicated by BNT.

The prediction will be taken until the history bits become both zero. The BTB is accessed during the D1 stage of U and V pipelines. For a new branch instruction, there is no target address in the BTB and the prediction is not taken. There are two 32-byte buffers. One buffer prefetches instructions from the current program address and the other buffer prefetches instructions from the target address when the BTB's prediction is 'branch taken (BT)'. If the predictions are correct, clock cycles are not wasted. If the predictions are incorrect or predictions are correct, but the target address is wrong, the pipelines will be flushed. This looses three clock cycles in the U pipeline and four clock cycles in the V pipeline.

Most of the times, we use conditional jumps to form loops in programs. The prediction, 'branch is taken (BT)' forms the required multiple passes through a loop. In Pentium, the history bits are set to 11 for a new entry. So, using the dynamic branch prediction, the wastage of clock cycles is minimised.

13.7.12 The Instruction and Data Caches

In this section, we will see the concept of cache memory, advantages of using caches and Pentium cache organization.

13.7.12.1 Cache Memory

In a computer system the program which is to be executed is loaded in the main memory (DRAM). Processor then fetches the code and data from the main memory to execute the program. The DRAMs which form the main memory are slower devices. So it is necessary to insert wait states in memory read/write cycles. This reduces the speed of execution. To speed up the process, high speed memories such as SRAMs must be used. But considering the cost and space required for SRAMs, it is not desirable to use SRAMs to form the main memory. The solution for this problem is come out with the fact that most of the microcomputer programs work with only small sections of code and data at a particular time. In the memory system small section of SRAM is added along with main memory, referred to as **cache memory**. The program which is to be executed is loaded in the main memory, but the part of program (code) and data that work at a particular time is usually accessed from the cache memory. This is accomplished by loading the active part of code and data from main memory to cache memory. The cache controller looks after this swapping between main memory and cache memory with the help of DMA controller.

When processor finds the addressed code or data in the cache, it is called **'cache hit'**. The percentage of accesses where the processor finds the code or data word it needs in the cache memory is called the **'hit rate'**. It is given by,

$$\text{Hit rate} = \frac{\text{Number of hits}}{\text{Number of read / write bus cycles}} \times 100\%$$

The hit rate is normally greater than 90 percent. When the required code or data is not found in the cache, it is called **'cache miss'**.

Thus, cache is a special type of high-speed RAM and is used to speed up accesses to memory and reduce traffic on the processor's buses. The advanced processors use the on-chip cache to achieve high speed accesses to memory and hence the performance.

➡ **Example 13.1 :** *The application program in a computer system with cache uses 1400 instruction acquisition bus cycle from cache memory and 100 from main memory. What is the hit rate? If the cache memory operates with zero wait state and the main memory bus cycels use three wait states, what is the average number of wait states experienced during the program execution?*

Solution :

$$\text{Hit rate} = \frac{1400}{1400+100} \times 100 = 93.3333\ \%$$

$$\text{Total wait states} = 1400 \times 0 + 100 \times 3 = 300$$

$$\text{Average wait states} = \frac{\text{Total wait states}}{\text{Number of memory bus cycles}} = \frac{300}{1500} = 0.2$$

13.7.12.2 Two Level Cache System

We know that, the on-chip cache is a high-speedcache. But its size is limited by space constraints. Therefore to design a high performance system secondary cache is used along with the primary on-chip cache, called **external cache**. Such system is called two level cache system and in such systems secondary cache is constructed with SRAM chips. Fig. 13.66 shows a two level cache system in a microcomputer.

As shown in Fig. 13.66, an on-chip cache supplies instructions and data to the CPU's pipeline. When a code or data is required from memory, the processor first searches it in the on-chip cache (internal cache). If it is found in the internal cache (a cache hit), a copy of it is sent to the pipeline very fastly. Usually, it takes just a clock cycle. If it is not found in the internal cache (a cache miss), the processor examines an external cache (a second level cache). If a cache miss occurs at an external cache or there is no external cache, the processor accesses the main memory. The processor writes the copy of code or data to the cache from main memory.

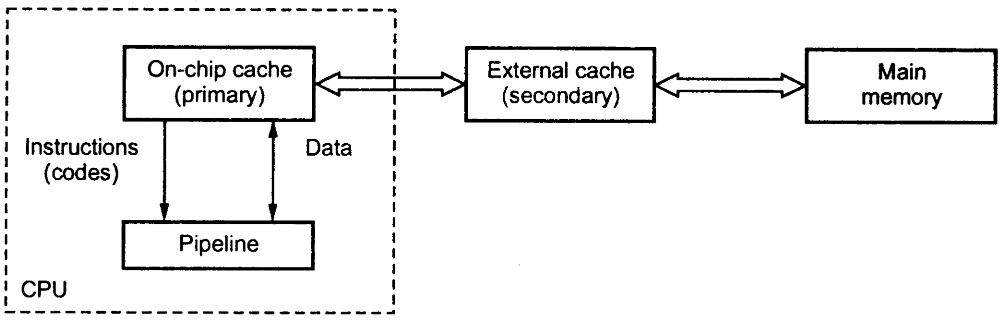


Fig. 13.66 Two-level cache system in a microcomputer

The secondary cache is much slower than the primary cache. But the size of secondary cache is large which ensures a high hit rate. The secondary cache thus reduces the impact of the main memory speed on the performance of a computer. The average access time experienced by the CPU in a two level cache system is

$$t_{av} = h_1 t_{a1} + (1-h_1)h_2 t_{a2} + (1-h_1)(1-h_2)\, t_a$$

Where t_{a1} is the access time and h_1 is the hit rate of L_1

t_{a2} is the access time and h_2 is the high rate of L_2

t_a is access time of main memory.

The number of hits in the secondary cache is given by the term $(1-h_1)\,h_2$ and the number of misses in the secondary cache is given by the term $(1-h_1)\ (1-h_2)$.

A 'hit-ratio' specifies the percentage of hits to total cache accesses. If the hit ratio is 0.9, then it means that the cache contains the requested information nine times out of ten. Thus, the average access time depends on the hit ratio. The average access time is given by,

$$T_{avg} = \text{hit-ratio} * T_{cache} + (1 - \text{hit-ratio}) * \left(T_{cache} + T_{RAM}\right)$$

13.7.12.3 Pentium Cache Organisation

Pentium processor provides separate caches for data and code. Both caches are organized as two-way set-associative caches with 128 sets. This gives 256 entries per cache. There are 32 bytes in a line (64 bytes per set), resulting in 8 kB of storage per cache. The data and instruction caches may be accessed simultaneously.

The Fig. 13.67 shows the internal structure of instruction and data cache. As mentioned earlier, it conists of 128 sets of two lines each. Each line is associated with a tag. The tags are triple ported, meaning that they can be accessed from three different places at the same time. Two of these ports are the U and V pipelines, which access the data cache to read/write instruction operands. The third port is used for a special operation called **bus snooping.** The code cache tags are also triple ported to support snooping and split line accesses. [The snooping is used to maintain consistent data in a multiprocessor system where each processor has a separate cache].

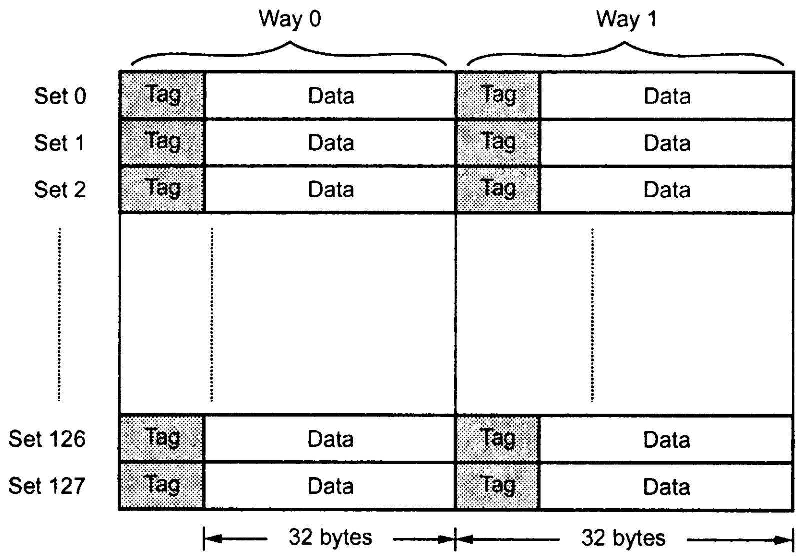


Fig. 13.67 Structure of 8 kB instruction and data cache

Each entry in the data cache can be configured for write back or write through. The code cache is an inherently write-protected cache. It is write protected to prevent self-modifying code from changing the execution program.

Each cache uses parity bits to maintain data integrity. Each tag is provided with one parity bit. There is one parity bit for every eight bytes of data (a quarter of a line or entry) in the instruction cache.

In pentium, individual pages can be configured as cacheable or non-cacheable by software or hardware. The caches can be enabled or disabled by software or hardware.

Translation lookaside buffers

Each cache has a dedicated Translation Lookaside Buffers (TLBs) to translate linear addresses into physical addresses. Physical addresses are used to access the cache because the same address is used to access main memory. The TLBs are also caches. The Table 13.15 gives the information of TLBs in the data cache and instruction cache.

TLB in data cache	TLB in instruction cache
• 2 TLBs	• 1 TLB
• **First** : 4-way set associative with 64 entries. It translates addresses for 4 kB pages of main memory.	• 4-way set-associative with 32 entries. It translates both addresses for 4 kB pages and 4 MB pages of main memory.
• **Second** : 4-way set associative with 8 entries. It translates addresses for 4 MB pages of main memory.	• 4 MB pages are cached as block of 4 kB each.
• Both TLBs are parity protected and dual ported.	• TLB is parity protected.
The cache controller uses Least-Recently-Used (LRU) algorithm to replace entries in all three TLBs. For that Pentium provides 3-bit LRU counter for each set.	

Table 13.15 TLBs for data and instruction cache

The Fig. 13.68 shows the overall cache organisation for Pentium processor.

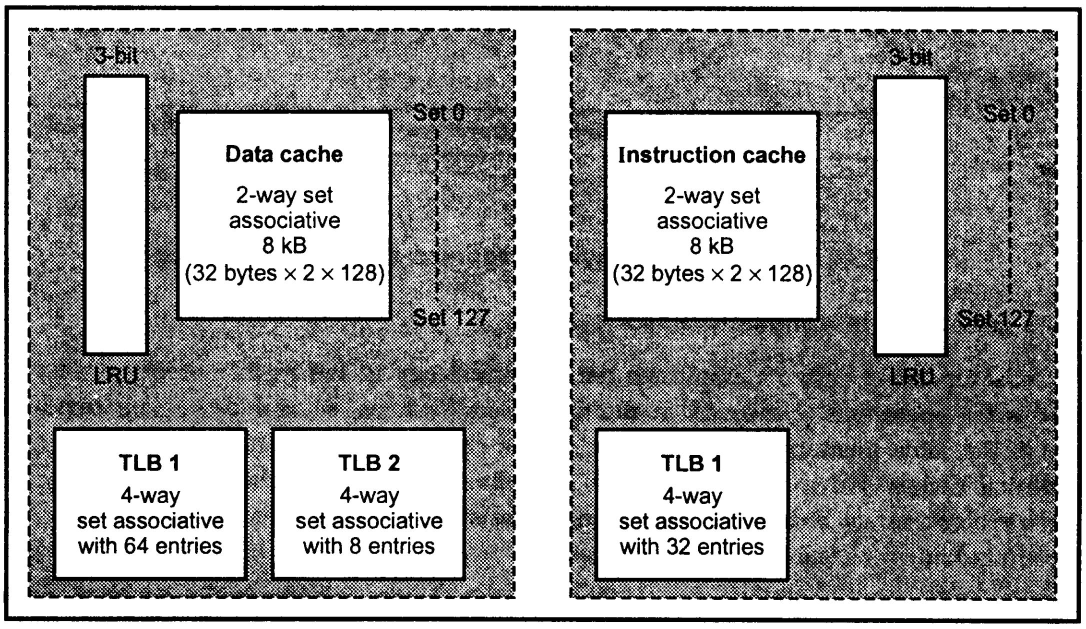


Fig. 13.68 Pentium cache organisation

Translating linear address into physical addresses with a TLB

The Fi 13.69 shows how TLB is used to translate linear address into physical address. The upper 20-bits of the linear address are checked against four tags and translated into the upper 20-bits physical address in case of cache hit. The lower 12-bits of the physical address are same as the lower 12-bits of linear address.

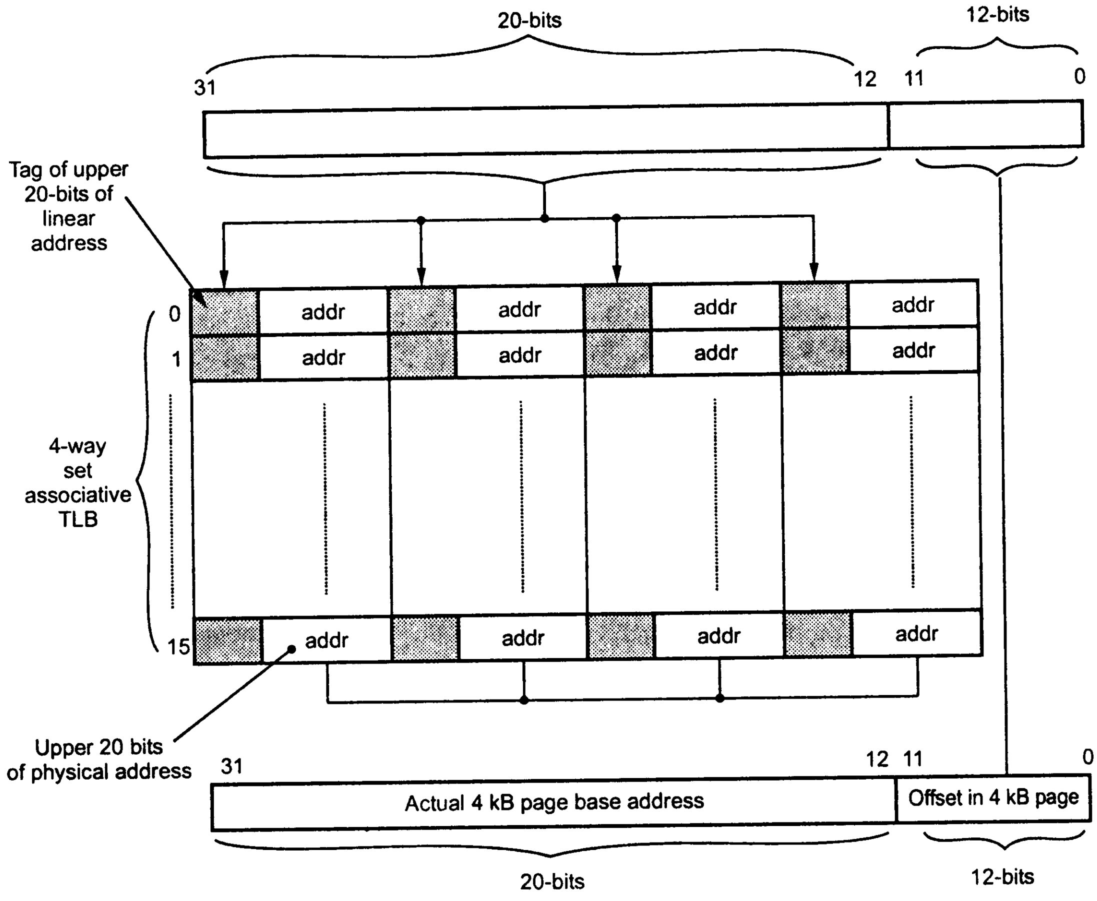

20-bits
12-bits
31
12
11
0
Tag of upper 20-bits of linear address
0
1
4-way set associative TLB
15
addr
Upper 20 bits of physical address
31
12
11
0
Actual 4 kB page base address
Offset in 4 kB page
20-bits
12-bits

Fig. 13.69 Generation of physical address from linear address

Cache coherency in a multiprocessor system

Cache updating systems eliminates data inconsistency in the main memory caused by cache write operations. However, in multiprocessor systems, several processors require a copy of the same memory block and they store a copy of the same memory block in their individual caches. Now, if the processors are allowed to update the data in the cached memory block in its individual cache, an inconsistent view of memory can result. This problem is known as 'cache coherence' problem.

To avoid such inconsistency and to maintain **cache coherency** in its data cache Pentium uses **MESI (Modified/Exclusive/Shared/Invalid) Protocol.** MESI protocol uses two bits to keep information of the state of each cache line. The state of each cache line is marked as modified, exclusive, shared or invalid. The meaning of each state in this protocol is as given below.

- **Modified** : The line in the cache, different from main memory is modified and this line is available only in this cache.
- **Exclusive** : The line in the cache is same as that in main memory and it is not present in any other cache.
- **Shared** : The line in the cache is same as that in main memory and the same line may be present in one or more other caches.
- **Invalid** : The line in the cache does not contain valid data.

13.7.13 Floating Point Unit

In 8086, 80286 and 80386, floating point operations were performed with the help of external coprocessors. Table 13.16 gives the list of coprocessors used with 80 × 86 family.

Processor (80 × 86 family)	Coprocessor (80 × 87 family)
8086/88	8087
80286	80287
80386	80387

Table 13.16 Processors and coprocessors

The 80 × 87 coprocessor, when used with 80 × 86 processor, shared address bus, data bus and control bus with the processor. A considerable time is required for the synchronization between the processor and the coprocessor to perform floating point operations. This problem was solved by placing coprocessor on the processor chip. This was done for 80486 and Pentium. Since the coprocessor is on the same chip as the processor, communication is faster and execution takes place quickly. Thus, there is an internal floating point unit (FPU) for 80486 and Pentium. The Pentium contains an improved, totally redesigned FPU over that used in the 80486.

The number of clock cycles required for many floating point instructions with 80 × 87 coprocessor units are reduced to few clock cycles in 80486 and Pentium. Also the new algorithms increase the speed of floating point operations. Consider an example of a floating-point multiply instruction, FMUL. Table 13.17 shows the number of clock cycles required for the execution of this instruction for different co-processors.

Coprocessor (80 × 87 family)	Minimum clock cycles required
8087	130
80287	130
80387	29
80486 FPU	16
Pentium FPU	1

Table 13.17 FMUL instruction performance

Thus, for many floating point instructions, there is an improvement in each generation, and highest improvement in the Pentium's FPU. Such a high speed of FPU is achieved using a pipeline. A FPU pipeline contains eight stages as shown in Fig. 13.70.

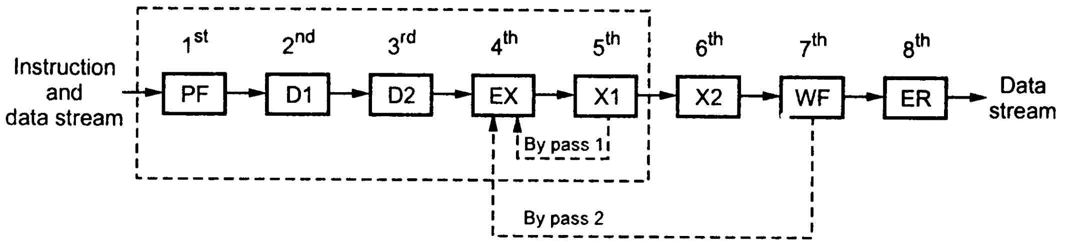


Fig. 13.70 Stages in FPU pipeline

As shown in Fig. 13.70, the first five stages are the ones that form the U pipeline, which processes integer instructions. Only difference is in the fifth stage. In U pipeline the fifth stage is WB (Writeback, as discussed earlier). In case of FPU pipeline, this fifth stage becomes the first stage for the floating point execution. The FPU pipeline consists of these five stages and extra three stages. Thus there are totally eight stages. All these stages and their functions are explained below.

i) PF : Prefetch

ii) D1 : Instruction decode

iii) D2 : Address generation

iv) EX : Memory and register read, floating-point data converted into memory format, memory write. The above stages are already explained in the section, pipelining.

v) X1 : Stage one in floating point execution. In this stage, memory data is converted into floating-point format, operand is written to floating point register file. Using bypass 1, data is sent back to EX stage. This allows a floating point register write operation in the X1 stage to bypass the floating point register file. The result is sent to the instruction performing a floating-point register read in the EX stage.

vi) X2 : Stage two in the floating-point execution.

vii) WF : Round floating-point result and write to floating-point register file. Bypass2 path is followed to send data back to EX stage. Using bypass 2, the result of an arithmetic instruction in stage WF is made available to the next instruction fetching operands in the EX stage.

viii) **ER** : Error reporting. The status word is updated.

There are eight 80-bit floating-point registers in the floating-point register file, ST(0) through ST(7). Two read and two write operations can be performed simultaneously since there are two ports in read section and two ports in write section. The data is written to the two write ports from the X1 and WF stages of the pipeline.

Pentium's FPU is designed such that fast floating-point execution can be achieved.

Review Questions

1. *List and explain the different functional units in 80386.*
2. *List the different registers in 80386.*
3. *Describe 80386 flag register with significance of each and every bit in detail. How does it differ from 8086 ?*
4. *Write machine status word (CR0) with significance of each and every bit in it.*
5. *Define the purpose of each debug register in 80386.*
6. *Define the purpose of each control register in 80386.*
7. *What is the function of test registers ?*
8. *Explain the data types supported by 80386.*
9. *Explain the addressing modes supported by 80386.*
10. *Explain the new instructions supported by 80386, with the help of example of each.*
11. *Write a short note on instruction enhancement in 80386.*
12. *Explain the significance of the following signals with relevance to 80386*
    a. $\overline{BE_0}$ *to* $\overline{BE_3}$
    b .BS_{16} *to* $\overline{NA}$ *with their interdependency*
    c. PREQ
    d. ERROR
    e. READY
13. *How will you generate A0 , A1 signals in 80386 ?*
14. *Explain with suitable diagrams the various bus cycles and control signals to perform one non pipelined write cycle, one non pipelined read cycle for 80386DX. What effects* $\overline{B16}$ *will have if asserted ?*
15. *What is dynamic bus sizing in 80386 ? What is the hardware support required ? Show the scheme to implement it.*
16. *Draw and explain the basic memory interface with 80386DX.*
17. *Draw the memory organisation in 80386DX.*
18. *Explain the terms "Aligned Transfer" and "Unaligned Transfer".*
19. *Draw and explain 16-bit and 32-bit bus SRAM interface with 80386DX.*

20. *Draw and explain 16-bit and 32-bit bus EPROM interface with 80386DX.*
21. *Draw and explain typical memory circuit for 80386DX.*
22. *What do you mean by interleaved memory ?*
23. *Explain the basic I/O systems for 80386 with the help of block diagram.*
24. *Explain I/O mapped I/O and memory mapped I/O techniques.*
25. *Explain typical I/O interface circuit for 80386.*
26. *List the features of 80486.*
27. *Draw and explain the architecture of a 80486.*
28. *Explain the advanced architectural features of 80486.*
29. *What is a significance of CD and NW bits in the control register 0.*
30. *Explain the bit pattern of EFLAGs register in 80486.*
31. *Explain the parity signals used in 80486.*
32. *Explain the bus control and bus arbitration logic in 80486.*
33. *Explain the following pins of 80486*
    a. $\overline{A20M}$ *b.* $\overline{IGNNE}$ *c.* $\overline{FERR}$ *d. PWT and PCB*
34. *Explain the new instruction supported by 80486.*
35. *Write a note on memory system of 80486.*
36. *Write a note on cache memory of 80486.*
37. *Explain the non-burst read cycle in 80486 with the help of timing diagram.*
38. *What is synchronous burst mode cache ?*
39. *Explain the burst cycle in 80486 with the help of timing diagram.*
40. *Write a note on memory managemen. .f 80486.*
41. *Explain important features of Pentium microprocessor.*
42. *Explain the significant additions and enhancements in the Pentium processor.*
43. *Draw and explain the block diagram of Pentium processor.*
44. *Explain any five Pentium processor signals.*
45. *Write a short notes on*
    a) Pentium RISC features. *b) Pentium super-scalar Architecture*
46. *What do you mean by simple instructions ?*
47. *What is data dependency ?*
48. *What is pipelining ?*
49. *Explain the pipelined instruction execution with the help of block diagram.*
50. *Explain the instruction pairing rules with the help of suitable examples.*
51. *What is branch prediction ?*
52. *Explain the dynamic branch prediction technique used in Pentium processors.*
53. *What is cache memory ?*
54. *Define hit rate.*
55. *Explain the two level cache system.*

56. *Draw and explain the internal structure of instruction and data cache of Pentium processor.*
57. *What is bus snooping ?*
58. *What is TLB ?*
59. *Write a short note on instruction and data cache in Pentium.*
60. *Draw and explain the Pentium cache organisation.*
61. *Explain, how linear address is converted into physical address using TLBs.*
62. *What is cache coherency ?*
63. *How Pentium maintains cache coherency ?*
64. *What is MESI protocol ?*
65. *Write a short note on floating point unit of Pentium processor.*

❑❑❑

A Instruction Formats for 8086

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move:	76543210	76543210	76543210	76543210
Register/Memory to/from Register	100010dw	mod reg r/m		
Immediate to Register/Memory	1100011w	mod 000 r/m	data	data if w = 1
Immediate to Register	1011 w reg	data	data if w = 1	
Memory to Accumulator	1010000w	addr-low	addr-high	
Accumulator to Memory	1010001w	addr-low	addr-high	
Register/Memory to Segment Register	10001110	mod 0 reg r/m		
Segment Register to Register/Memory	10001100	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	11111111	mod 110 r/m		
Register	01010 reg			
Segment Register	000 reg 110			
POP = Pop:				
Register/Memory	10001111	mod 000 r/m		
Register	01011 reg			
Segment Register	000 reg 111			
XCHG = Exchange:				
Register/Memory with Register	1000011w	mod reg r/m		
Register with Accumulator	10010 reg			
IN = Input from:				
Fixed Port	1110010w	port		
Variable Port	1110110w			
OUT = Output to:				
Fixed Port	1110011w	port		
Variable Port	1110111w			
XLAT = Translate Byte to AL	11010111			
LEA = Load EA to Register	10001101	mod reg r/m		
LDS = Load Pointer to DS	11000101	mod reg r/m		
LES = Load Pointer to ES	11000100	mod reg r/m		
LAHF = Load AH with Flags	10011111			
SAHF = Store AH into Flags	10011110			
PUSHF = Push Flags	10011100			
POPF = Pop Flags	10011101			

Mnemonic and Description	Instruction Code			
ARITHMETIC				
ADD = Add:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Reg./Memory with Register to Either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s: w = 01
Immediate to Accumulator	0 0 0 0 0 1 0 w	data	data if w = 1	
ADC = Add with Carry:				
Reg./Memory with Register to Either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s: w = 01
Immediate to Accumulator	0 0 0 1 0 1 0 w	data	data if w = 1	
INC = Increment:				
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1			
BAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1			
SUB = Subtract:				
Reg./Memory and Register to Either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w = 01
Immediate from Accumulator	0 0 1 0 1 1 0 w	data	data if w = 1	
SSB = Subtract with Borrow				
Reg./Memory and Register to Either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w = 01
Immediate from Accumulator	0 0 0 1 1 1 w	data	data if w = 1	
DEC = Decrement:				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG = Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
CMP = Compare:				
Register/Memory and Register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with Register/Memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w = 01
Immediate with Accumulator	0 0 1 1 1 1 0 w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	0 0 1 1 1 1 1 1			
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1			
MUL = Multiply (Unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL = Integer Multiply (Signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV = Divide (Unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV = Integer Divide (Signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0			
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1			

Mnemonic and Description	**Instruction Code**			
LOGIC	76543210	76543210	76543210	76543210
NOT = Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR = Shift Logical Right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR = Shift Arithmetic Right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL = Rotate Left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR = Rotate Right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL = Rotate Through Carry Flag Left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR = Rotate Through Carry Right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
AND = And:				
Reg./Memory with Register to Either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w =1
Immediate to Accumulator	0 0 1 0 0 1 0 w	data	data if w = 1	
TEST = And Function to Flags, No Result:				
Register/Memory and Register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate Data and Accumulator	1 0 1 0 1 0 0 w	data	data if w = 1	
OR = Or:				
Reg./Memory and Register to Either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1
Immediate to Accumulator	0 0 0 0 1 1 0 w	data	data if w = 1	
XOR = Exclusive or:				
Reg./Memory and Register to Either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 1 0 1 0 w	data	data if w = 1	
STRING MANIPULATION				
REP = Repeat	1 1 1 1 0 0 1 z			
MOVS = Move Byte/Word	1 0 1 0 0 1 0 w			
CMPS = Compare Byte/Word	1 0 1 0 0 1 1 w			
SCAS = Scan Byte/Word	1 0 1 0 1 1 1 w			
LODS = Load Byte/Wd to AL/AX	1 0 1 0 1 1 0 w			
STOS = Stor Byte/Wd from AL/A	1 0 1 0 1 0 1 w			
CONTROL TRANSFER **CALL = Call:**				
Direct within Segment	1 1 1 0 1 0 0 0	disp-low	disp-high	
Indirect within Segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m		
Direct Intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m		

Mnemonic and Description	Instruction Code		
JMP = Unconditional Jump	76543210	76543210	76543210
Direct within Segment	11101001	disp-low	disp-high
Direct within Segment-Short	11101011	disp	
Indirect within Segment	11111111	mod 1 0 0 r/m	
Direct Intersegment	11101010	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	11111111	mod 1 0 1 r/m	
RET = Return from CALL:			
Within Segment	11000011		
Within Seg Adding Immed to SP	11000010	data-low	data-high
Intersegment	11001011		
Intersegment Adding Immediate to SP	11001010	data-low	data-high
JE/JZ = Jump on Equal/Zero	01110100	disp	
JL/JNGE = Jump on Less/Not Greater or Equal	01111100	disp	
JLE/JNG = Jump on Less or Equal/ Not Greater	01111110	disp	
JB/JNAE = Jump on Below/Not Above or Equal	01110010	disp	
JBE/JNA = Jump on Below or Equal/ Not Above	01110110	disp	
JP/JPE = Jump on Parity/Parity Even	01111010	disp	
JO = Jump on Overflow	01110000	disp	
JS = Jump on Sign	01111000	disp	
JNE/JNZ = Jump on Not Equal/Not Zero	01110101	disp	
JNL/JGE = Jump on Not Less/Greater or Equal	01111101	disp	
JNLE/JG = Jump on Not Less or Equal/ Greater	01111111	disp	
JNB/JAE = Jump on Not Below/Above or Equal	01110011	disp	
JNBE/JA = Jump on Not Below or Equal/Above	01110111	disp	
JNP/JPO = Jump on Not Par/Par Odd	01111011	disp	
JNO = Jump on Not Overflow	01110001	disp	
JNS = Jump on Not Sign	01111001	disp	
LOOP = Loop CX Times	11100010	disp	
LOOPZ/LOOPE = Loop While Zero/ Equal	11100001	disp	
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	11100000	disp	
JCXZ = Jump on CX Zero	11100011	disp	
INT = Interrupt			
Type Specified	11001101	type	
Type 3	11001100		
INTO = Interrupt on Overflow	11001110		
IRET = Interrupt Return	11001111		

Mnemonic and Description	Instruction Code	
PROCESSOR CONTROL	76543210	76543210
CLC = Clear Carry	11111000	
CMC = Complement Carry	11110101	
STC = Set Carry	11111001	
CLD = Clear Direction	11111100	
STD = Set Direction	11111101	
CLI = Clear Interrupt	11111010	
STI = Set Interrupt	11111011	
HLT = Halt	11110100	
WAIT = Wait	10011011	
ESC = Escape (to External Device)	11011xxx	mod xxx r/m
LOCK = Bus Lock Prefix	11110000	

❑❑❑