Design with Microprocessors

Year III Computer Sci. English

1-st Semester

Lecture 14: 8086/8088-BASED MULTIPROCESSOR SYSTEMS

8086 in maximum mode (#MX=0)

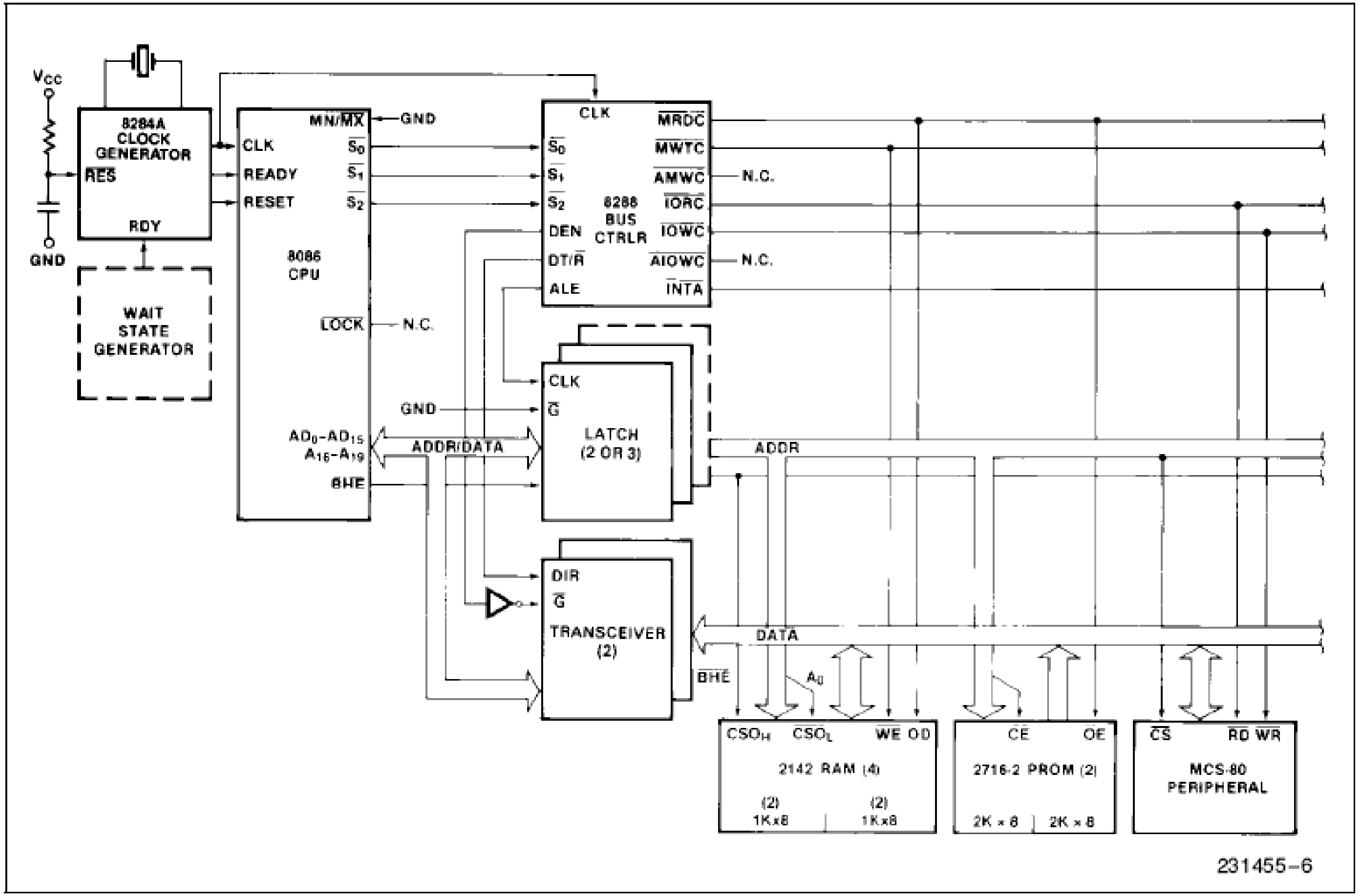
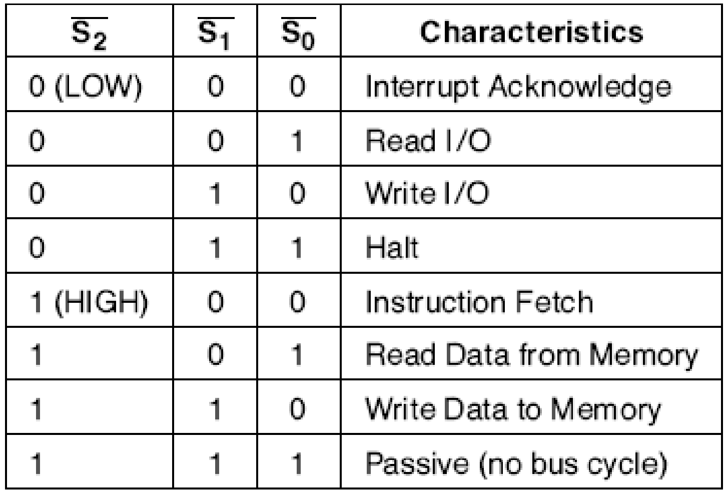


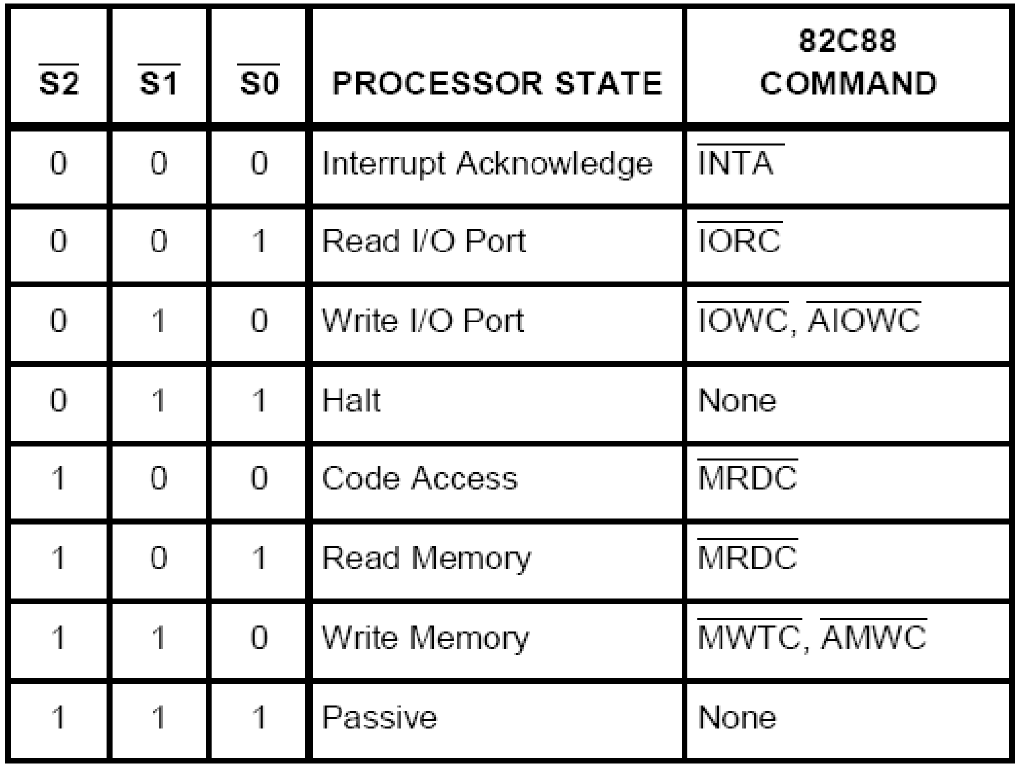
Figure 4b. Maximum Mode 8086 Typical Configuration

8288 Command Decode Definition



$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

TABLE 1. COMMAND DECODE DEFINITION



$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	PROCESSOR STATE	82C88 COMMAND
0	0	0	Interrupt Acknowledge	$\overline{\text{INTA}}$
0	0	1	Read I/O Port	$\overline{\text{IORC}}$
0	1	0	Write I/O Port	$\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$
0	1	1	Halt	None
1	0	0	Code Access	$\overline{\text{MRDC}}$
1	0	1	Read Memory	$\overline{\text{MRDC}}$
1	1	0	Write Memory	$\overline{\text{MWTC}}$, $\overline{\text{AMWC}}$
1	1	1	Passive	None

8086 in maximum mode – specific signals

QUEUE STATUS: The queue status is valid during the CLK cycle after which the queue operation is performed.
QS_1 and QS_0 provide status to allow external tracking of the internal 8086 instruction queue.

QS_1	QS_0	Characteristics
0 (LOW)	0	No Operation
0	1	First Byte of Op Code from Queue
1 (HIGH)	0	Empty the Queue
1	1	Subsequent Byte from Queue

$\overline{RQ}/\overline{GT}_0$, $\overline{RQ}/\overline{GT}_1$	30, 31	I/O	**REQUEST/GRANT:** pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\overline{GT}_1$. $\overline{RQ}/\overline{GT}$ pins have internal pull-up resistors and may be left unconnected. The request/grant sequence is as follows
$\overline{LOCK}$	29	O	**$\overline{LOCK}$:** output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF in "hold acknowledge".

LOCK FACILITY

Semaphores are used to ensure that at any given time only one process may enter its critical section of code in which a shared resource is accessed:

```
            MOV     AL,0
TRYAGAIN:   XCHG    SEMAPHORE,AL
            TEST    AL,AL
            JZ      TRYAGAIN
             .   }  Critical section in which a process
             .   }  accesses a shared resource
             .   }
            MOV     SEMAPHORE,1
```

1. Works fine for a system in which all of the processes are executed by the same processor, because the processor cannot switch from one process to another in the middle of an instruction.

2. Processor A and processor B are running independently and are concurrently accessing the same resource ⇒ they might test the semaphore at the same time.

XCHG instruction requires two bus cycles !!!

SEMAPHORE contains a 1 and both processors A and B are executing TRYAGAIN: XCHG SEMAPHORE,AL :

1. Processor A uses the first available bus cycle to get the contents of SEMAPHORE.
2. Processor B uses the next bus cycle to get the contents of SEMAPHORE.
3. Processor A clears SEMAPHORE during the next bus cycle, thus completing its XCHG instruction.
4. Processor B clears SEMAPHORE during the next bus cycle, thus completing its XCHG instruction.

⇒ AL registers in both processors will contain 1 and the TEST AL,AL instruction will cause the JZ instructions to fail ⇒ both processors will enter their critical sections of code.

LOCK FACILITY

⇒ the processor that starts executing its XCHG instruction first (which in this example is processor A) must have exclusive use of the bus until the XCHG instruction is completed ⇒ LOCK prefix:

1	1	1	1	0	0	0	0

⇒ maximum mode CPU, activates the \LOCK output pin during the execution of the instruction that follows the prefix.

⇒ \LOCK signal indicates to the bus control logic that no other processors may gain control of the system bus until the locked instruction is completed:

```
TRYAGAIN: LOCK XCHG SEMAPHORE,AL
```

⇒ would ensure that each XCHG will be completed in two *consecutive* bus cycles.

Physically, in a loosely coupled system each processing module includes a bus arbiter and the bus arbiters are connected together by special control lines in the system bus. One of these lines is a busy line which is active whenever the bus is in use.

If a \LOCK signal is sent to the arbiter controlling the bus, then that arbiter will retain control of the system by holding the busy line active until the \LOCK signal is dropped.Thus, if a processor applies a \LOCK signal throughout the execution of an entire instruction, its arbiter will not relinquish the system bus until the instruction is complete.

LOCK FACILITY

Another possible application of the bus lock capability:

⇒ fast execution of an instruction which requires several bus cycles.

For example, in a multiprocessor system a block of data can be transferred at a higher speed by using the LOCK prefix as follows:

```
LOCK REP MOVSB DEST , SRC
```

During the execution of this instruction the system bus will be reserved for the sole use of the processor executing the instruction.

REP

```
if CX <> 0 then
        do following chain instruction
        CX = CX - 1
        go back to check_cx
else
        exit from REP cycle
```

MOVSB

```
ES:[DI] = DS:[SI]

if DF = 0 then
        SI = SI + 1
        DI = DI + 1
else
        SI = SI - 1
        DI = DI - 1
```

8086/8088-BASED MULTIPROCESSING SYSTEMS

Ref: Microcomputer Systems: The 8086/8088 Family, Liu & Gibson, 1986

Multiprocessor Systems refer to the use of multiple processors that execute instructions simultaneously and communicate using mailboxes and semaphores

Maximum mode of 8086 is designed to implement **3 basic multiprocessor configurations**:

1. **coprocessor** (8087)
2. **closely coupled** (dedeicated I/O processor:8089)
3. **loosely coupled** (Multibus)

Coprocessors and closely coupled configurations are similar - both the CPU and the external processor share:

- **Memory**
- **I/O system**
- **Bus & bus control logic**
- **Clock generator**

Coprocessor Configuration

WAIT instruction ⇒ allows the processor to synchronize itself with external hardware, eg., waiting for 8087 math co-processor.

⇒ When the CPU executes **WAIT** ⇒ waiting state.

⇒TEST input is asserted (low), the waiting state is completed and execution will resume.

ESC instruction: **ESC opcode , operand**
opcode : immediate value recognizable to a coprocessor as an instruction opcode
operand: name of a register or a memory address (in any mode)
⇒ When the CPU executes the **ESC** instruction, the processor accesses the memory operand by placing the address on the address bus.
⇒ If a coprocessor is configured to share the system bus, it will recognize the ESC instruction and therefore will get the opcode and the operand.

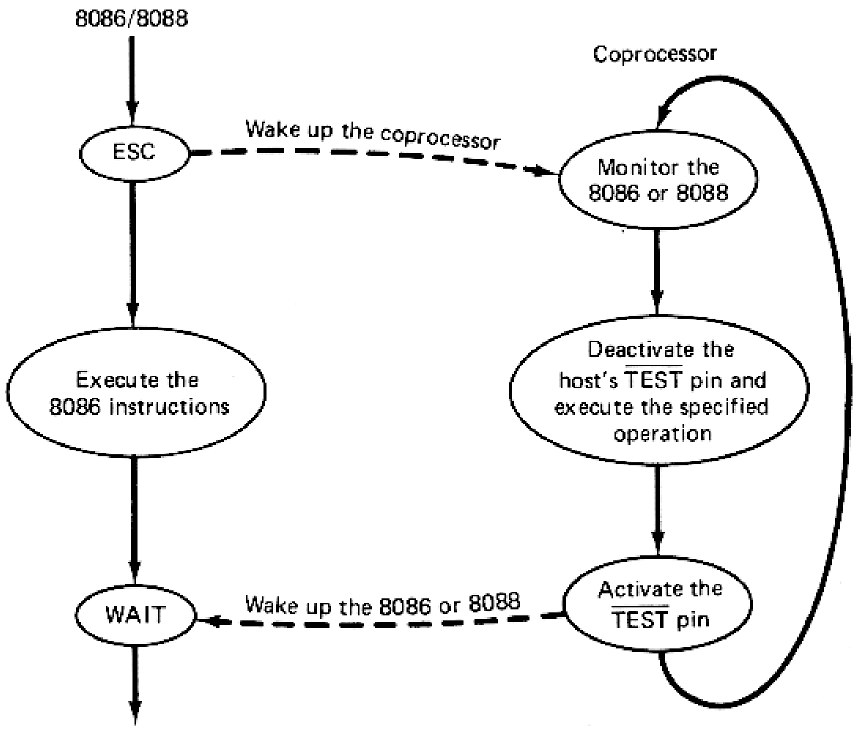

8086/8088
Coprocessor
ESC
Wake up the coprocessor
Monitor the 8086 or 8088
Execute the 8086 instructions
Deactivate the host's $\overline{\text{TEST}}$ pin and execute the specified operation
WAIT
Wake up the 8086 or 8088
Activate the $\overline{\text{TEST}}$ pin

Figure 11-3 Synchronization between the 8086 and its coprocessor.

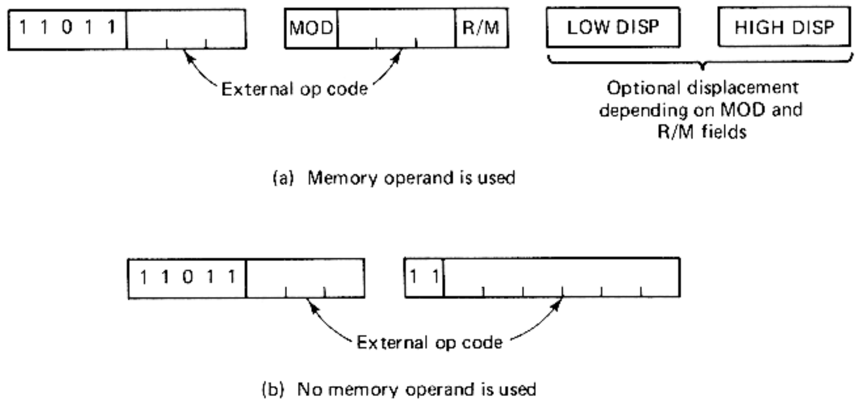

1 1 0 1 1
MOD
R/M
LOW DISP
HIGH DISP
External op code
Optional displacement depending on MOD and R/M fields
(a) Memory operand is used
1 1 0 1 1
1 1
External op code
(b) No memory operand is used

Figure 11-4 Machine code formats for the ESC instruction.

Coprocessor Configuration

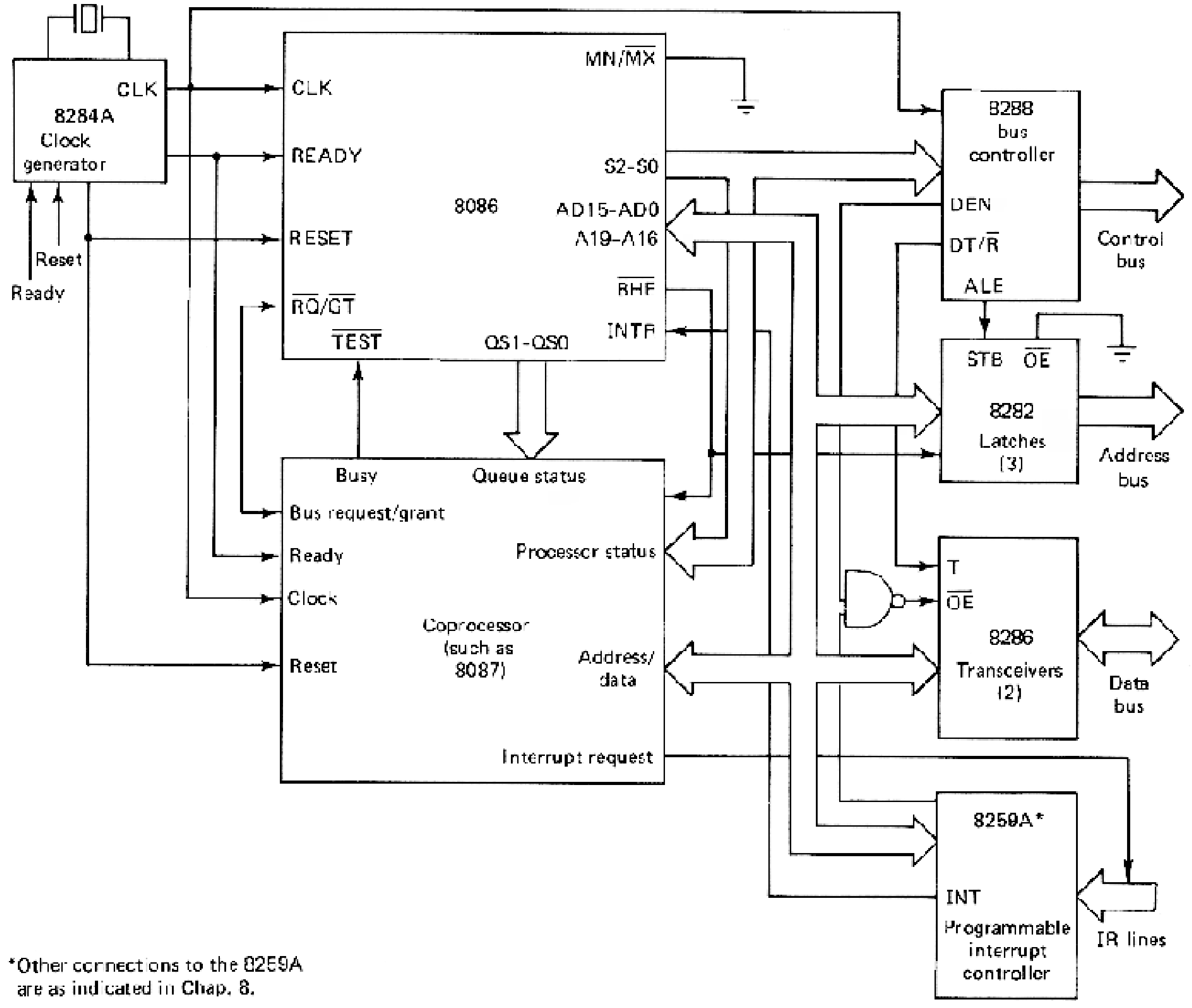
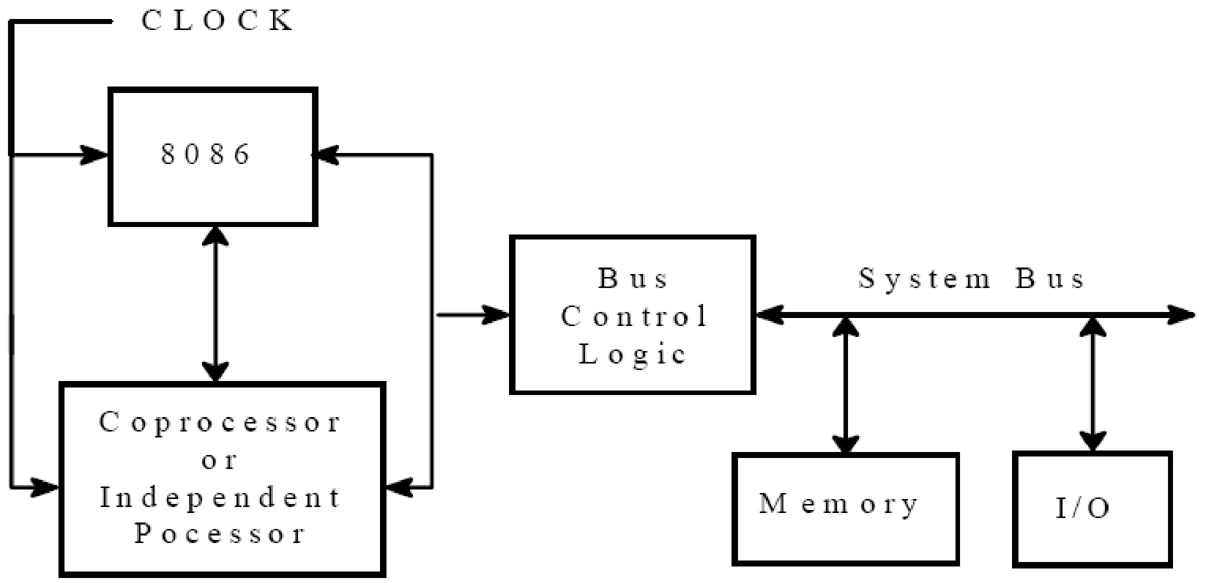


Figure 11-5 Coprocessor configuration.

Closely Coupled Configuration



⇒ Coprocessor cannot take control of the bus, it does everything through the CPU

⇒ Closely Coupled processor may take control of the bus independently

- 8089 shares CPU's clock and bus control logic
- communication with host CPU is by way of shared memory
- host sets up a message (command) in memory
- independent processor interrupts host on completion

Two 8086's cannot be closely coupled

Closely Coupled Configuration

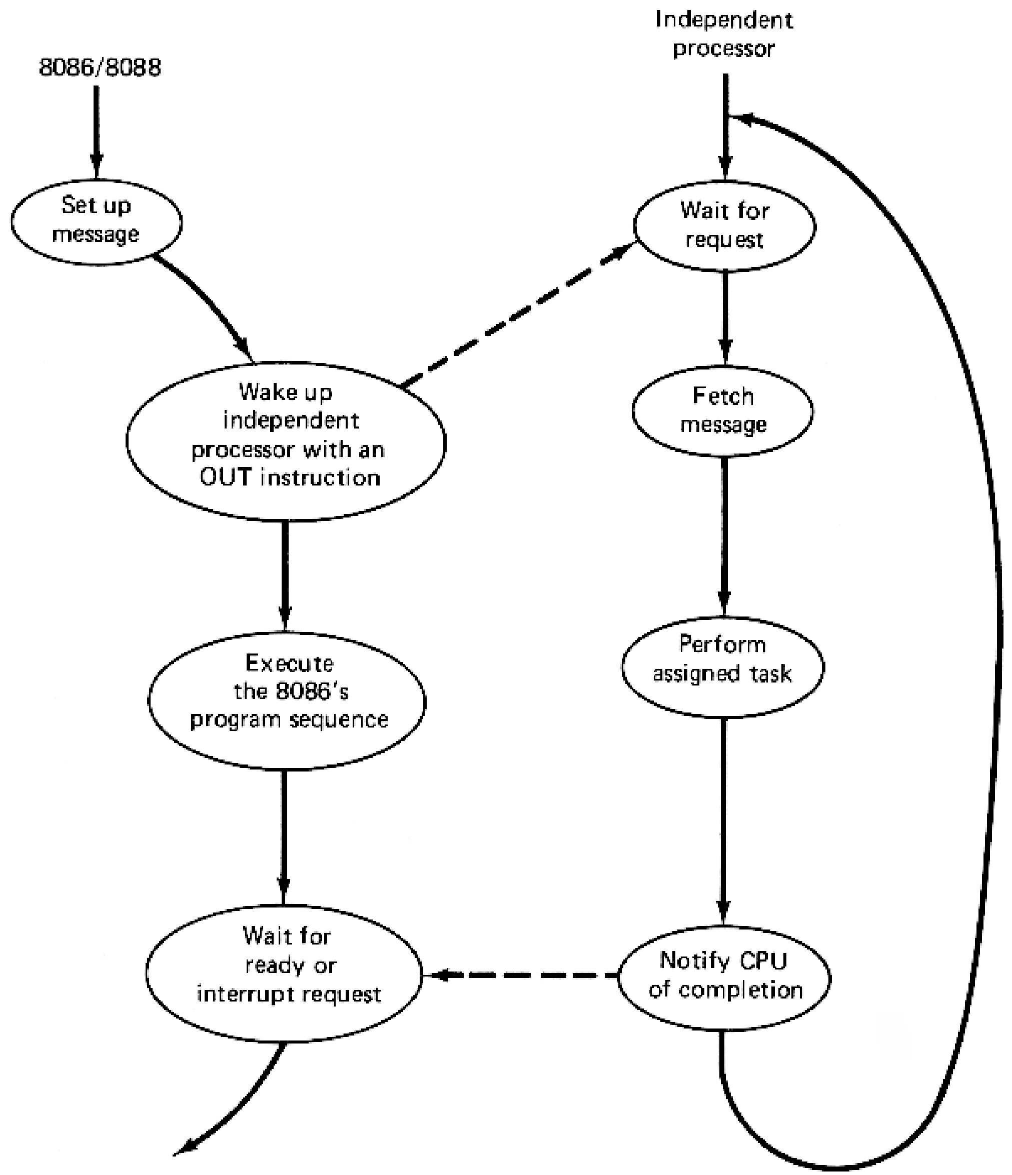

8086/8088
Independent processor
Set up message
Wake up independent processor with an OUT instruction
Execute the 8086's program sequence
Wait for ready or interrupt request
Wait for request
Fetch message
Perform assigned task
Notify CPU of completion

Figure 11-6 Interprocessor communication through shared memory.

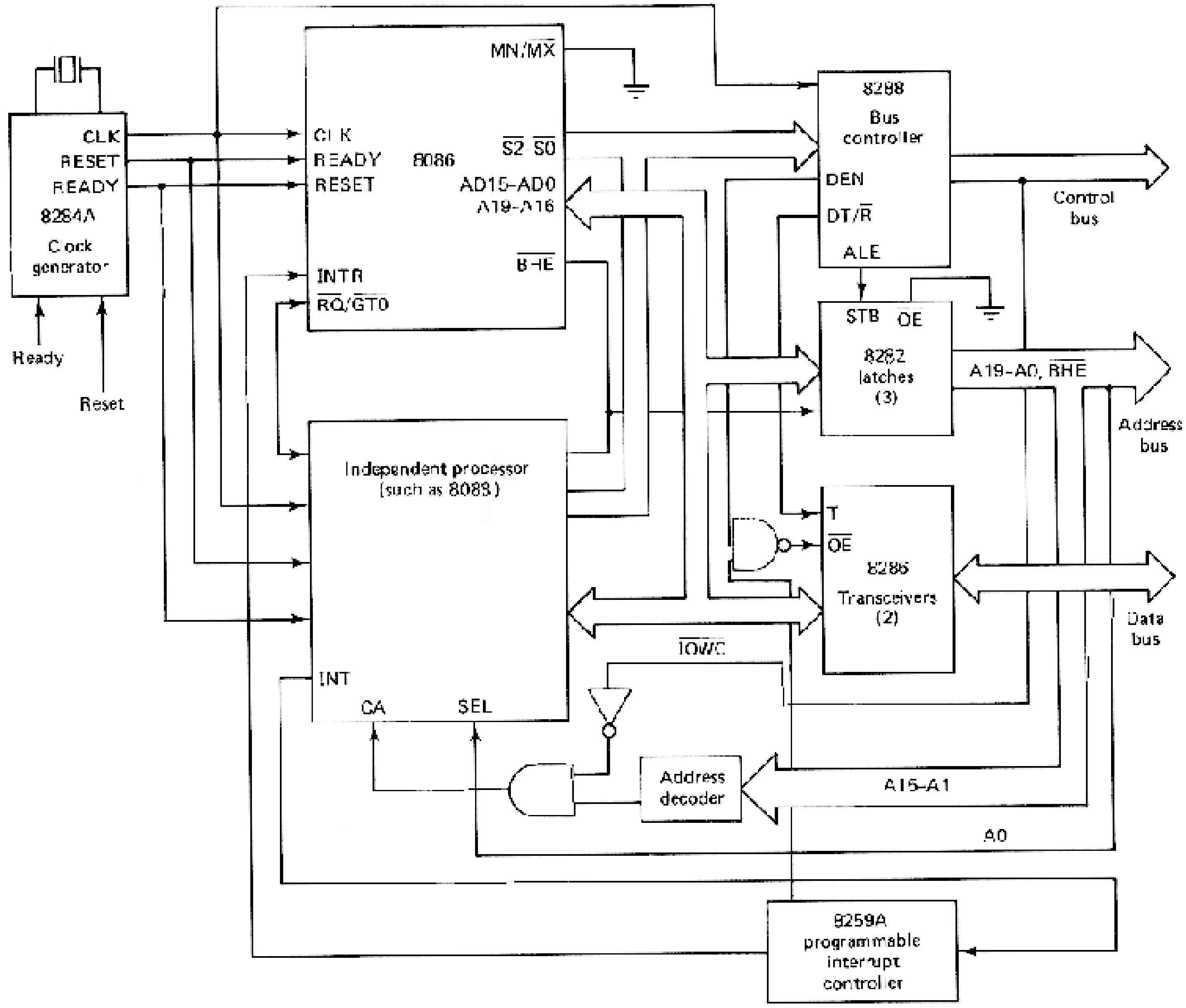
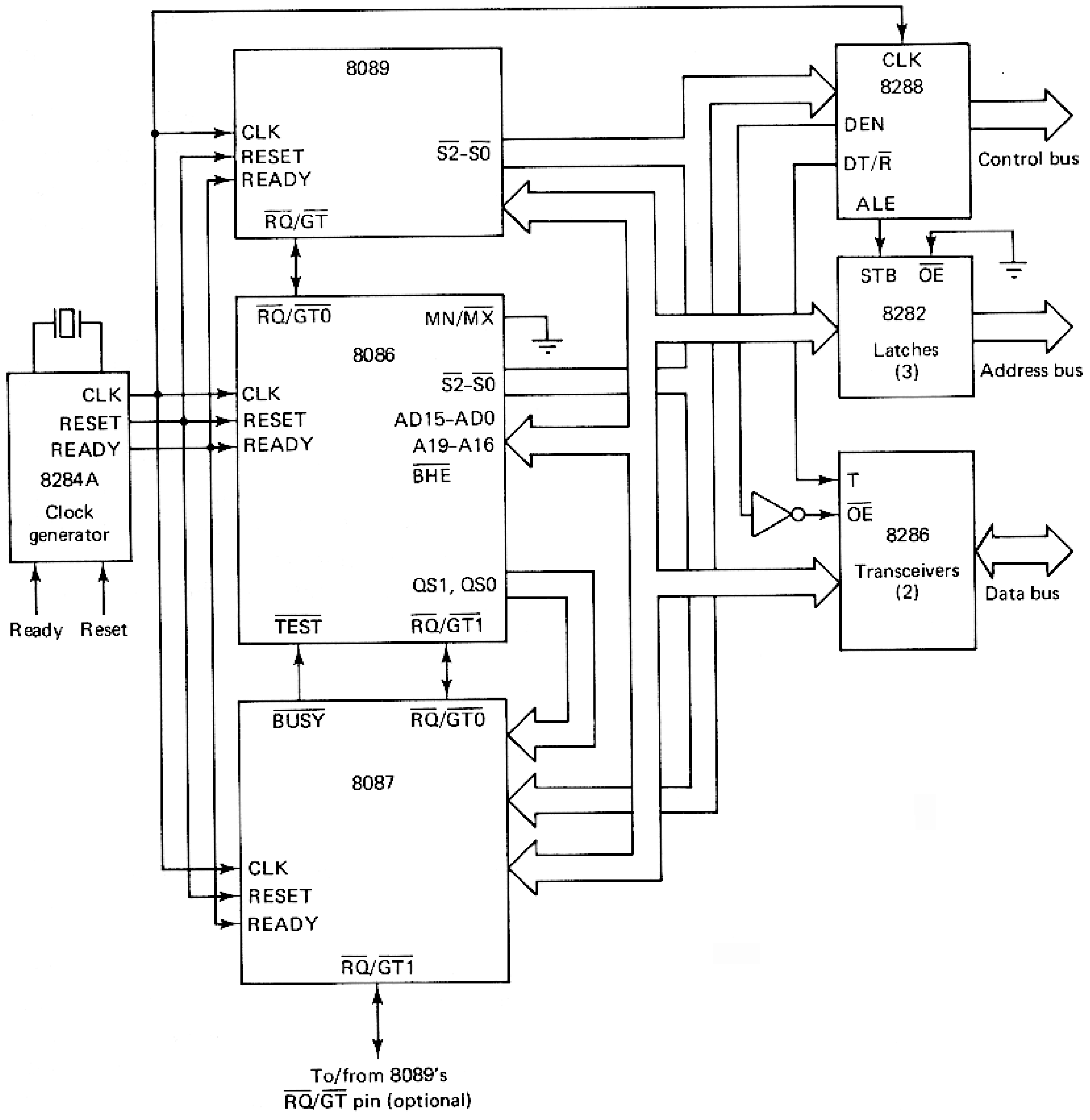


Figure 11-7 8086 connections in a closely coupled configuration.



NOTE: Interrupt system and other details are not shown.

Figure 11-8 Configuration involving both a coprocessor and an independent processor.

Loosely Coupled Configurations

A loosely coupled configuration provides the following advantages:

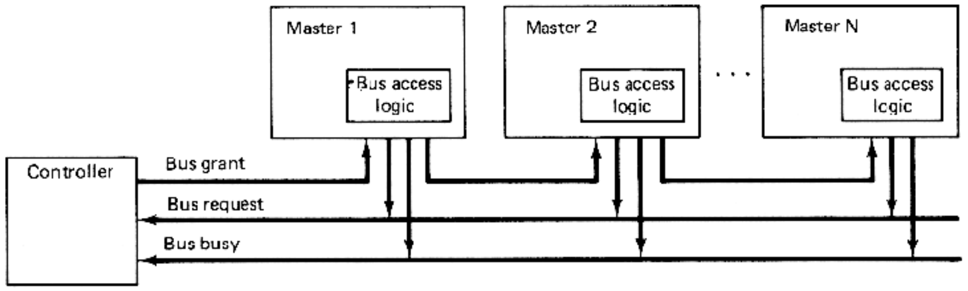
1. High system throughput can be achieved by having more than one CPU.
2. The system can be expanded in a modular form. Each bus master module is an independent unit and normally resides on a separate PC board. Therefore, a bus master module can be added or removed without affecting the other modules in the system.
3. A failure in one module normally does not cause a breakdown of the entire system and the faulty module can be easily detected and replaced.
4. Each bus master may have a local bus to access dedicated memory or I/O devices so that a greater degree of parallel processing can be achieved.

More than one bus master module may have access to the shared system bus ⇒ extra bus control logic must be provided to resolve the bus arbitration problem.

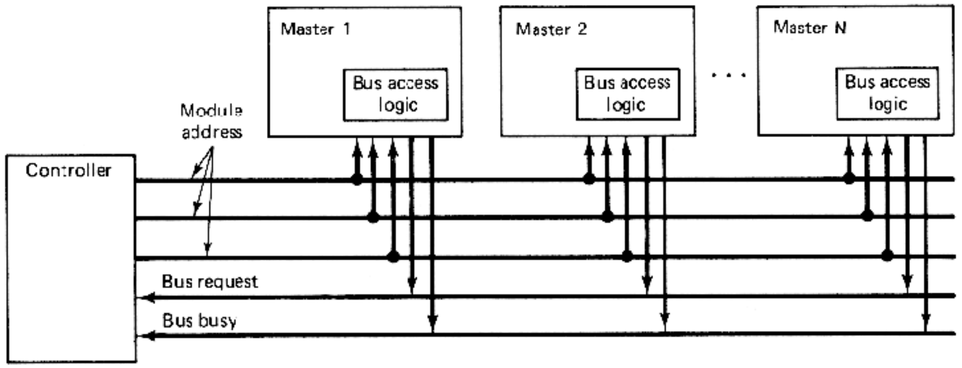
The extra logic is called *bus access logic* and it is its responsibility to make sure that only one bus master at a time has control of the bus.

Simultaneous bus requests are resolved on a priority basis. There are three schemes for establishing priority:

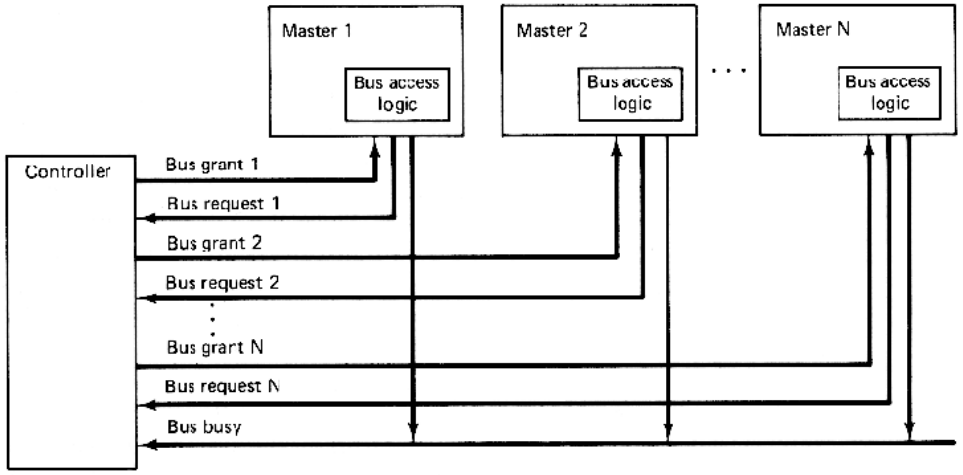
1.Daisy chaining.
2.Polling.
3.Independent requesting.



(a) Daisy chain method



(b) Polling method



(c) Independent requests method

Figure 11-9 Bus allocation schemes.

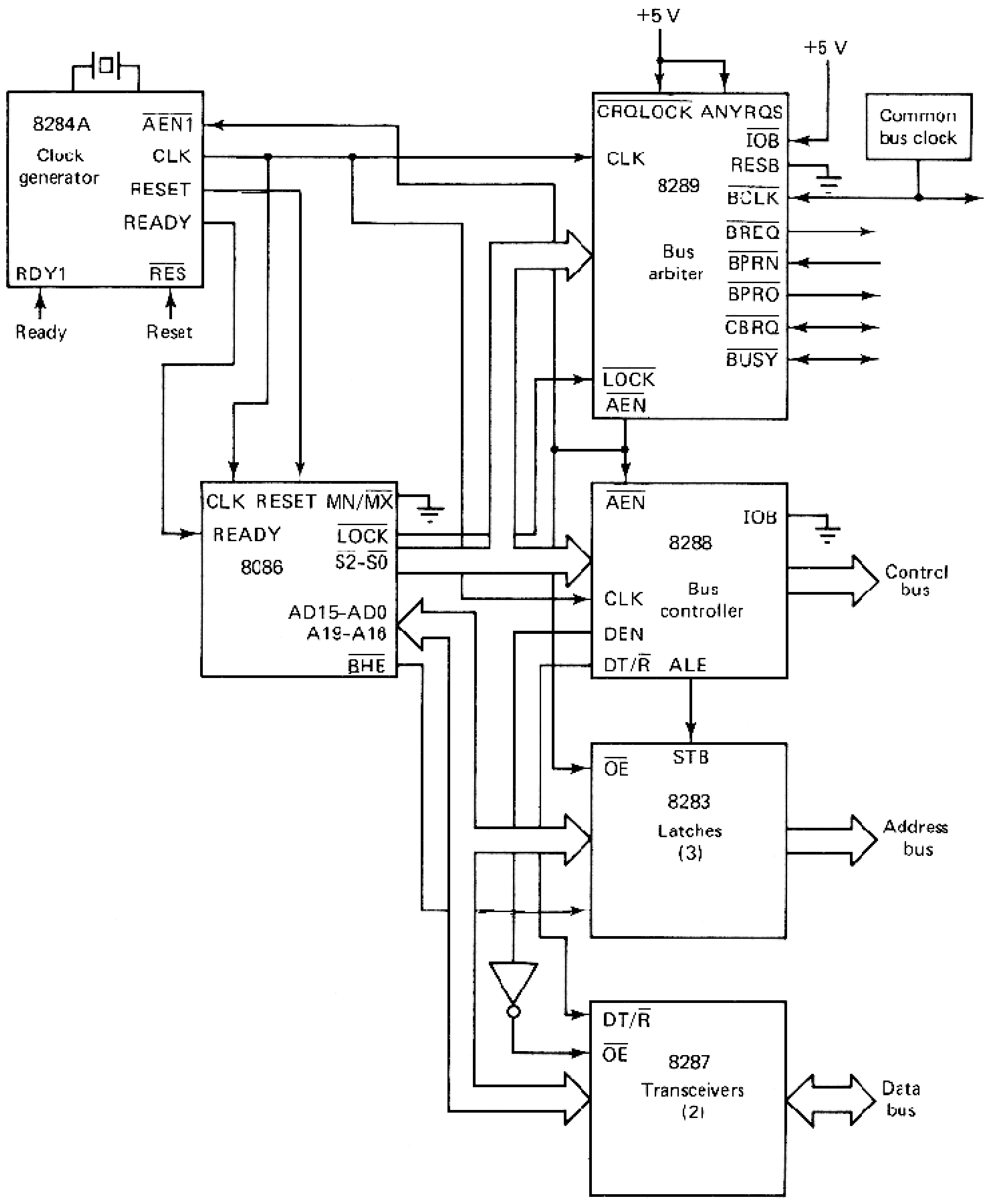


Figure 11-10 Single-processor module.

BPRN (BUS PRIORITY IN): returned to the arbiter to instruct it that it may acquire the multi-master system bus on the next falling edge of BCLK. BPRN active indicates to the arbiter that it is the highest priority requesting arbiter presently on the bus. The loss of BPRN instructs the arbiter that it has lost priority to a higher priority arbiter.

BPRO (BUS PRIORITY OUT): used in the serial priority resolving scheme where BPRO is daisy-chained to BPRN of the next lower priority arbiter.

CBRQ (COMMON BUS REQUEST): instructs the arbiter if there are any other arbiters of lower priority requesting the use of the multi-master system bus.

BCLK (BUS CLOCK): The multi-master system bus clock to which all multi-master system bus interface signals are synchronized.

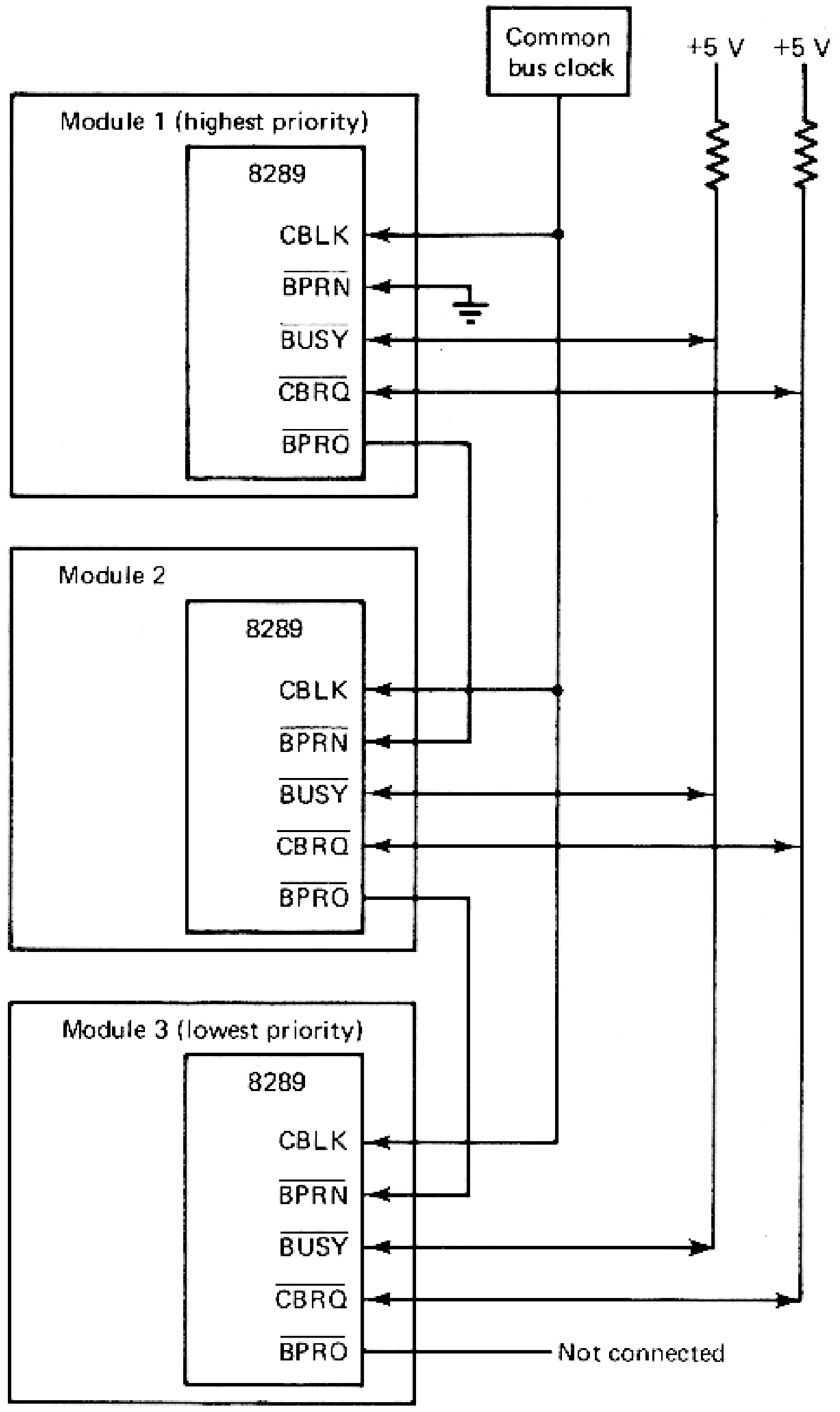


Figure 11-12 Daisy chain bus allocation using 8289s.

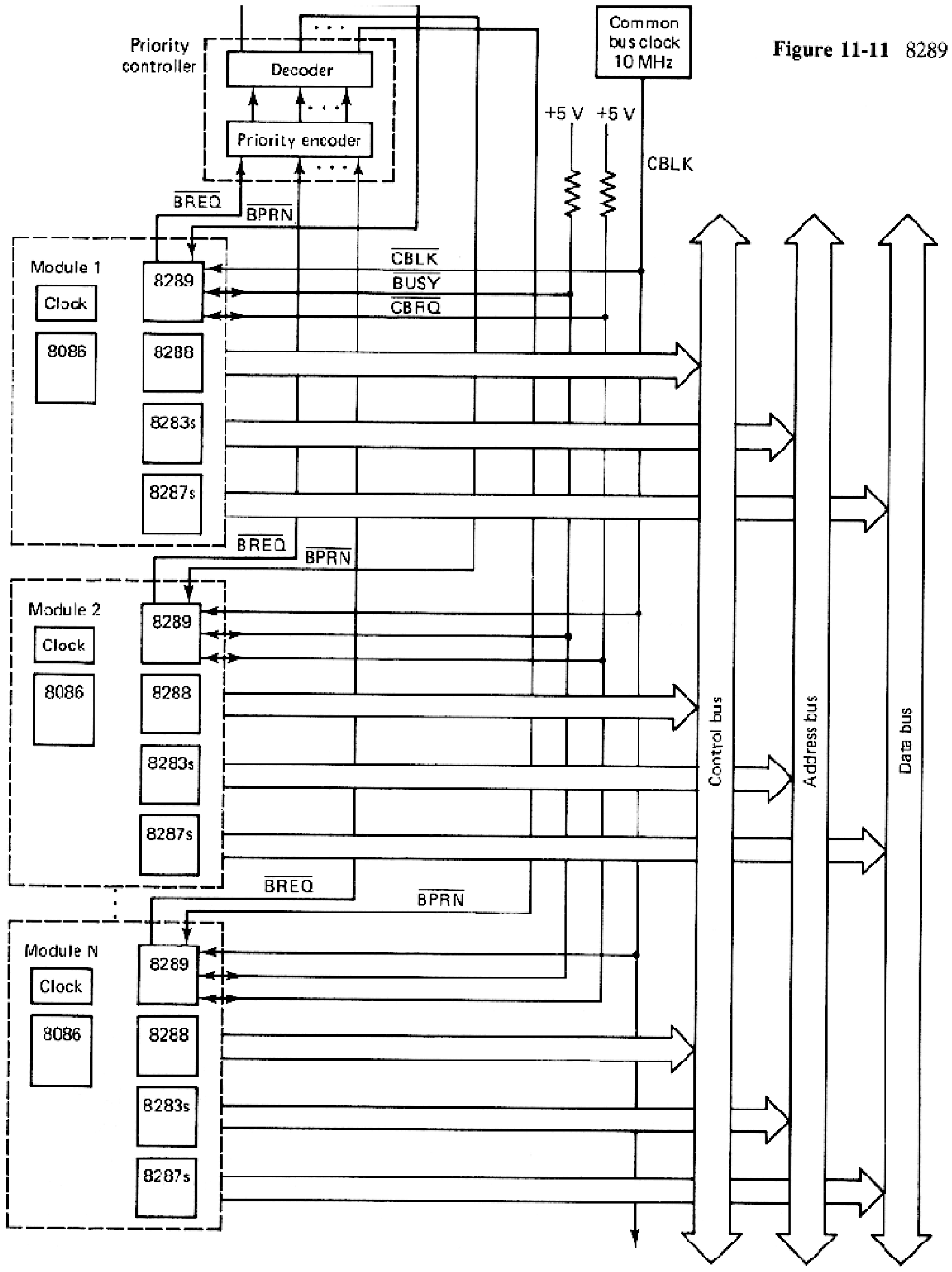


Figure 11-11 8289 connections to a priority controller.

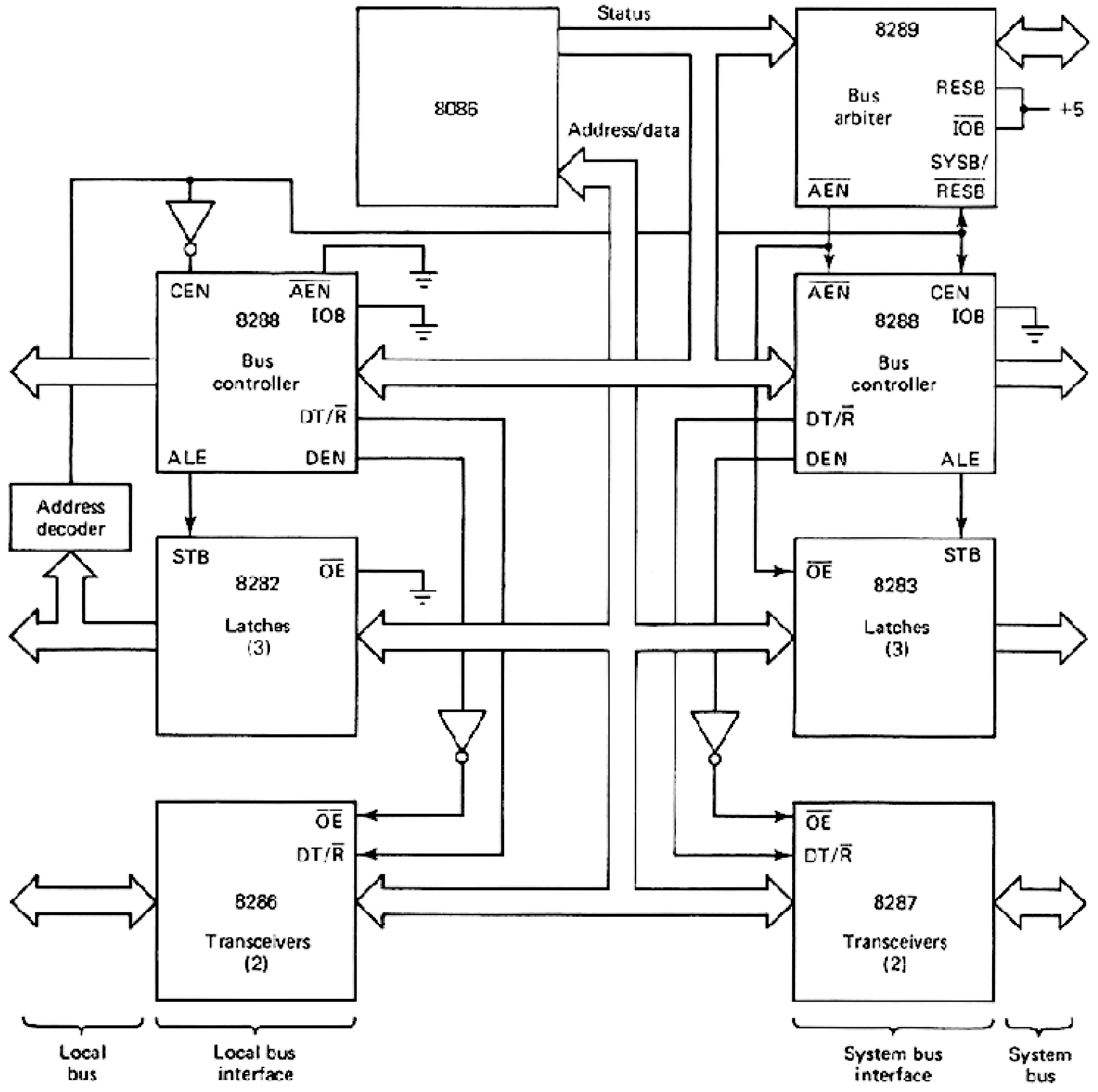


Figure 11-14 Configuration with both a local bus and a system bus.

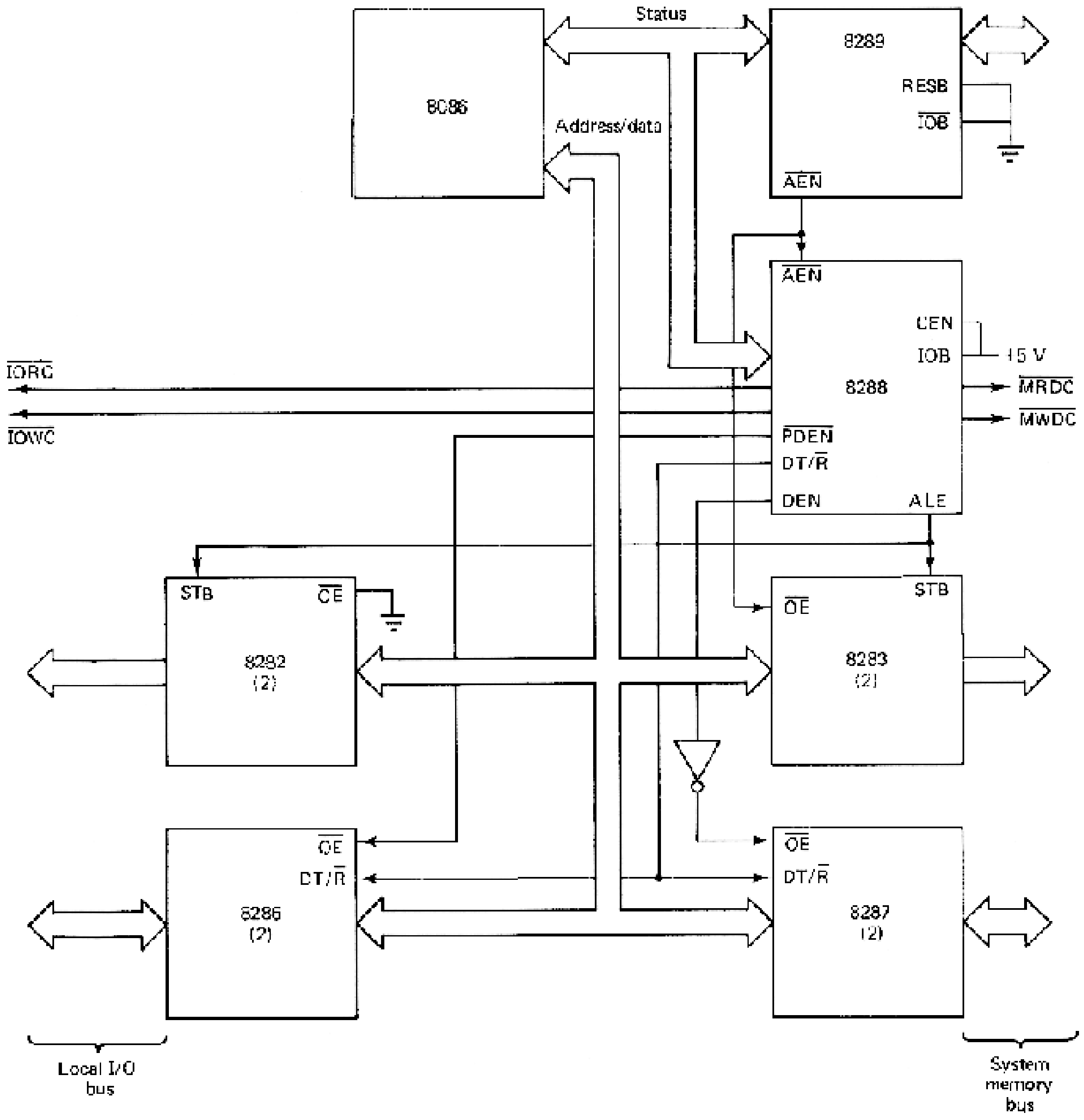


Figure 11-15 Separate local I/O bus configuration.

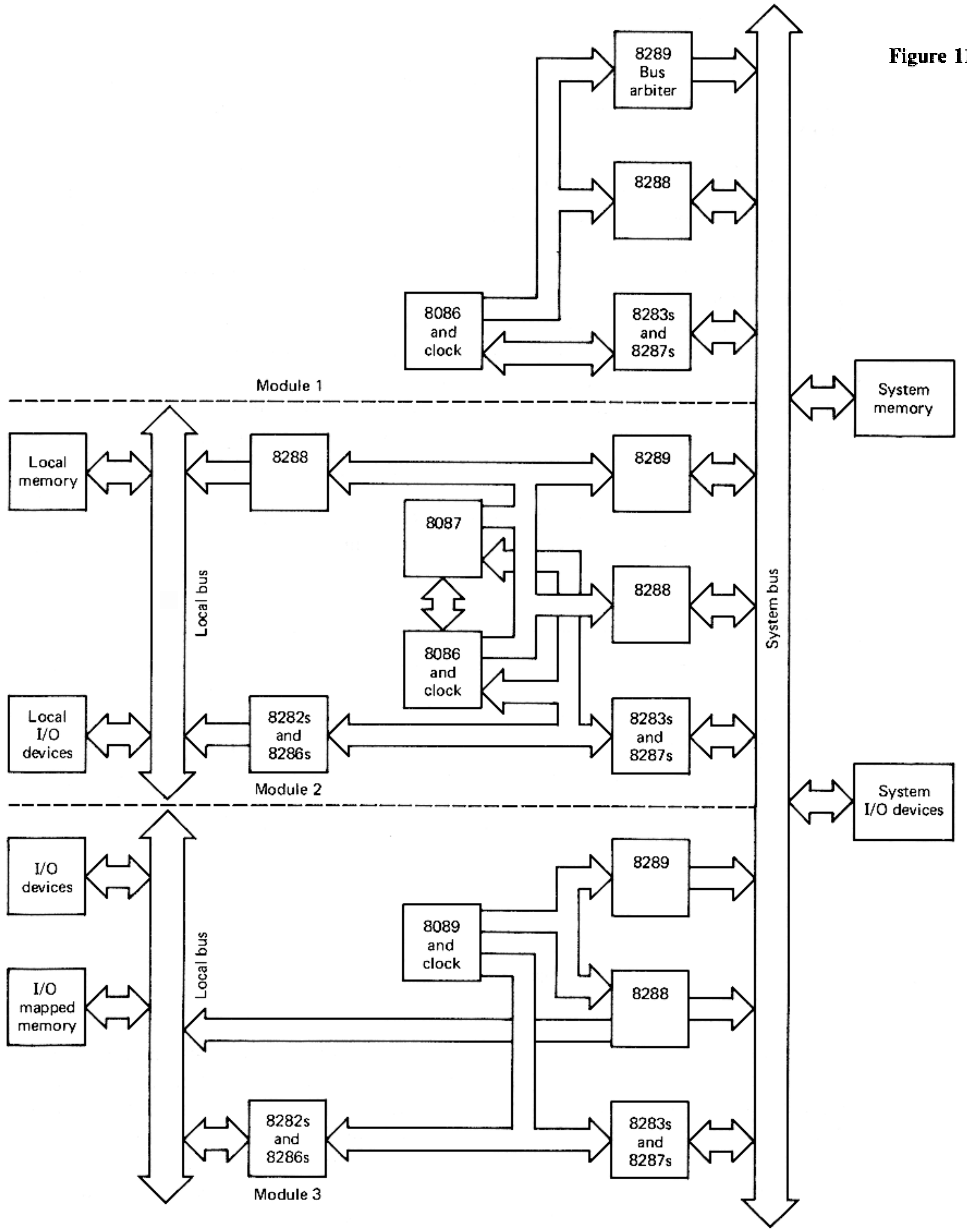


Figure 11-17 Complex multiprocessing system.

In summary, processing modules of different configurations may be combined to form a complex, loosely coupled multiprocessor system.Each module in such a system may be:

1.A single 8086 or 8088 or an independent processor such as an 8089.
2.A cluster of processors consisting of an 8086 or 8088 and a coprocessor (such as an 8087) and/or independent processors.
3.A cluster of independent processors (such as two 8089s).

In addition, each module may include a local bus or a dedicated I/O bus.

11-2-4 Microcomputer Networks

The previous multiprocessor configurations have a common characteristics and that is that all processors share the same system bus.Thus, the interprocessor communications are through shared memory and processors must be physically located close to each other.

By using serial links, many microcomputer systems can communicate with each other and share some of the same hardware and software resources.Large systems of this type are called *computer networks*.