Part 5: MULTIPROCESSOR 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

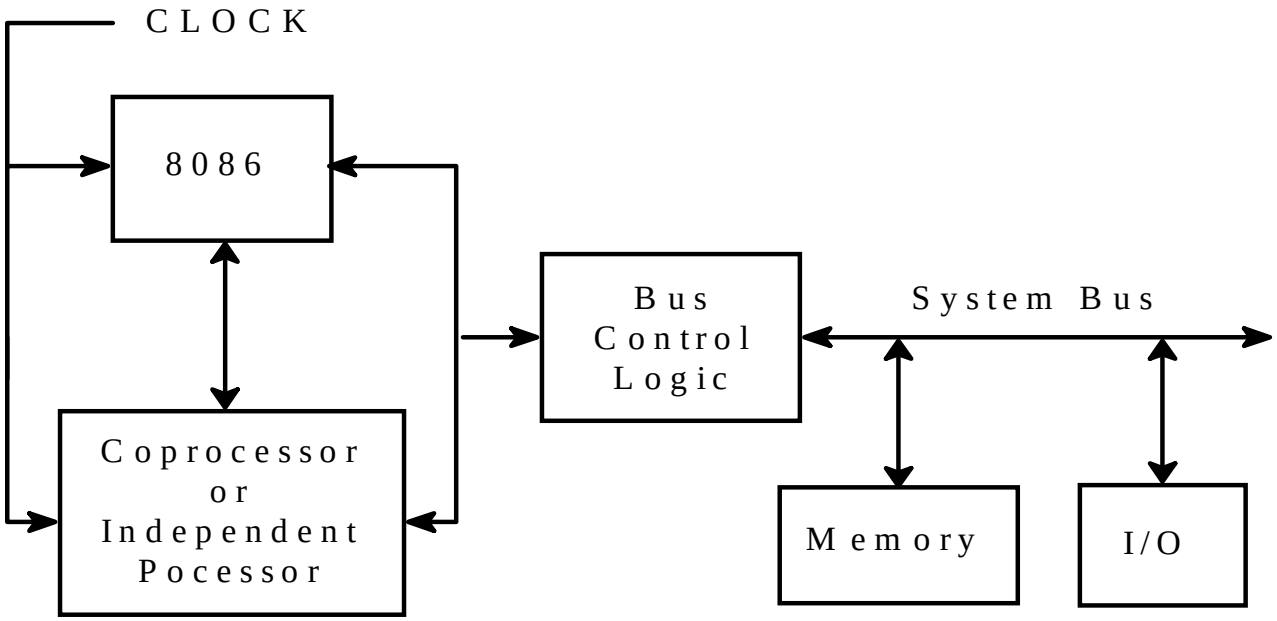
Maximum mode of 8086 is designed to implement 3 basic multiprocessor configurations:

1. coprocessor (8087)
2. closely coupled (8089)
3. loosely coupled (Multibus)

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

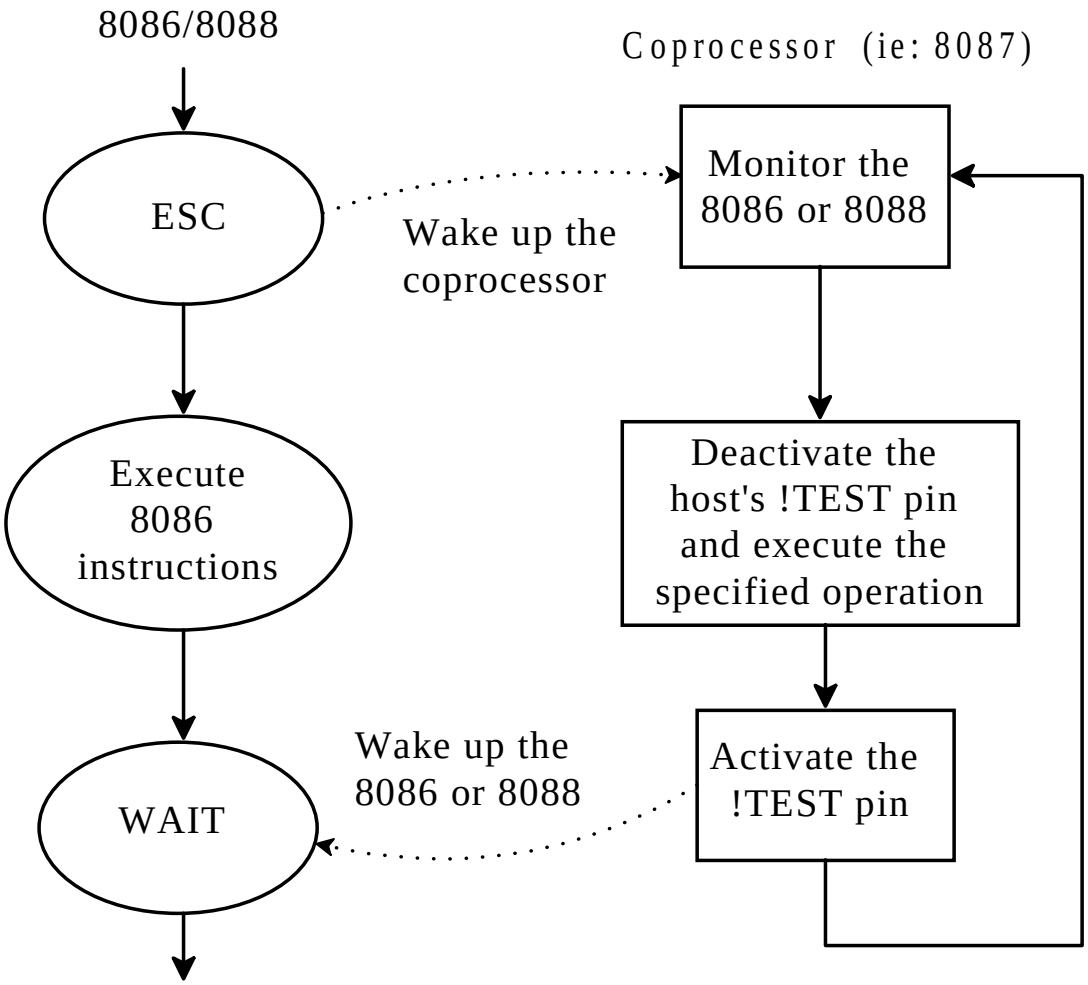
- Memory
- I/O system
- Bus & bus control logic
- Clock generator

Closely Coupled Configuration:



- Can have 8086, 8087 & 8089 running in prallel
- How do we synchronize operations?

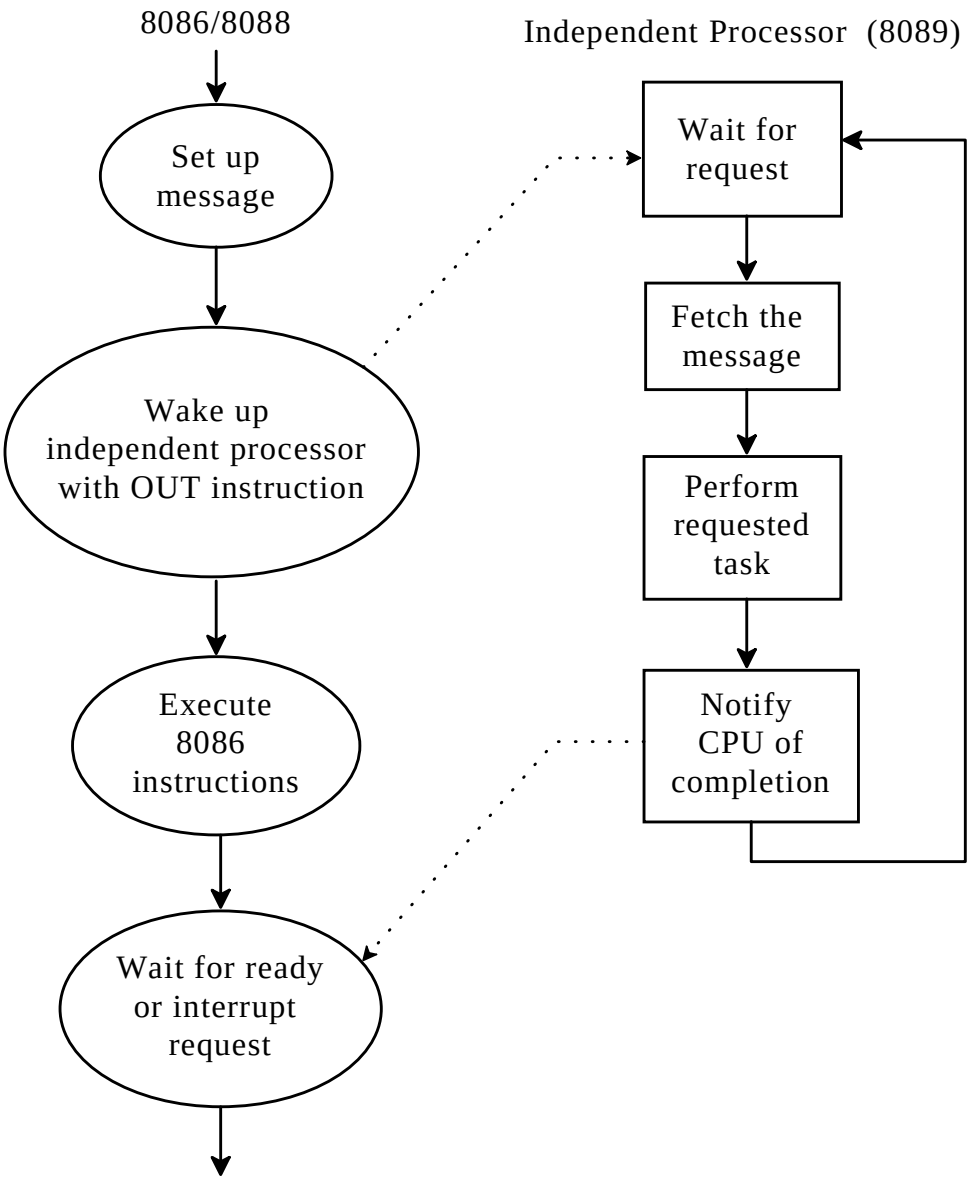
Example: 8086/8087



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

- 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

NOTE: Closely Coupled processor may take control of the bus independently Two 8086's cannot be closely coupled


8086/8088
Set up message
Wake up independent processor with OUT instruction
Execute 8086 instructions
Wait for ready or interrupt request
Independent Processor (8089)
Wait for request
Fetch the message
Perform requested task
Notify CPU of completion

Loosely Coupled Configuration: (cont)

- has shared system bus, system memory, and system I/O
- each processor has its own clock as well as its own memory (in addition to access to the system resources, such as the system clock)
- clocks are of similar frequency, but asynchro-nous towards each other

- Used for medium to large multiprocessor systems
- Each module is capable of being the bus master
- Any module could be a processor capable of being a bus master, a coprocessor configuration or a closely coupled configuration.
- No direct connections between the modules. Each share the system bus and communicate through shared resources.
- Processor in their saeparate modules can simulateneously access their private subsystems through their local busses, and perorm their local data references and instruction fetches independandtly. This results in improved degree of concurrent processing.
- Ecellent for real time applications, as separate modules can be assigned specialized tasks.

ADVANTAGES:

- high system throughput can be achieved by having more than one CPU.
- The system can be expanded in modular form. Each bus master module is an independant unit and normally resides on a separate PC board. One can be added or removed without affecting the others in the system.
- A failure in one module normally does not affect the breakdown of the entire system and the faulty module can be easily detected and replaced
- each bus master has its own local bus to access dedicated memory or IO devices so a greater degree of parallel processing can be achieved.

PROBLEMS:

- Bus Arbitration (contention): Make sure that only 1 processor can access the bus at any given time
- must synchronize local and system clocks for synchronous transfer
- requires control chips to tie into the system bus

Processor Bus Access:

- Needs some kind of priority allocation
- Output a Bus Request >BRQ= to request the bus >> BRQ line goes to some controller
- Input a Bus Grant >BGR= to gain access to bus >> BGR line from some controller
- Output a Bus Busy >BBSY= signal to hold the bus

Clocking:

- take both clocks and derive a common clock (ie: local clock & system clock)

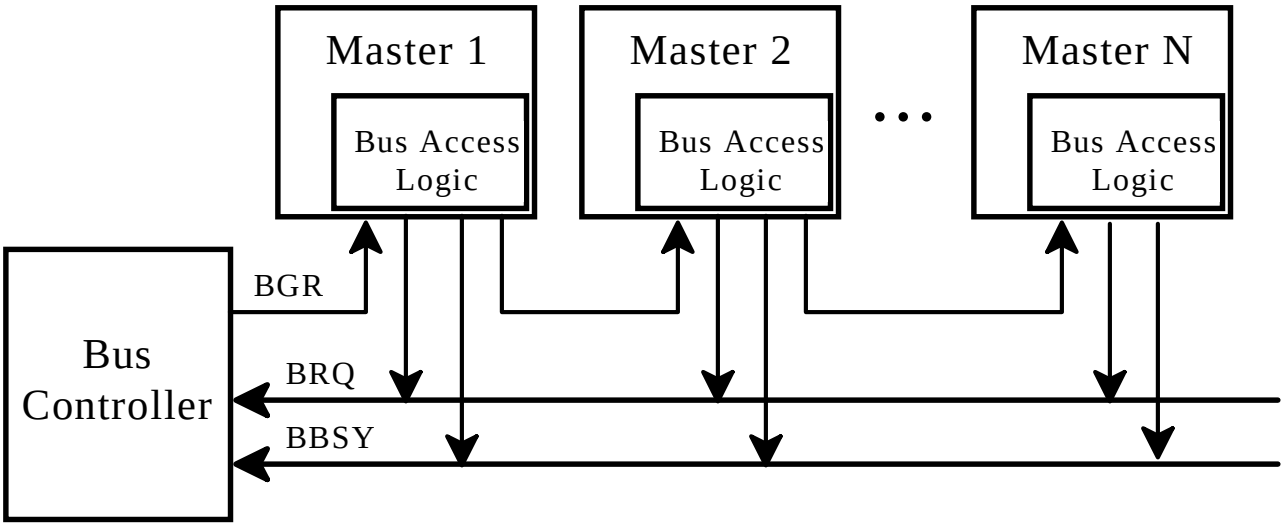
or

- take leading edge of one of the clocks >> can alternate or change for each individual operation (clock will jitter)

BUS ALLOCATION SCHEMES:

Daisy Chaining:

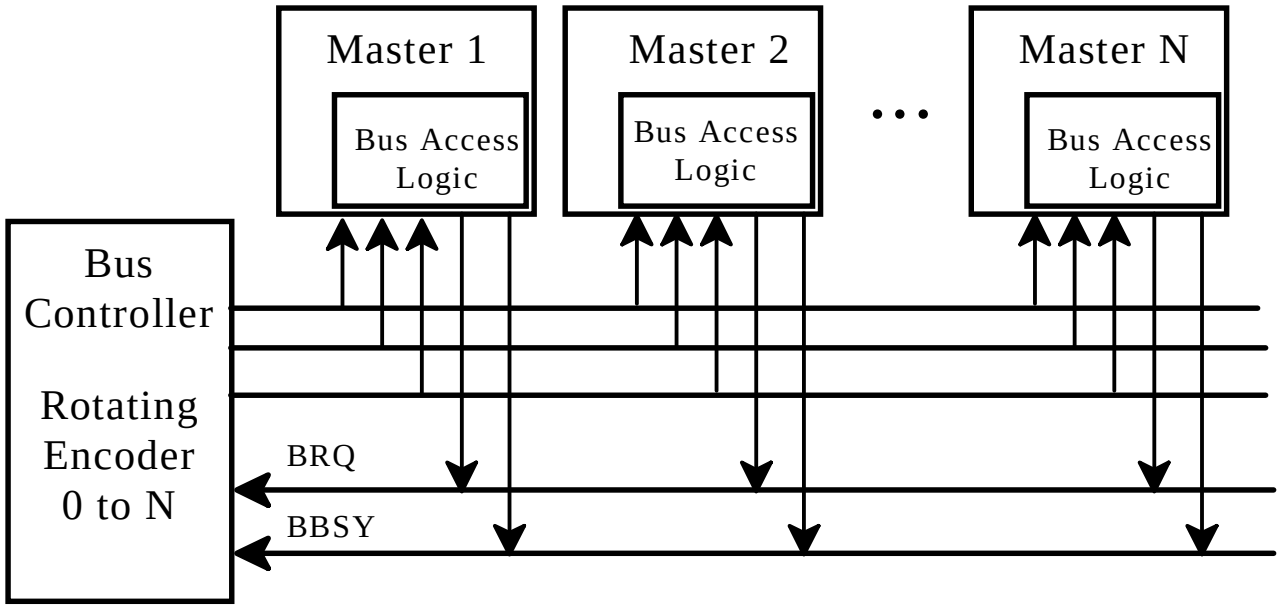
- Need a bus controller to monitor bus busy and bus request signals
- Sends a bus grant to a Master >> each Master either keeps the service or passes it on
- Controller synchronizes the clocks
- Master releases the Bus Busy signal when finished



Polling:

- Controller sends address of device to grant bus access
- Can use priority resolution here:
  memory= highest priority
- Highest priority is granted first, if it does not respond, then a lower priority is granted, and so on until someone accepts

(ie: one request line, 3-bit grant line)



Independent:

- Each master has a request and grant line

- Now just a question of priority

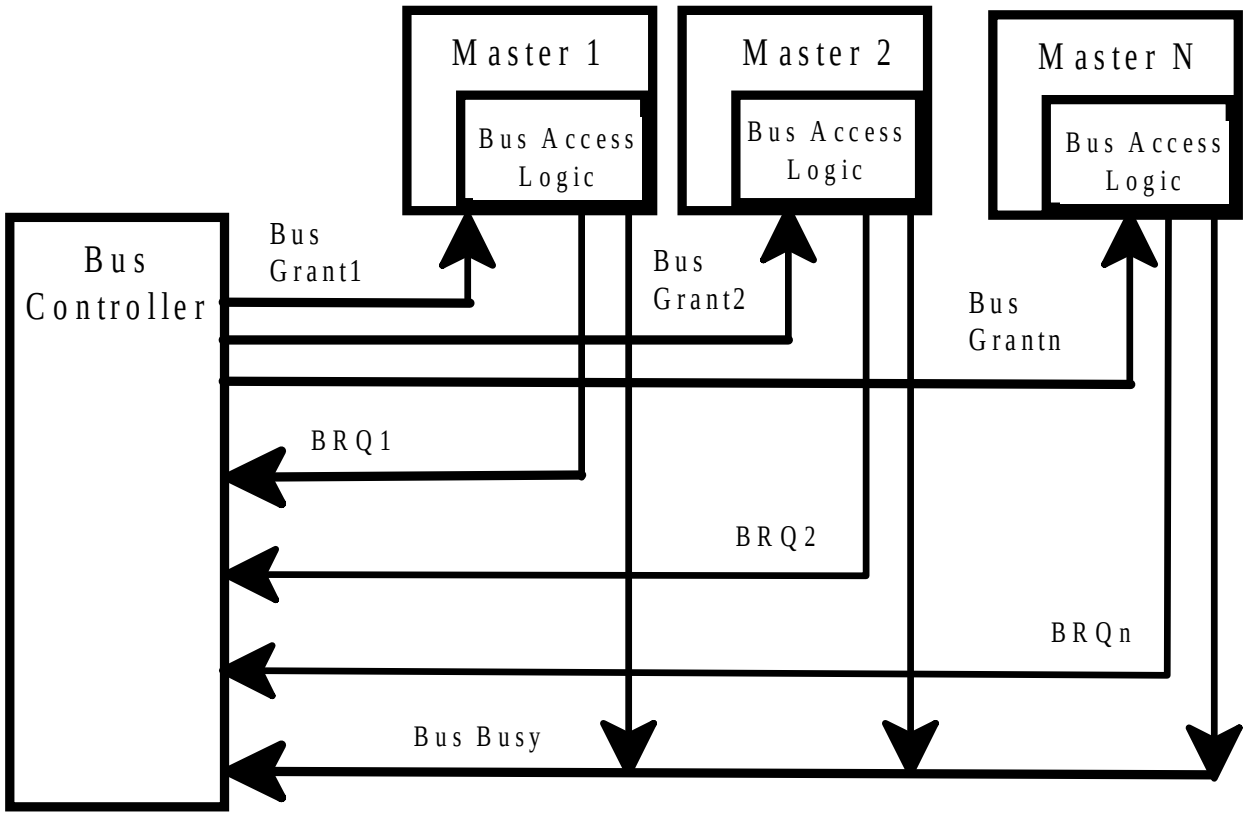
- Could have fixed priority, rotating priority, etc.
  usually fixed because memory is desired to be the highest priority

- Synchronization of the clocks must be performed once a Master is recognized

- Master will receive a common clock from one side and pass it to the controller which will derive a clock for transfer

- Can accurately predict calculations (since memory is always the highest priority)

.


Master 1
Bus Access Logic
Master 2
Bus Access Logic
Master N
Bus Access Logic
Bus Controller
Bus Grant1
Bus Grant2
Bus Grantn
BRQ1
BRQ2
BRQn
Bus Busy