MICROPROCESSOR AND COMPUTER ARCHITECTURE

UNIT-4

memory optimisation cont. & IO subsystem

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

TYPES OF CACHE MISSES

1) Compulsory Miss

- When cache initially empty, compulsory miss
- Very first access to block; cold miss
- Occur in infinite cache
- Independent of size

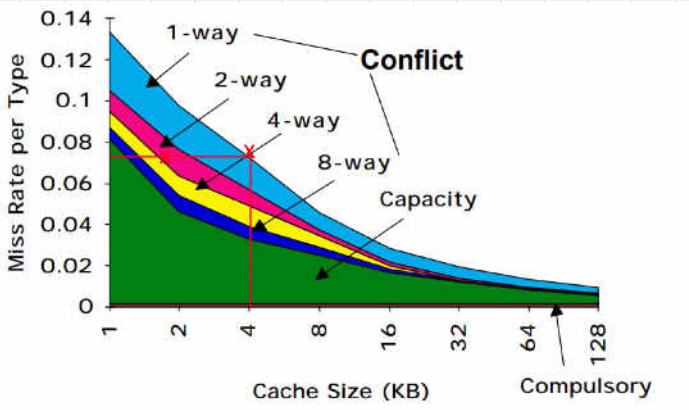
2) Capacity Miss

- Lack of space
- Cannot hold all blocks of a program
- Occur on finite FA cache
- Decrease as cache size increases

3) Conflicting Miss

- Set associative mapping or direct mapped, not FA cache
- Due to constraints, even if blocks empty
- Decrease as associativity increases

2:1 RULE

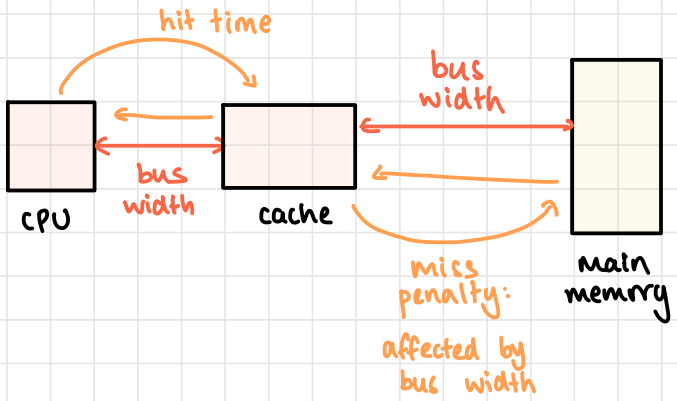


- Miss rate halves when associativity doubles

AVERAGE MEMORY ACCESS TIME

constant — additional cc if miss incurred

AMAT = Hit time + miss rate × miss penalty



- Reduce AMAT:
  - Reduce hit time: faster, smaller cache
  - Reduce miss rate: larger cache
  - Reduce miss penalty

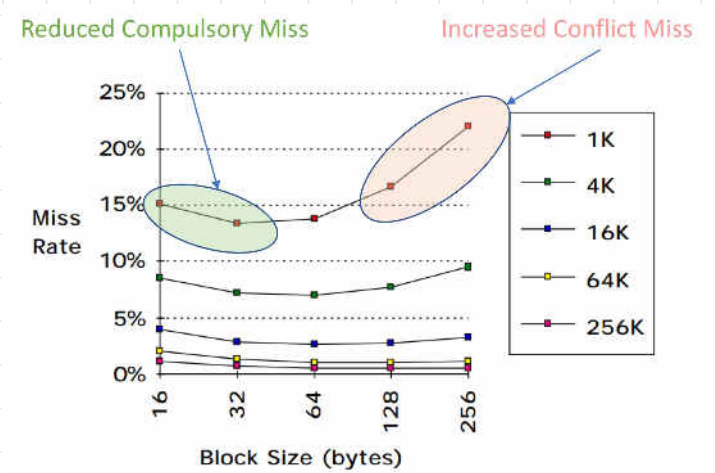
- Miss penalty depends on bus width

Six Optimisations

1. First Optimisation: Larger Block Size to Reduce Miss Rate

- block size = 8 means that 8 words fetched at a time when block is fetched

- only first word miss when CPU makes request (compulsory miss)
- if application exploits spatial locality of reference
- if block size = 16 instead, 16 words fetched at a time



DRAWBACKS

Bus Width Issue

if block size > bus width, multiple clock cycles required to fetch one block (bus width not addressed)

Cache pollution

unnecessarily bring in unwanted data due to large block size

Increased Miss Penalty

due to bus width issue

Increased Conflict Misses

due to less number of blocks and mapping constraints

ADVANTAGES

- Utilises spatial locality of reference
- Reduces compulsory misses

2. Second Optimisation: Larger Cache to Reduce Miss Rate

DRAWBACKS

- increased hit time
- increased cost, area, power (see graphs in ppt)

ADVANTAGES

- reduced capacity misses
- accomodate larger memory footprint

3. Third Optimisation: Higher Associativity to Reduce Miss Rate

- increase associativity to optimal level, not full (hit rate)

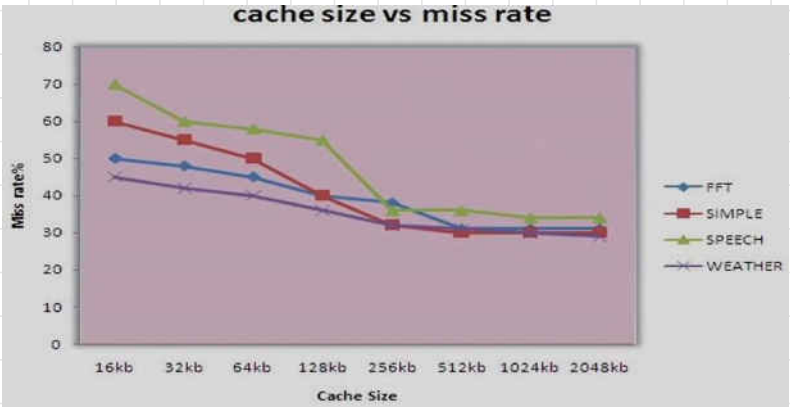
DRAWBACKS

- Increased hit time : indexing time
- complex design

ADVANTAGES

- reduced conflicting miss
- reduces miss rate and eviction rate

Cache Size vs Miss Rate



Block Size vs Miss Rate

Block Size	1K	4K	16K	64K	256K
16	15.05%	8.57%	3.94%	2.04%	1.09%
32	13.34%	7.24%	2.87%	1.35%	0.70%
64	13.76%	7.00%	2.64%	1.06%	0.51%
128	16.64%	7.78%	2.77%	1.02%	0.49%
256	22.01%	9.51%	3.29%	1.15%	0.49%

Cache sizes

Miss rates

Q: Assume the memory system takes 80 clock cycles of overhead and then delivers 16 bytes every 2 clock cycles. That is, it can supply 16 bytes in 82 clock cycles, 32 bytes in 84 clock cycles, and so on... Which block size has the smallest average memory access time for each cache size?

Miss Rates

Block size	Cache size 4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

AMAT = hit time + miss rate x miss penalty

Assume hit time = 1 cc

For cache size 4K

miss penalty of 16-byte block = $\frac{16}{16} \times 2 + 80 = 82$

32-byte block = 84 cc

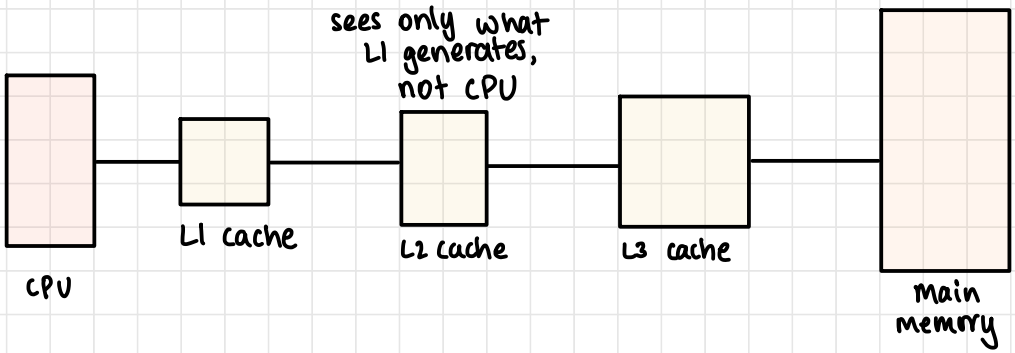
64-byte block = 88 cc

128-byte block = 96 cc

256-byte block = 112 cc

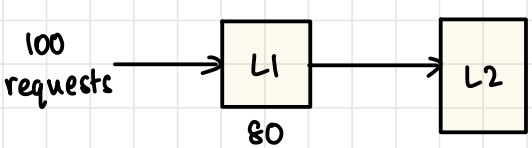
AMAT
16 byte = 1 + 8.57% x 82 = 8.0274
32 byte = 1 + 7.24% x 84 = 7.0816 → lowest
64 byte = 1 + 7.00% x 88 = 7.16
128 byte = 1 + 7.78% x 96 = 8.4688
256 byte = 1 + 9.51% x 112 = 11.6512

4. Fourth Optimisation: Multilevel Caches to Reduce Miss Penalty



- Memory wall problem: gap between memory & processor speeds
  - smaller, faster cache
  - larger cache
  - trade-off required

- Miss rate:
  - global miss rate (wrt CPU) miss rate$_{L1}$ x miss rate$_{L2}$
  - local miss rate (higher cache level)



HR L1 = 80/100
MR L1 = 20/100

HR L2 = 15/20
MR L2 = 5/20
GMR = 5/100

For 2-Level Cache

AMAT = hit time + miss rate × miss penalty

= $\text{hit time}_{L1} + \text{miss rate}_{L1} \times \text{miss penalty}_{L1}$ —— (1)

$\text{miss penalty}_{L1} = \text{hit time}_{L2} + \text{miss rate}_{L2} \times \text{miss penalty}_{L2}$ —— (2)

(2) in (1)

$\text{AMAT} = \text{hit time}_{L1} + \text{miss rate}_{L1} \times (\text{hit time}_{L2} + \text{miss rate}_{L2} \times \text{miss penalty}_{L2})$

Avg mem stalls per inst = $\text{Miss per inst}_{L1} \times \text{Hit Time}_{L2} + \text{Miss per instr}_{L2} \times \text{miss pen}_{L2}$

Q: Suppose that in 1000 memory references there are 40 misses in the first level cache and 20 misses in the second – level cache. What are the various miss rates?

Assume the miss penalty from the L2 cache to memory is 200 clock cycles, the hit time of the L2 cache is 10 clock cycles, hit time for L1 cache is 1 clock cycle and there are 1.5 memory references per instruction.

What is the average memory access time and average stall cycles per instruction? Ignore impact of writes.

miss rate (local & global) of L1 cache = $\frac{40}{1000}$

= 0.04

global miss rate L2 cache $= \frac{20}{1000} = 0.02$

local miss rate L2 cache $= \frac{20}{40} = 0.5$

miss penalty$_{L2}$ = 200 CC

hit time$_{L1}$ = 1 CC hit time$_{L2}$ = 10 CC

1.5 ref/inst

AMAT = hit time$_{L1}$ + miss rate$_{L1}$ x (hit time$_{L2}$ + miss rate$_{L2}$ x miss penalty$_{L2}$)

Avg mem stalls per inst = Miss per inst$_{L1}$ x Hit Time$_{L2}$ + Miss per instr$_{L2}$ x miss pen$_{L2}$

= miss rate$_{L1}$ x miss penalty$_{L1}$

AMAT = 1 + 0.04 (10 + 0.5 x 200)

= 1 + 0.04 (10 + 100) = 1 + 0.04 (110) = 5.4 CC

AMSPI = 0.04

Average memory stalls per instruction = (Miss Rate$_{L1}$ x Miss Penalty$_{L1}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$)* Memory Reference per Instruction

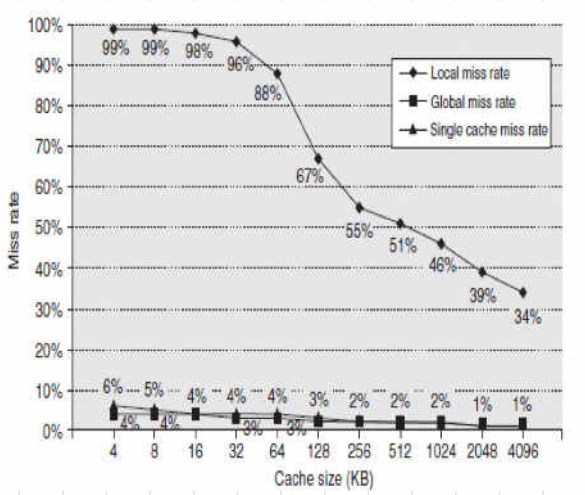
Q. Consider a system with a 2-level cache. Access time of L1 cache, L2 cache and main memory are 1 ns, 10 ns and 500 ns respectively. The hit rates of L1 cache, L2 cache are 0.8 and 0.9 respectively. What is AMAT?

AMAT = $\text{hit time}_{L1} + \text{miss rate}_{L1} \times (\text{hit time}_{L2} + \text{miss rate}_{L2} \times \text{miss penalty}_{L2})$

$$\text{AMAT} = 1 + 0.2(10 + 0.1 \times 500)$$

$$= 13 \text{ ns}$$

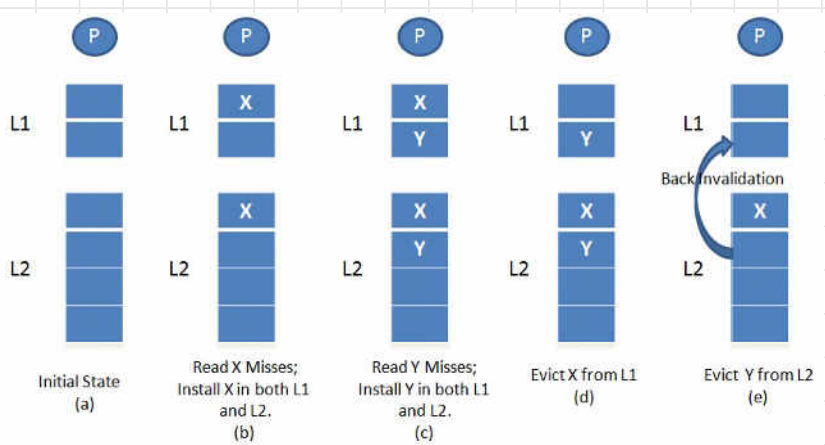
Cache size vs miss rate



Types of Cache Hierarchy

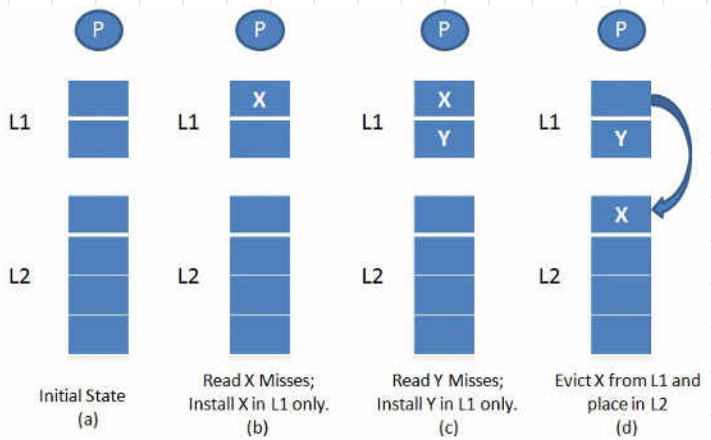
1. Inclusive Cache

- L1 is a subset of L2
- Cache size = L2
- miss on L1, search L2
- any modifications on L1 must be reflected in L2 (write through or write back)



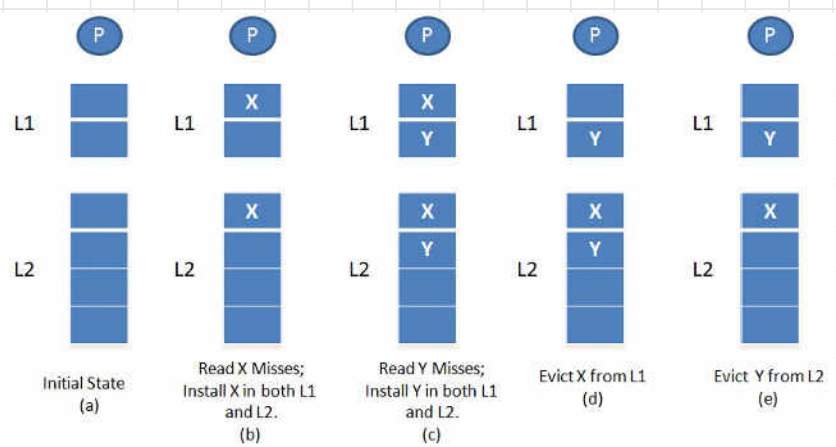
2. Exclusive

- L2 only contains blocks not present in L1, and so on
- modern day caches
- Read miss on L1, search L2 and if hit, block moved to L1 and possible evicted block moved to L2 cache
- L2: victim cache
- If miss on L2 also, fetched from mem and placed in L1



3. Non-Inclusive Non-Exclusive

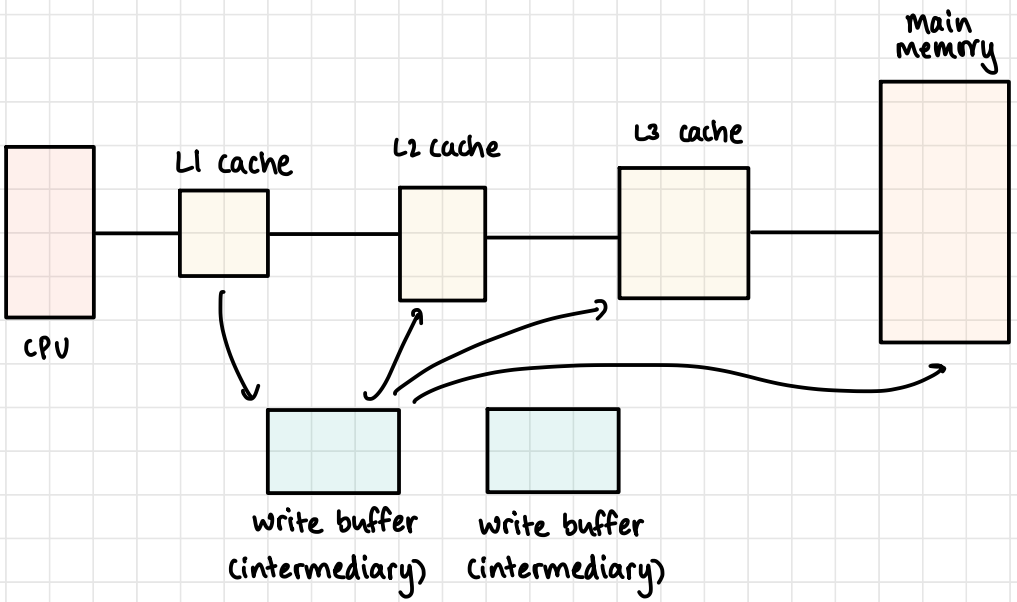
- back inclusive
- most modern processors



- data in L1 may or may not be present in L2

5. Fifth Optimisation: Giving Priority to Read Misses Over Write Misses

- Processor issues read or write request
- Read request: performance — in order to continue with CPU operation read must be fulfilled

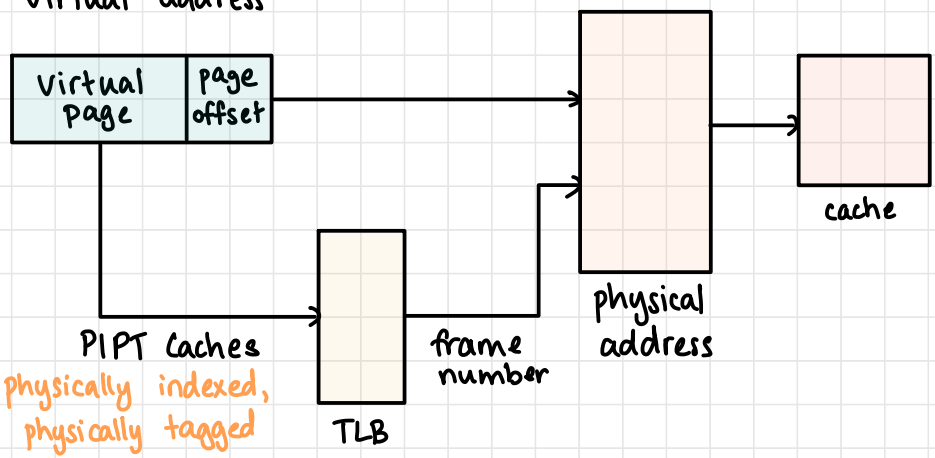


DRAWBACKS

- If processor issues request to recently evicted block and read miss occurs, L2 level of cache may supply outdated data as dirty block is in write buffer
- In other words, RAW hazard

6. Sixth Optimisation: Avoid Address Translation in Cache Indexing To Reduce Hit Time

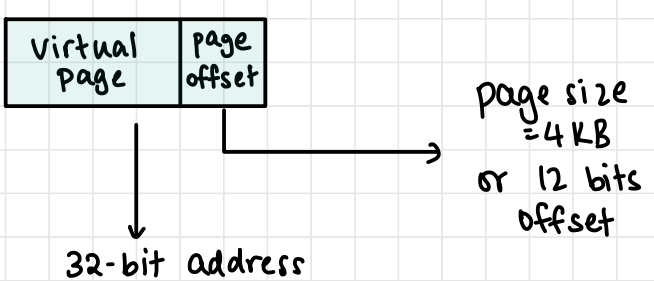
- Virtual address

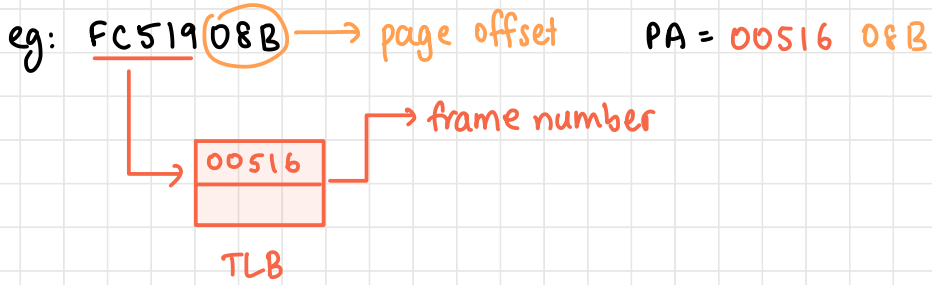


Total Hit Latency = TLB hit latency + Cache hit latency

- Solution: VIPT Cache: Virtually Indexed Physically Tagged Caches (indexing into cache using page offset)
- Do not wait for VA to translate to PA; extract index from VA and extract tag no from PA

Example:

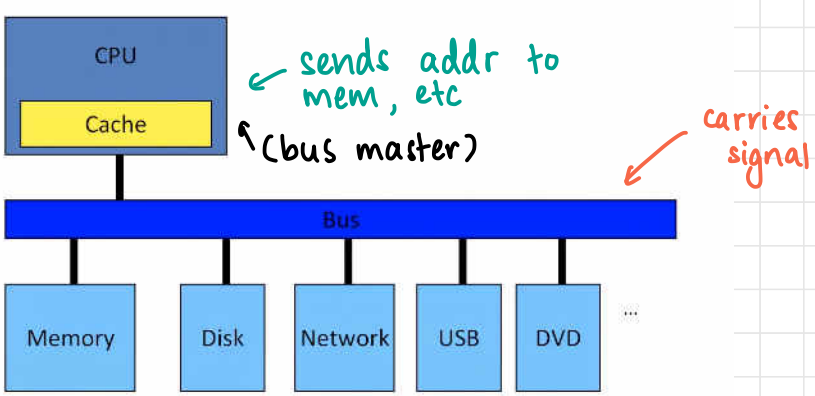




I/O Devices

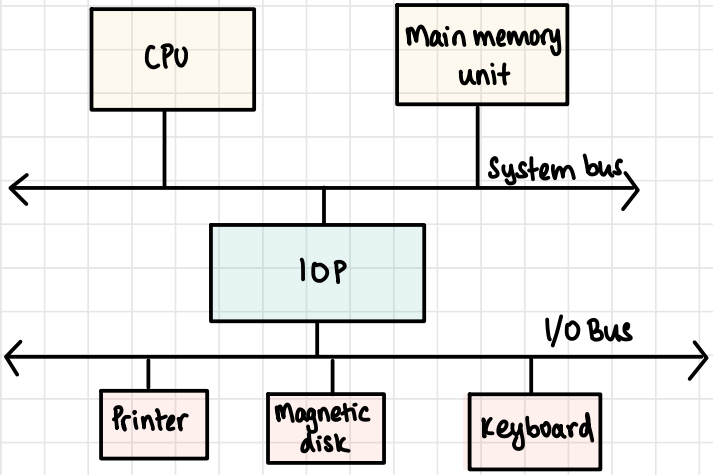
- Every device associated with device drivers
- Moment device plugged in, system automatically detects device and runs device driver
- Connected via buses (lines – 32 bit, 64 bit etc)
- Coordination between devices, bus architecture

ACCESSING I/O DEVICES

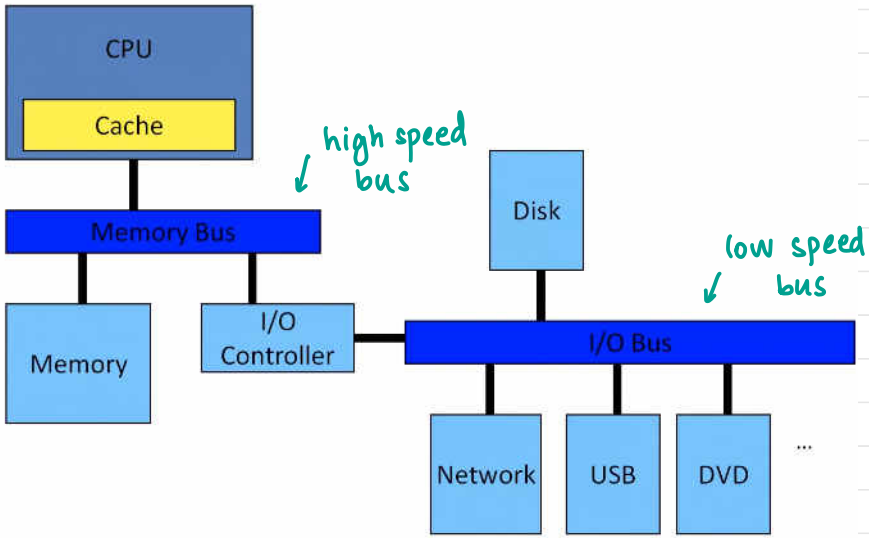


- Decoding happens

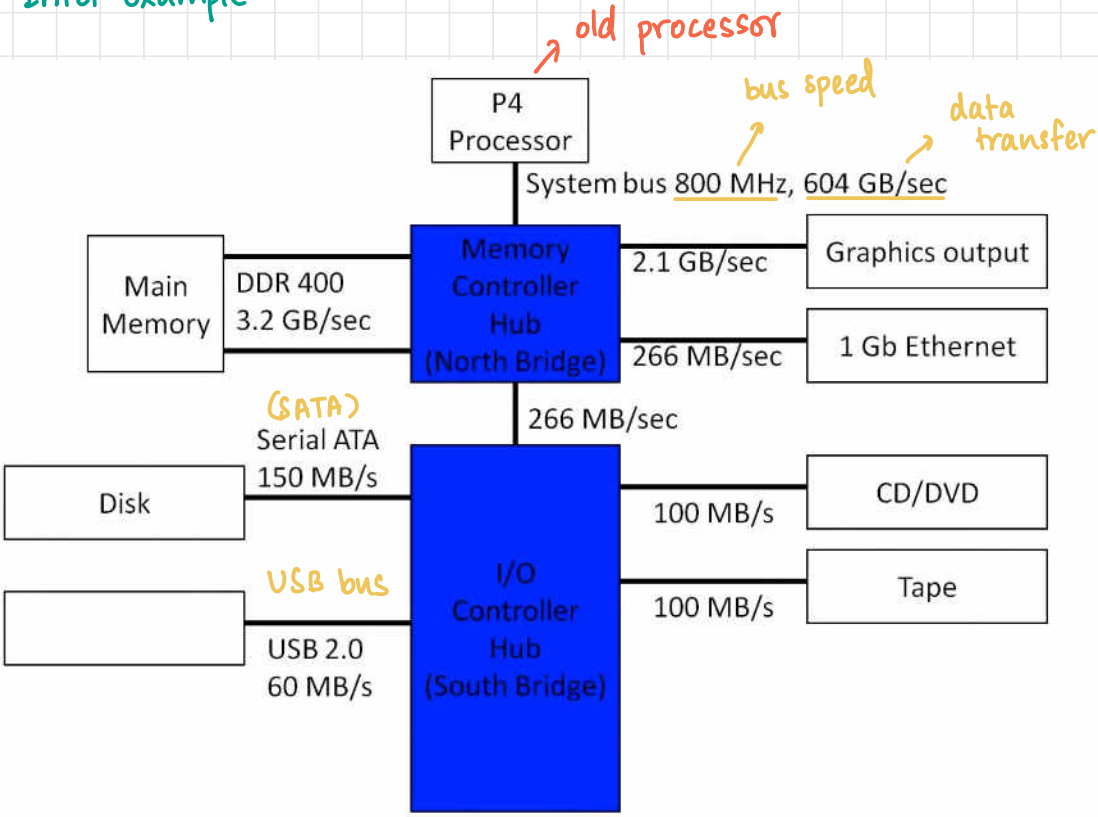
I/O Interface



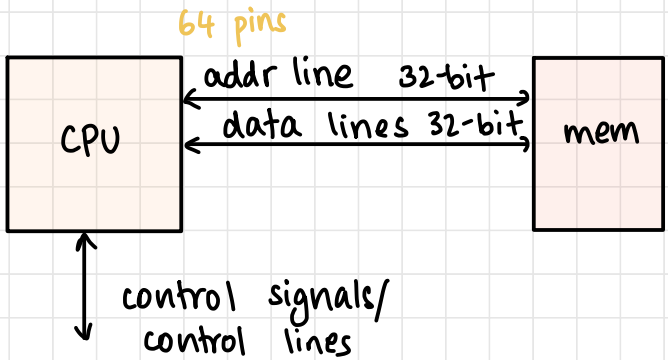
I/O Hierarchy

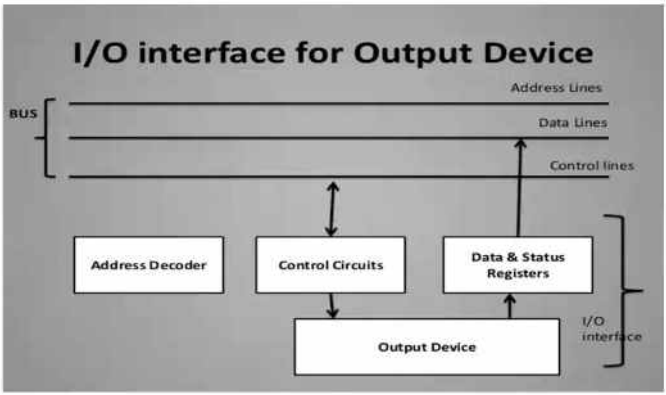


Intel Example



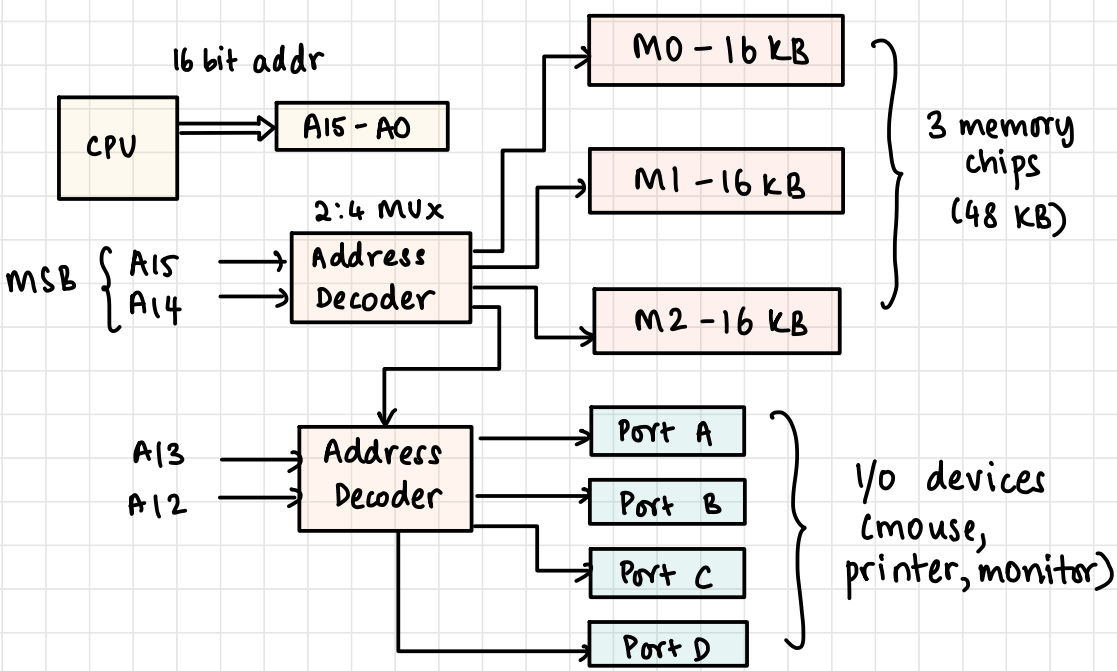
I/O Interface





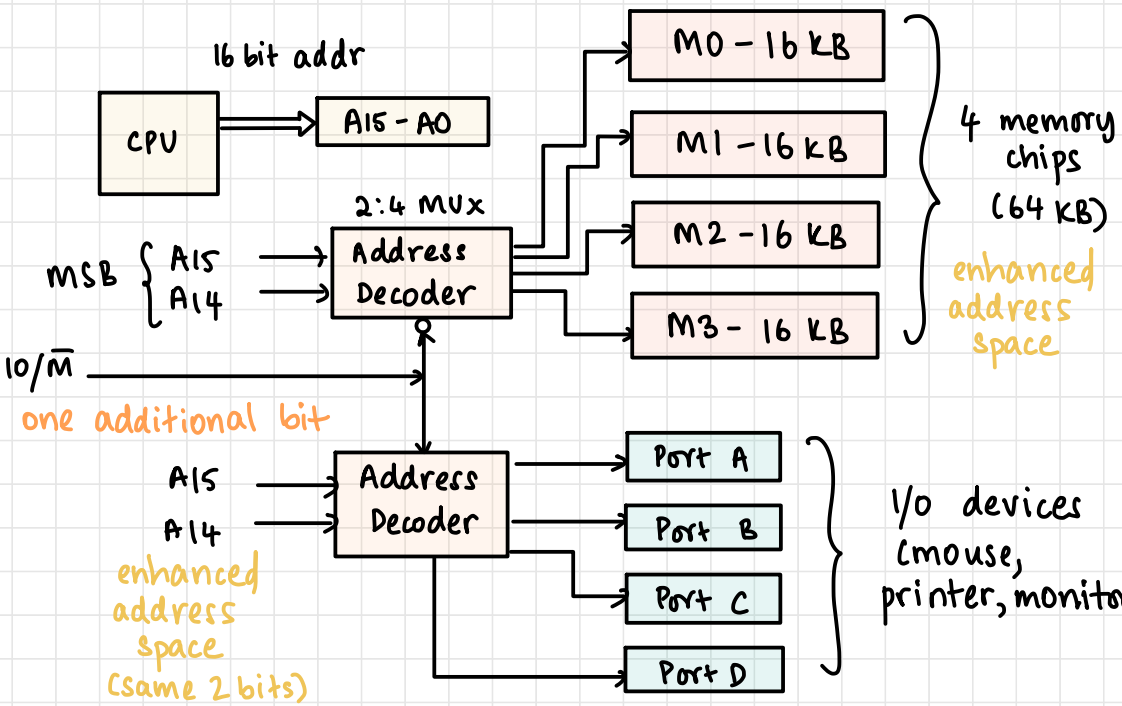
1. **Memory Mapped I/O**

- Same memory mapped I/O technique for data transfer and I/O transfer
- No separate I/O instructions; all data transfer instrs (LDR, STR)



2. I/O Mapped I/O

- Separate I/O data transfer instructions (IN/OUT)
- In buffer, out buffer



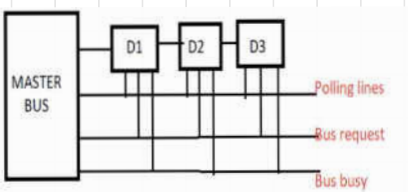
DATA TRANSFER TECHNIQUES

1. Programmed I/O

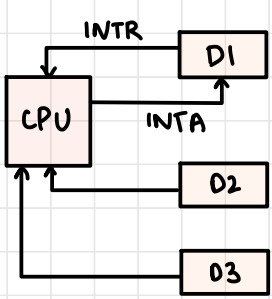
- CPU executes program that transfers data between I/O device and memory

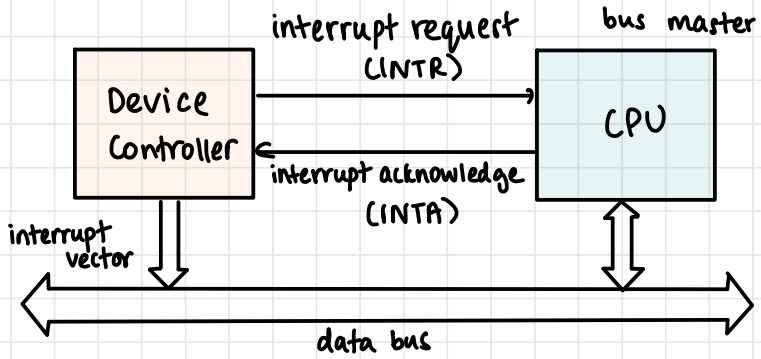
(i) Synchronous
- fixed rate of transfer (dictated by program)

(ii) Asynchronous
- handshaking — polling for sending / receiving data
- check status of devices
- status bit - polling technique



(iii) Interrupt-driven
- interrupt / exception ⟶ ISR
- CPU initiates data transfer and continues onto other tasks
- when I/O device ready, informs CPU through interrupt
- services interrupt with ISR and then returns back
- CC not wasted on polling

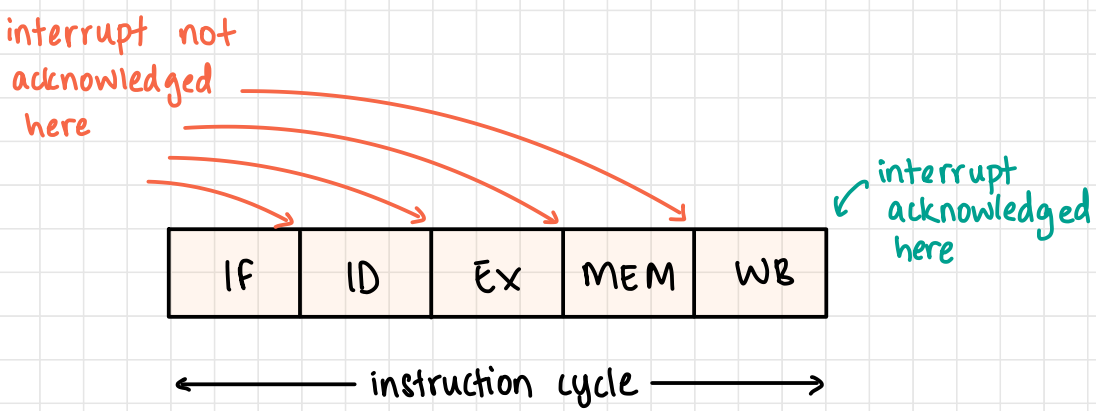




ARM Interrupt Table

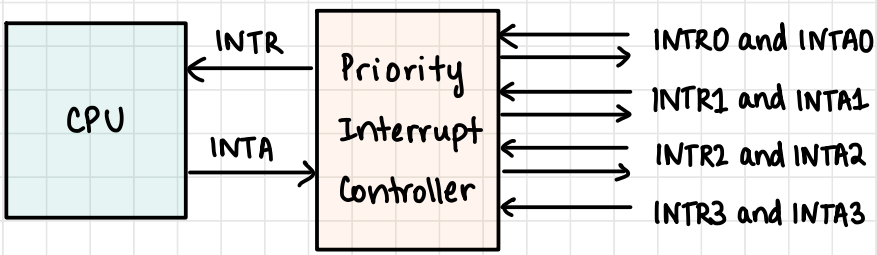
	Address	Exception	Mode in Entry
1	0x00000000	Reset	Supervisor
2	0x00000004	Undefined instruction	Undefined
3	0x00000008	Software Interrupt	Supervisor
4	0x0000000C	Abort (prefetch)	Abort
5	0x00000010	Abort (data)	Abort
x	0x00000014	Reserved	Reserved
6	0x00000018	IRQ (external interrupt)	IRQ
7	0x0000001C	FIQ (fast interrupt)	FIQ

- interrupt acknowledged after WB stage only



PRIORITY INTERRUPT CONTROLLER

- Multiple devices, multiple interrupts (interrupt stack)
- Simultaneous interrupts dealt with by using priority interrupt controller
- Sends interrupt vector to CPU for interrupt requests
- Controller connected to multiple devices on one side and CPU on the other



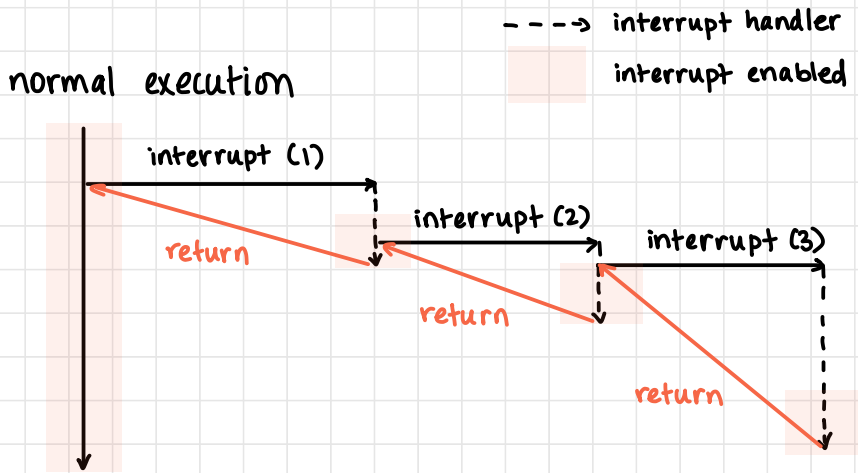
Nested Interrupts

- If an interrupt is being serviced (say, by device D0) and another interrupt (by device D1) is requested, one of two things can happen

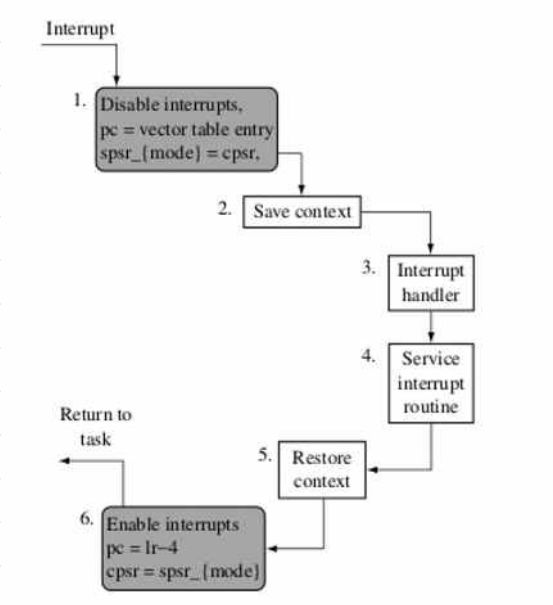
1. The interrupt being serviced (D0) gets interrupted (D1) and serviced first before D0 can complete executing. This leads to the problem of nested interruption

2. The interrupt request for D1 is not serviced until the ISR of D0 completes executing. Here, no nesting occurs.

1. Nested interrupts

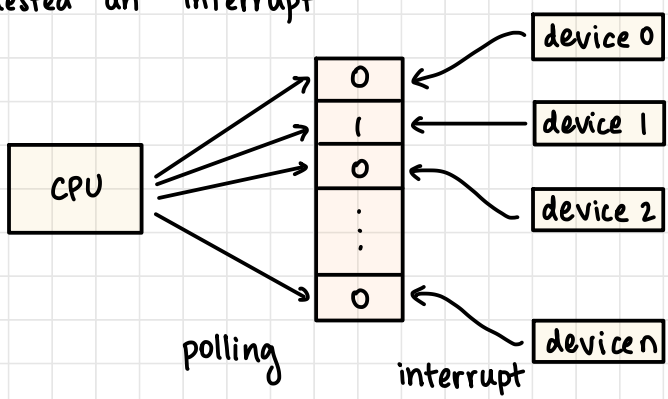


2. Non-Nested Interrupt Handler (NNIH)



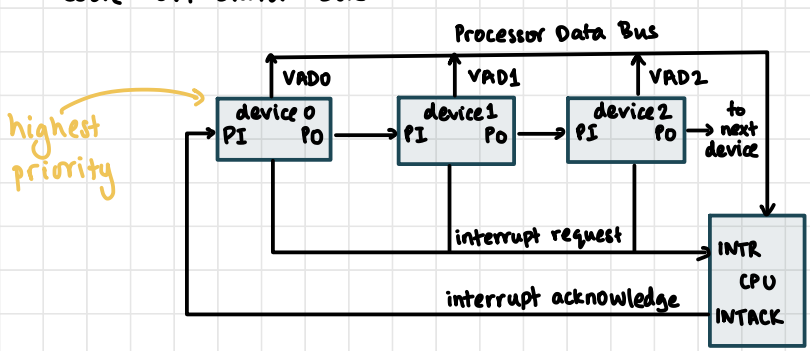
Polling Technique

- Each device has associated with it a status bit (0 or 1) depicting whether or not it has requested for an interrupt
- CPU polls the status bits to check which devices have requested an interrupt

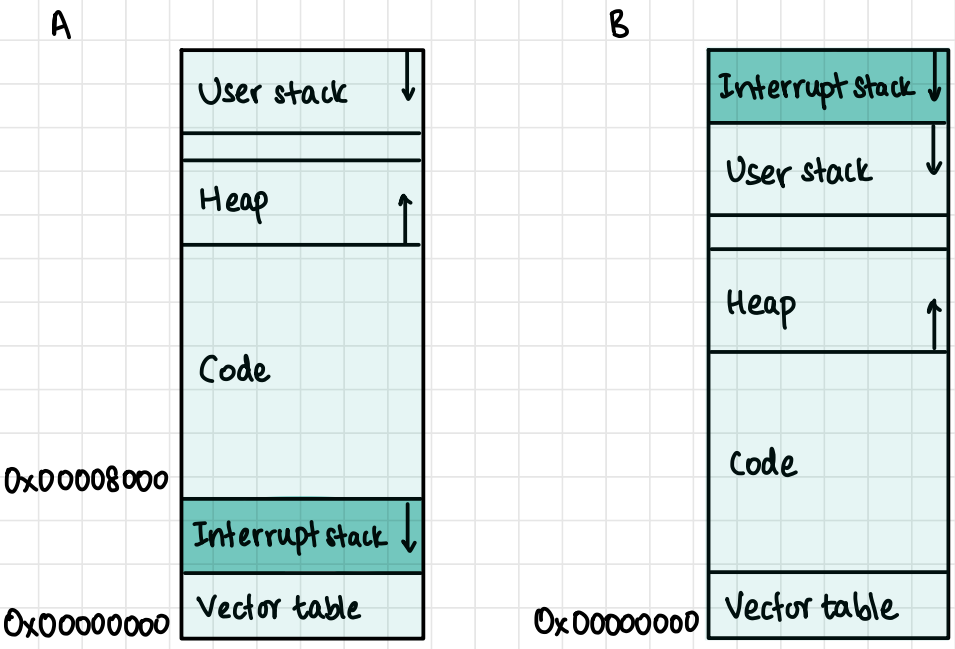


Daisy Chain Technique

- Common INTR line for all devices
- INTA line connected in a daisy-chain fashion (INTACK)
- If device receives INTA: passes to next device if it has not raised interrupt, else stops INTA and puts identifying code on data bus



interrupt stack

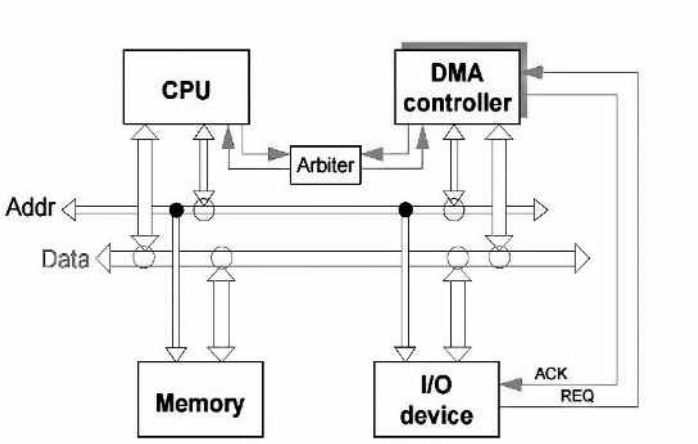


Interrupt Handler

```
interrupt_handler
        SUB     r14,r14,#4                      ; adjust lr
        STMFD   r13!,{r0-r3,r12,r14}            ; save context
        <interrupt service routine>
        LDMFD   r13!,{r0-r3,r12,pc}^            ; return
```

2. Direct Memory Access (DMA)

- External controller directly transfers data between I/O devices and memory without CPU intervention



- Bus in tristate / high impedence state — bus is free
- DMA controller for block data transfer between devices and memory
- Count register, starting address, destination address required by DMA; CPU gives information to DMA controller
- Cycle stealing process — DMA controller is bus master and no further interrupts allowed

Bus architecture - see slides