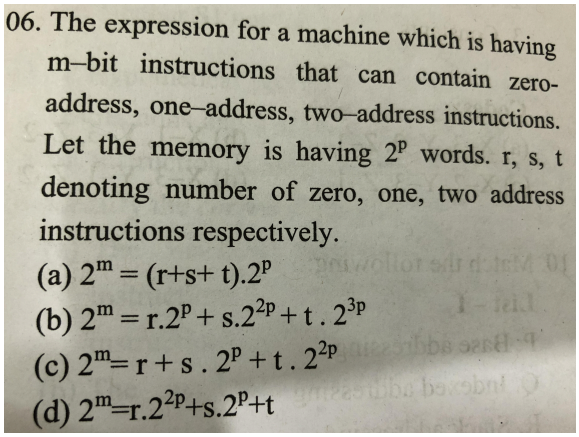
Computer Organisation and Architecture


06. The expression for a machine which is having m–bit instructions that can contain zero-address, one–address, two–address instructions. Let the memory is having 2^p words. r, s, t denoting number of zero, one, two address instructions respectively.

(a) $2^m = (r+s+t).2^p$

(b) $2^m = r.2^p + s.2^{2p} + t \,.\, 2^{3p}$

(c) $2^m = r + s \,.\, 2^p + t \,.\, 2^{2p}$

(d) $2^m = r.2^{2p} + s.2^p + t$

2^m = r + s*2^p+t*2^2p

The primary function of any processor (microprocessor) is to execute a sequence of instructions i.e. programs.

The **program** is stored in the external Main Memory.

Program Execution is carried as follows:

1. **CPU** transfers instructions/input data(operands) from **Main memory** to **Registers**.
2. **CPU** executes the instruction in the stored sequence except when the sequence is explicitly altered by a branch instruction.
3. When necessary, CPU transfers output data(result) from registers to Main Memory.

The efficient management of these instructions and data streams is the basic function of the CPU.

As CPU registers are 100 times or faster than the Main memory, most of the hardware designers have arrived at a decision to apply a CACHE MEMORY in between CPU registers and Main Memory. The cache is smaller and Faster than Main Memory and may reside wholly or partially in the same chip as the CPU.

CPU communicates with IO devices in the same way as with Main Memory. In some computers there are no IO instructions per se; all IO data transfers are implemented by memory referencing instructions, an approach called **Memory-mapped IO**. This requires memory and IO ports to share the same set of addresses.

Other computers employ IO instructions that are distinct from memory referencing instructions. These instructions produce control signals to which only IO ports but not the memory locations respond. This second approach is called **IO Mapped IO**.

User and Supervisor Modes

Programs executed by CPU are of two types: User program, supervisor program.
The user program handles specific applications like word processing.
The supervisor program manages various routine tasks on behalf of its users.*(It is typically part of the computer's OS).* Supervisory functions include controlling a graphics interface, transferring data between Hard Disk and Main Memory.
CPU switches frequently between user and supervisor modes. For example in case of handling an interrupt to the CPU.

CPU Operation

The sequence of operations performed by the CPU in processing an instruction constitutes an instruction cycle. Details of instruction cycle vary with type of instruction, all instructions require two major steps: a **FETCH** step during which a new instruction is read from external memory **M** and an **EXECUTE** step during an operation specified by the instructions are executed.
A check for pending interrupt requests is also usually included in the instruction cycle as shown in *fig. 3.2*

Actions of the CPU during an instruction cycle are defined by a **sequence of micro-operations.**
The *time* required for the shortest well-defined *micro-operation* is the **CPU cycle time** or **Clock period T_{clock}** and is the basic unit of time for measuring CPU actions.
CPU clock frequency $f = 1/T_{clock}$
We will assume that each instruction is fetched from memory in 1 clock cycle and can be executed in another CPU cycle.
Accumulator Based CPU

Cache memory uses S-RAM, and Main Memory uses D-RAM because, Cache Memory we need fast access and in RAM, we need to make it more economical at the same time, giving dynamic access and high density.

4cc to refresh a row,
RAM is 8KB i.e. 8192 rows are there
This means 8192*4 = 32768 clocks just to refresh all rows
Clock Speed: 133MHz which means 1 second has 133000000 oscillations.
Total time in refreshing: 32768/133000000 = 246 micro seconds.
SRAM has a refreshing period of 64 milliseconds.
i.e. 0.4% time in refreshing.

Tag is used to identify which of the blocks mapped to this cache are resident in this cache?

Main memory is divided into three parts, block offset, line offset and tag bits

- Throughput of a pipeline is(1/D) MIPS where D is Max stage time + delay.
- **Throughput** = n / (k + n – 1) * Tp
- Addressing smaller memories is much faster, as the size of the decoder plays a crucial role in determining the time for the decoding operation.
- Big endian is generally what we use, storing MSB first, little endian is just the opposite.
- Byte alignment : there is a buffer, we have to store new data as per the byte alignment.
- Instruction set architecture is of 4 types, stack. Accumulator, Reg-Mem, Load store architecture
- Addressing modes

– **Register**	add r1, r2	r1 <- r1+r2
– **Immediate**	add r1, #5	r1 <- r1+5
– **Direct**	add r1, (0x200)	r1 <- r1+M[0x200]
– **Register indirect**	add r1, (r2)	r1 <- r1+M[r2]
– **Displacement**	add r1, 100(r2)	r1 <- r1+M[r2+100]
– **Indexed**	add r1, (r2+r3)	r1 <- r1+M[r2+r3]
– **Scaled**	add r1, (r2+r3*4)	r1 <- r1+M[r2+r3*4]
– **Memory indirect**	add r1, @(r2)	r1 <- r1+M[M[r2]]
– **Auto-increment**	add r1, (r2)+	r1 <- r1+M[r2], r2++
– **Auto-decrement**	add r1, -(r2)	r2--, r1 <- r1+M[r2]

- When program changes, speedup changes and execution time changes
- Clock is represented in terms of number of c
- ❖ All processors are driven by clock.
- ❖ Expressed as clock rate in GHz or clock period in ns
- ❖ CPU Time = CPU clock cycles x clock cycle time

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction Count}}$$

$$\text{CPU Time} = \text{IC} \times \text{CPI} \times \text{CCT}$$

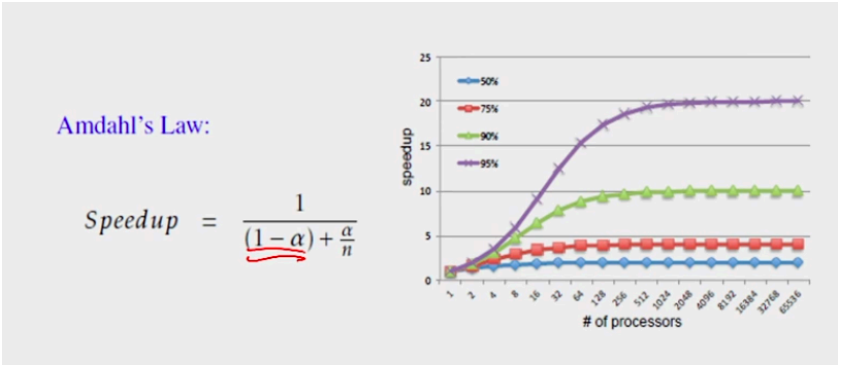
$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instructions}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

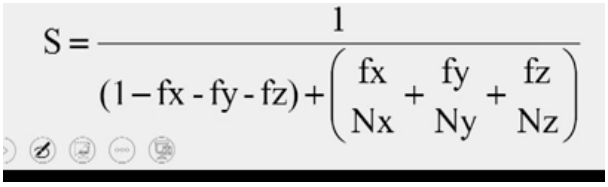
-
-

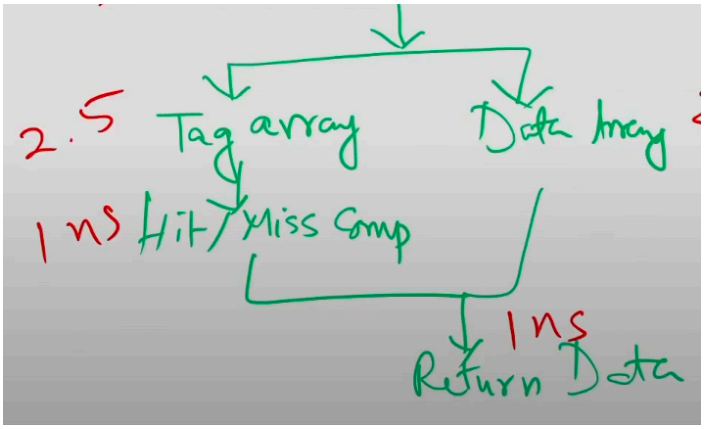
$$ET_{new} = ET_{old} \times (1 - \text{fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}$$

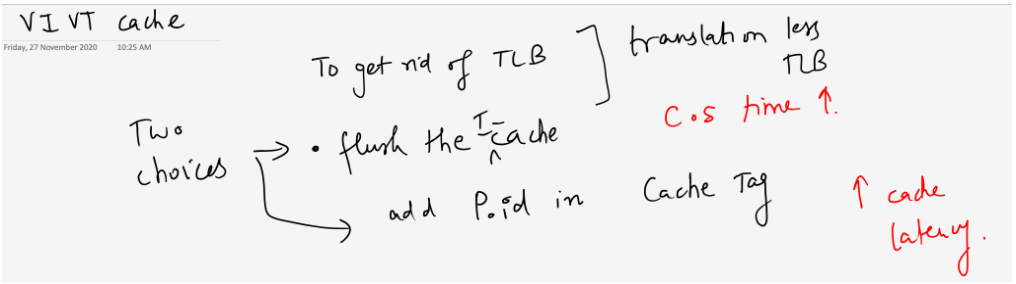
$$Speedup = \frac{ET_{old}}{ET_{new}} = \frac{1}{(1 - \text{fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

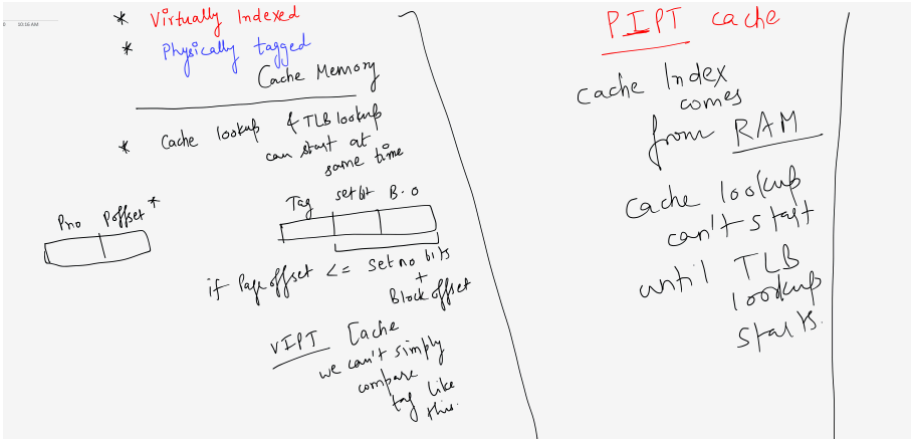
-

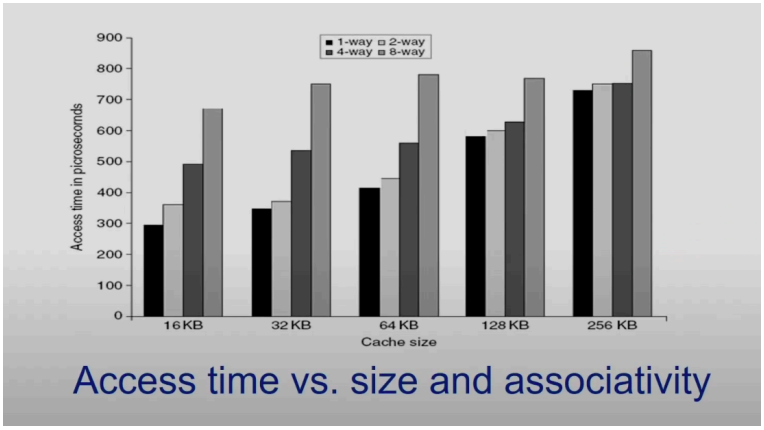
- 

- 

- CISC has a microprogrammed control unit
- RISC processors are pipelined to achieve average clocks per instruction around 1
- CISC processors are generally not highly pipelined.
- If speed increased by 20% this means time = number of instructions/speed i.e. 1/1.2 this means we will divide the time by 1.2. As time and speed are inversely proportional GATE IT 2004
- Maskable means which can be turned off, and vectored means which has a specific address.
- INTR is non vectored maskable interrupt signal
- 

- Hit latency is **Max(**Data array , **Tag Array+Hit/miss comp**) + data return time
- 

- 

- The direct mapped cache can overlap tag comparison and transmission of data. Data is kept ready, when we know its a hit, we extract
- In the k-way set associative, tag comparison and data transmission happened sequentially.
- Lower associativity reduces power
- When we work with small caches, set bits is less means indexing time is less
- Lower associativity tag comparison time + data comparison time we save but we increase conflicts, i.e. miss rate increases.
- 

-

Best answer

57

(B) is the answer. Absolute addressing mode means address of operand is given in the instruction.

option (A), operand is inside the instruction → immediate addressing

option (C), register containing the address in specified in operand → register Indirect addressing

option (D), the location of operand is implicit → implicit addressing

As the number of sets gets decreased, the number of possible cache block entries that a set maps to gets increased. So, we need more tag bits to identify the correct entry. Hence width of tag bit comparator increases.

If associativity is doubled, keeping the capacity and block size constant, then the number of sets gets halved. So, width of set index decoder can surely decrease.

Width of way-selection multiplexer must be increased as we have to double the ways to choose from.

Hence option D is the correct answer.

1. Cache memory
   Effective Memory Access Time = Cache access time * hit rate + miss rate * Miss penalty
   The above formula is too simple and given in many texts. But it hides what is exactly miss penalty. Because it depends on the implementation and there are simultenous cache look up and hierarchical. Unless otherwise stated in question always assume hierarchical access (case for write through cache explained below). So, for hierarchical cache, we can have the formula:
   Effective Memory Access Time = Cache access time * hit rate + (1 - hit rate) * (cache access time + main memory access time)
   = Cache access time + (1 - hit rate) * main memory access time
2. Multi level cache
   The above formula can be extended to $n + 1$ levels for n levels of caches and main memory
3. Write Through Cache
   This is applicable when question distinguishes between read and write accesses. When the cache is write through it means all writes going to main memory as well as to cache. Considering the best practical implementation, this should always be considered to be done in parallel and hence only the main memory write matters.
4. Memory access times can be given in unit of blocks/words and we must use the appropriate one as per the question. When a level 1 access from level 2, block size to be used is of level 1 and similarly for other levels.
5. In all the above cases we considered only cache and main memory. But if virtual memory is used, we have to consider TLB, page table access etc. TLB is like a cache for page table.
6. Cache access can be before TLB access - if cache is virtually indexed and virtually tagged - less common
7. Cache access is after TLB access - if cache is physically indexed and physically tagged - again less common as if TLB is missed we need a main memory lookup for page tables before coming to cache
8. Cache access is together with TLB acess - if cache is virtually indexed and physically tagged - most common. Here, cache indexing and TLB look-up happens simultaneoulsy and at time of tag comparison, we require physical address.

One merit of memory-mapped I/O is that, by discarding the extra complexity that port I/O brings, a CPU requires less internal logic and is thus cheaper, faster, easier to build, consumes less power and can be physically smaller; this follows the basic tenets of reduced instruction set computing, and is also

advantageous in embedded systems. The other advantage is that, because regular memory instructions are used to address devices, all of the CPU's addressing modes are available for the I/O as well as the memory, and instructions that perform an ALU operation directly on a memory operand (loading an operand from a memory location, storing the result to a memory location, or both) can be used with I/O device registers as well. In contrast, port-mapped I/O instructions are often very limited, often providing only for simple load-and-store operations between CPU registers and I/O ports, so that, for example, to add a constant to a port-mapped device register would require three instructions: read the port to a CPU register, add the constant to the CPU register, and write the result back to the port.

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

An instruction to transfer input/output data is executed only after the processor determines that the IO device is ready .

Processor waits for the IO device to send an interrupt request.

To transfer large block data at high speed(**limited by the bandwidth of bus**), we use DMA block transfer mode which does not require continuous interruption from CPU

A DMA interface has Status and control bits

Starting address register and word count register

Input status bit in an interface is cleared as soon as the input data buffer is read Why it is so important?

After reading the input data, it is necessary to clear the input status flag before the program begins a new read operation. Otherwise, the same input data would be read a second time.

. A subroutine is called by a program instruction to perform a function needed by the calling program. An interrupt-service routine is initiated by an event such as an input operation or a hardware error. The function it performs may not be at

all related to the program being executed at the time of interruption. Hence, it must not affect any of the data or status information relating to that program.

Average memory access time is always simultaneous as it is the time spent by the processor to read both the cache and ram parallely for reading the contents of the memory

Effectiveness of cache is based on property of computers which is known as locality of reference

Suppose you want to read some array element in the cache, which is a part of for loop, you will only read one element? No, you will need to read the entire array, it is always advised to load the next adjacent memory blocks which are spatially located near your array element. This is known as locality of reference.

Temporal locality is another side of the coin which says that if you load a block, will the same block again be accessed in near future(time).

Processor<---------->Cache<---------->RAM

Write Through

$$T_{read} = T_{cache} + (1 - H) \times T_{memory\_block} \qquad T_{write} = T_{memory\_word}$$

Write Back

$$T_{read} = T_{write} = H \times T_{cache} + (1 - H) \times (T_{memory\_block} + T_{write\_back}),$$
$$\text{where } T_{write\_back} = x \times T_{memory\_block},$$
$$\text{where } x \text{ is the fraction of dirty blocks}$$

So basically, in **write through** we read from cache and when there is a miss, we go to memory to read and while writing we take main memory time only as write is being done to both cache and RAM in parallel, so obviously Ram time is more, so total time is max(Cache, RAM) which is RAM time also known as Time to write a block to memory and cache in parallel.

And in **Write back**, it depends on the question, is it simultaneous or hierarchical access, generally we take hierarchical access by default, good thing here is that time taken in read is same as time taken in write as we have to do it in an order, go to cache, if there is a hit, read/write and if there is a miss, again, go to RAM with some miss penalty and read/write

Also it incurs some time taken to write back which is the fraction of dirty blocks times the main memory access time.

Direct mapping: Block j of RAM, maps onto block j modulo number of sets in the cache where each set has only one line.
Suppose RAM has 1024 blocks and Cache has 16 sets then we will map these 1024 blocks to the 16 sets in the cache.
PAS is 10 bits, Cache has 4 bits for set offset and suppose 4 bits is block offset then
PAS is divided like |2|4|4| where the first 2 bits are called as TAG this means, for a particular set in Cache, we have 4 possible candidates from the RAM, at a time, one of those candidates is present in the particular set in the Cache. This set is having 4 bits, this means we can specify the exact set number from 0000 to 1111 containing 16 unique set numbers, the last but not the least last 4 bits represents the block offset or the word offset where the actual data is present.

These lower-order 4 bits(word offset) correspond to the 16 unique words in a block
When a new block enters the cache, the 4 set bits, determines the cache set number where this block will be stored.
As the execution proceeds, the 4-bit set bit field of each address generated by the processor points to the particular block of the cache. Direct mapping is too easy to implement and the fastest mapping technique.

Associative Mapping: This gives you full freedom to put the RAM block to any set in the cache.
10 bits are now divided into only 6 Tag bits and 4 words offset bits |6|4|

The cost of this is a little higher than the direct map as we need to search all 64 tags to determine whether the given block is in the cache or not. All the 64 tags must be searched in parallel, so we need 64 comparators which are running in parallel.

Set Associative Mapping: Mixture of direct-mapped cache and associative cache. Here hardware cost is reduced as now we don't need to search those many tags as in the case of associative mapping. Now, we will be requiring few comparators. Here each set has k lines, that is why it is also known as k-way set associative cache.

Now suppose 10 bits PAS is divided like |5|3|2| Number of sets = $\frac{CS}{k \times BS}$ $\frac{64B}{2*4B} = 8sets$
This means 8 sets are there each set is having 2 lines as total we have 16 lines in the Cache
Now in these 8 sets we have 5 tags that means we have the 32 main memory blocks, at a time only 2 of those can be resident in the set as each set has 2 entries.

One more control bit i.e. the **valid bit** must be provided for each block. This bit tells whether the block contains valid data. **Dirty bit** tells whether the block is modified during its lifetime in the cache.

Whenever the Main memory block is updated by the DMA, a check is made to determine whether the block being loaded is currently in the cache or not. If DMA has updated the data in the RAM, the corresponding cache block valid bit is set to 0 to ensure that stale data is not present in the cache.

When DMA transfer is made from RAM to the disk, and if the cache uses the write back protocol. In this case, data in the RAM might not be the correct data, as the correct data is with the cached copy. One solution to this is to flush the cache by forcing the dirty data to be written back to the memory before DMA transfer from RAM to disk takes place, this ensures that correct data is getting transferred to the memory. This does not hamper performance as OS can handle this easily as such data transfers don't happen too often. LRU performs poorly when we access sequential elements of the array.

Sum = $x \oplus y \oplus c_{in}$ Carry = $xy + yc_{in} + xc_{in} = xy + c_{in}(x + y) = G_i + P_i c_i$

$c_{i+1} = G_i + P_i c_i = G_i + P_i(P_{i-1} c_{i-1})$

If $G_i = 1$, it implies that $c_{i+1} = 1$independent of input carry c_i

This is because of the boolean algebra, x+1 is always 1 as one of the input is 1, irrespective of the value of x, same is happening here also

- *Compulsory*—The very first access to a block *cannot* be in the cache, so the block must be brought into the cache. Compulsory misses are those that occur even if you had an infinite sized cache.
- *Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses (in addition to compulsory misses) will occur because of blocks being discarded and later retrieved.
- *Conflict*—If the block placement strategy is not fully associative, conflict misses (in addition to compulsory and capacity misses) will occur because a block may be discarded and later retrieved if multiple blocks map to its set and accesses to the different blocks are intermingled.

The efficiency is defined as the ratio of speedup to the number of processors. Efficiency measures the fraction of time for which a processor is usefully utilized.

$$E = \frac{S}{p} = \frac{T_S}{pT_p}$$

Cisc has more number of registers than Risc

Consider a reduced 7-bit IEEE floating-point format, with 3 bits for the exponent and 3 bits for the significand (Mantissa). What will be the value which get stored in the system corresponding to (−0.5625)?

A 1010001 Correct Option

Solution :
(a)

Here, Sign bit = 1
Also, 0.5625 = (0.1001)
For normalized form,
0.5625 = 1.001×2^{-1}
Hence, Actual Exponent = -1
Biased Exponent = Actual Exponent + Bias
Bias = Largest signed integer possible with 3 bits = 3
Biased Exponent = 3 – 1 = 2 (010)
Therefore, Binary representation will be (1010001).

DMA numerical GATE 2005-70

It is given the disk has 3000rpm.

3000 rotations in 60seconds.

Time taken for 1 rotation = 60/3000 seconds = 1/50 seconds

Each track has 512 sectors and each sector is 1KB in size. So each track size = 512KB.

In one rotation one track can be read. So in 1/50 seconds 512KB can be read.

It is given that one word is 4 bytes long. So time taken to read one word from the disk(4 bytes) = (4/512K) * (1/50) seconds ~= 160 ns

Here 160ns is the data preparation time. And time taken by the DMA to transfer the 4bytes data is one cycle = 40ns.

In cycle stealing mode, the CPU gets blocked for one clock cycle after the data is made ready by the I/O device in this case the disk.

So, % time the cpu gets blocked during DMA operation = 40ns / 160ns = 25%