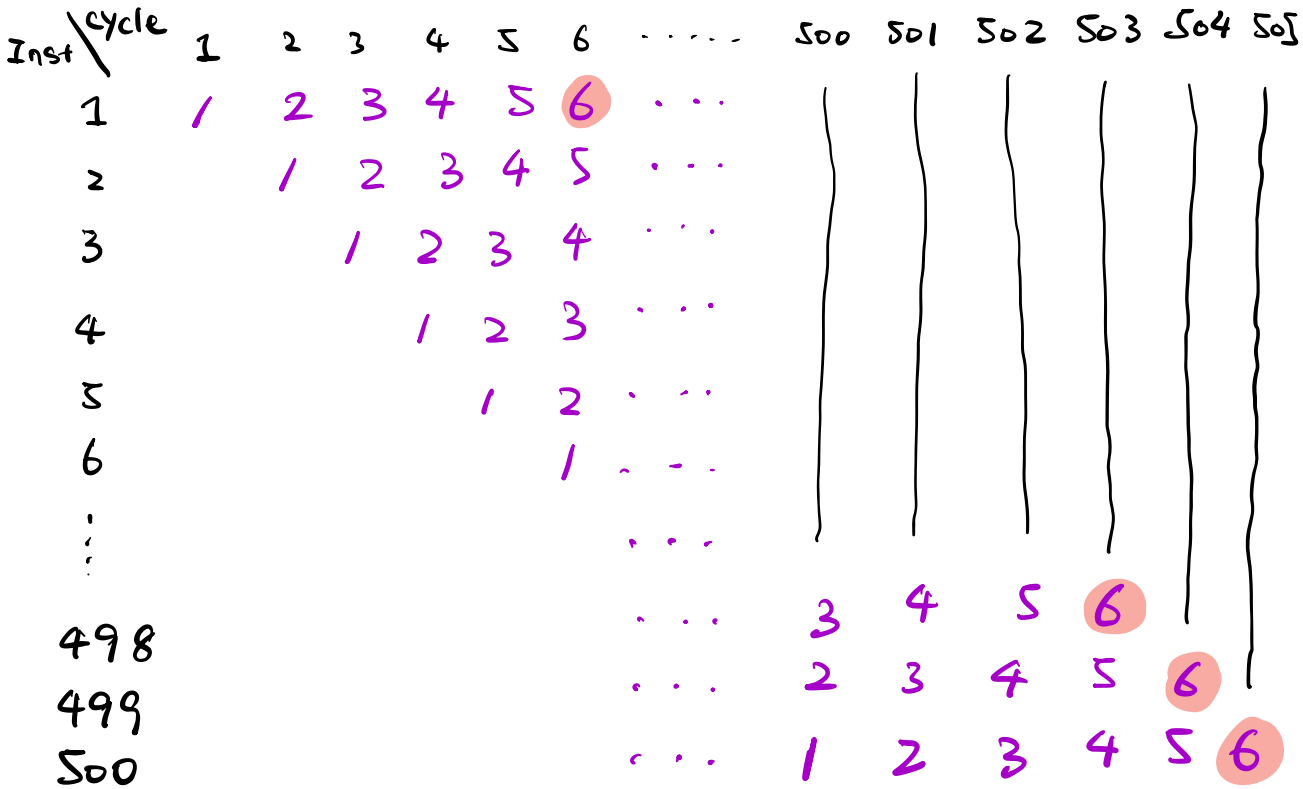
1.

(a)

We could draw the graph:



From the graph, we see we need 505 cycles to finish (500 cycles for the instruction #500 to start, and additional 5 cycles for it to finish)

(b) Total time = Cycles / Frequency

$= 505 / 2\,GHz$

$= 2.525 \cdot 10^{-7} s = 252.5\,ns$

2.

(a) the line A[i] += B[i] + B[i+1]

is actually A[i] = A[i] + B[i] + B[i+1]

So. for this single line. we need to have A[i], B[i]. B[i+1] in our instruction.

We could draw a table for the memory we need for each instruction: (Read directly from memory is marked purple, read from register is marked green)

inst #			
0	A[0]	B[0]	B[1]
1	A[1]	B[1]	B[2]
2	A[2]	B[2]	B[3]
⋮	⋮	⋮	⋮
≥55	A[255]	B[255]	B[256]

From the table. we could see that:

from memory: 256 x 2 + 1 = 513

from register: 255

Time it takes to read from memory:

513 x 100 ns = 51300 ns = 51.3 ms

(b) The hit ratio is:

$$\frac{255}{255+513} = \frac{255}{768} \approx 0.332 = 33.2\%$$

3. Again, we could draw a table

purple = read from memory
blue = read from cache
green = read from register

inst #	register			cache
0	A[0]	B[0]	B[1]	Cache A[0:3] Cache B[0:3]
1	A[1]	B[1]	B[2]	
2	A[2]	B[2]	B[3]	
3	A[3]	B[3]	B[4]	Cache B[4:]
4	A[4]	B[4]	B[5]	
⋮	⋮	⋮	⋮	
255	A[255]	B[255]	B[256]	cache B[256:259]

(this means the memory from base_ptr + 256 to base_ptr + 259, inclusive. B itself might not be that long)

From the table, we could see that;

from memory : 64 (from A) + 65 (from B) = 129

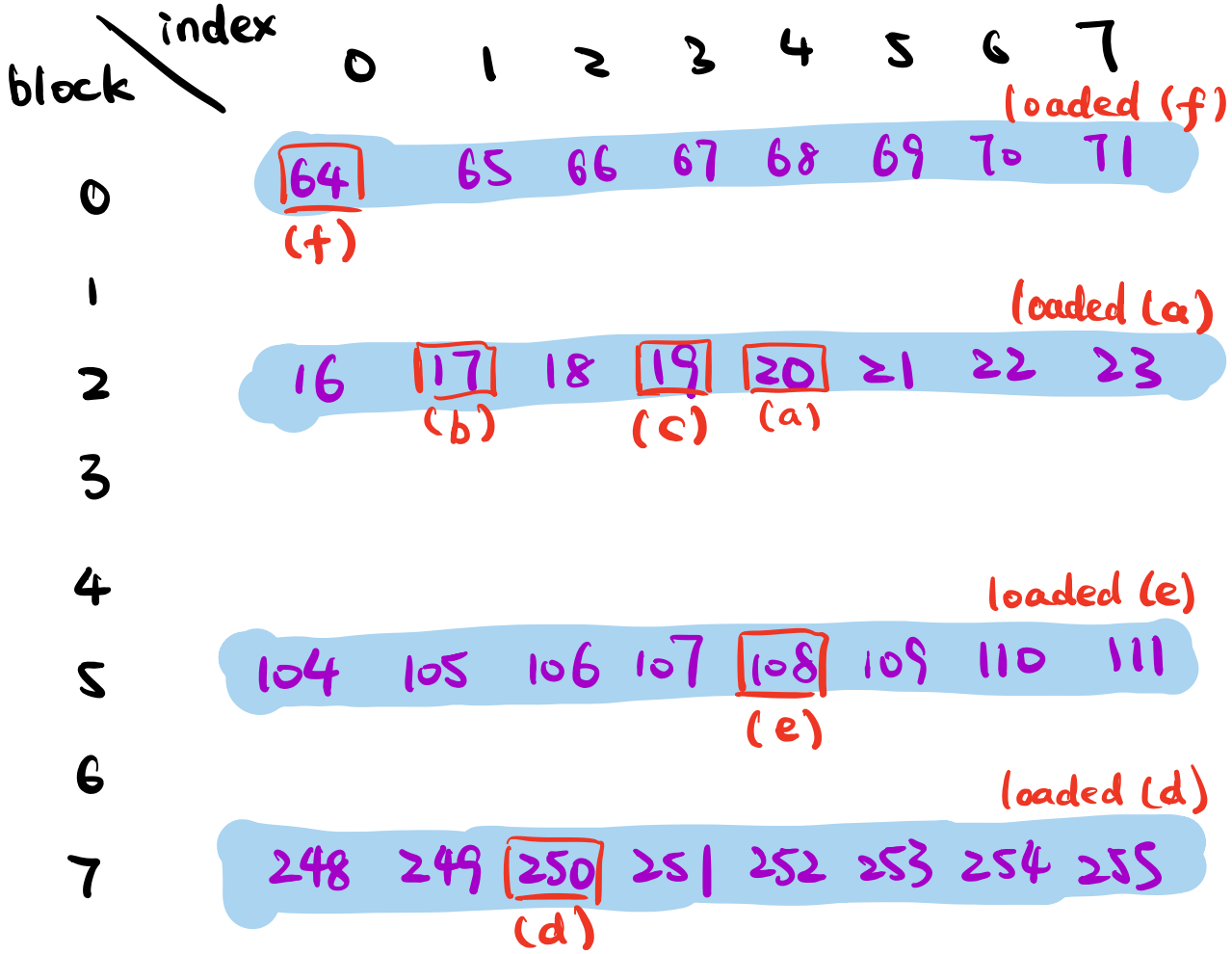
from cache : 192 (from A) + 192 (from B) = 384

from register : 255

Thus, time takes:

$t = 129 \times 100ns + 384ns \times 2ns$ = 13668 ns = 13.668 ms

4. we could draw a table



(a) line : $\lfloor 20/8 \rfloor = 2$

block : $2 \% 8 = 2$

since there is no line currently loaded. This is a cold miss. Load cache line #2 to block 2.

(b) line : $\lfloor 17/8 \rfloor = 2$

block : $2 \% 8 = 2$

The line #2 is already in cache (block 2).

this is a hit, read from the block 2

(c) line: $\lfloor 19/8 \rfloor = 2$

block: $2 \% 8 = 2$

The line #2 is already in cache (block 2).

this is a hit, read from the block 2

(d) line: $\lfloor 250/8 \rfloor = 31$

block: $31 \% 8 = 7$

Line #31 currently is not loaded in cache. This is a cold miss. Load cache line #31 to block 7.

(e) line: $\lfloor 108/8 \rfloor = 13$

block: $13 \% 8 = 5$

Line #13 currently is not loaded in cache. This is a cold miss. Load cache line #13 to block 5.

(f) line: $\lfloor 64/8 \rfloor = 8$

block: $8 \% 8 = 0$

Line #8 currently is not loaded in cache. This is a cold miss. Load cache line #8 to block 0.

5.

(1) False. It's not always useful. Consider a cache line of size 4. If CPU asks data on address 0, 4, 8, 12... It still misses everytime, and need to go to the memory. In this case, the caching hardware need to do extra works (load cache) that has no benefits.

(2) True. If the program has spatial/temporal locality. Prefetching could reduce compulsory misses. Yet, it's not guaranteed to reduce such misses if the program's memory access doesn't have certain patterns (say the program randomly accesses the memory, or program traverses the matrix in column major).

(3) True

(4) False. In the Q1, we see it's not N·m steps for N independent operation. The steps (or cycles) is actually N+(M-1).

(5) False. The pipeline is designed to speed up multiple instructions. For single instruction,

It stills requires same cycles. Say for an instruction that takes M cycles to execute, if you split the instruction into M substeps each of which takes 1 cycle to finish, you still need $1+(M-1) = M$ cycles to execute, and in real situations, implementing the pipeline might make each substep slower. So, pipeline could not make single instruction faster, it speeds up multiple instruction, though.