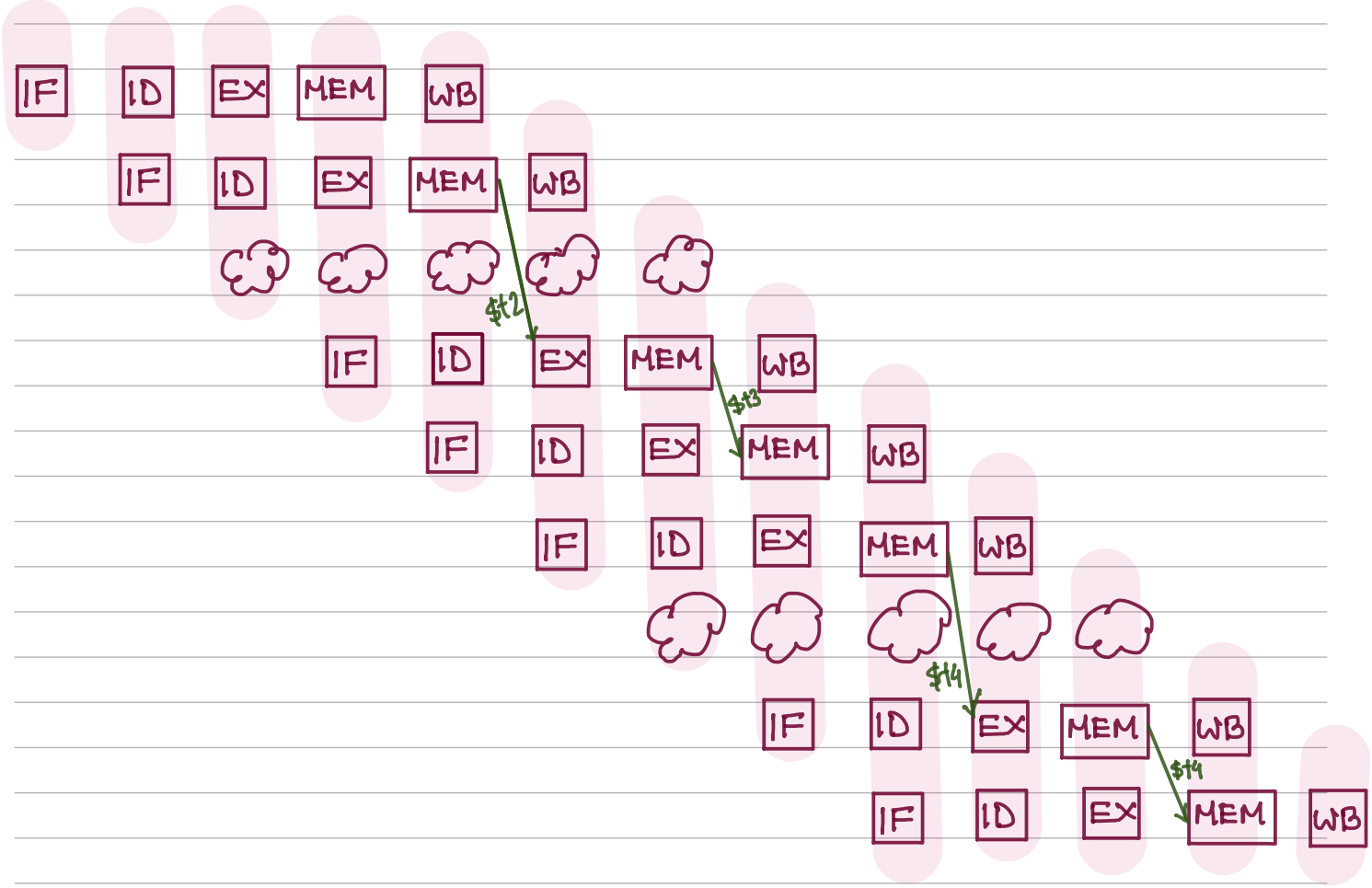
2019 – 2020

5. (a) Assume pipelined execution for the following sequence of instructions. The corresponding C code is A = B + C; D = A + D. **(15)**

```
lw $t1, 0($t0) // load B
lw $t2, 4($t0) // load C
add $t3, $t1, $t2 // add B & C
sw $t3, 12($t0) // store A
lw $t4, 8($t0) // load D
add $t4, $t3, $t4 // add A & D
sw $t4, 8($t0) // store D
```

Draw the execution pipeline for the given sequence of instructions and clearly show the stalls and the forwarding paths. Note that data forwarding is available if needed.



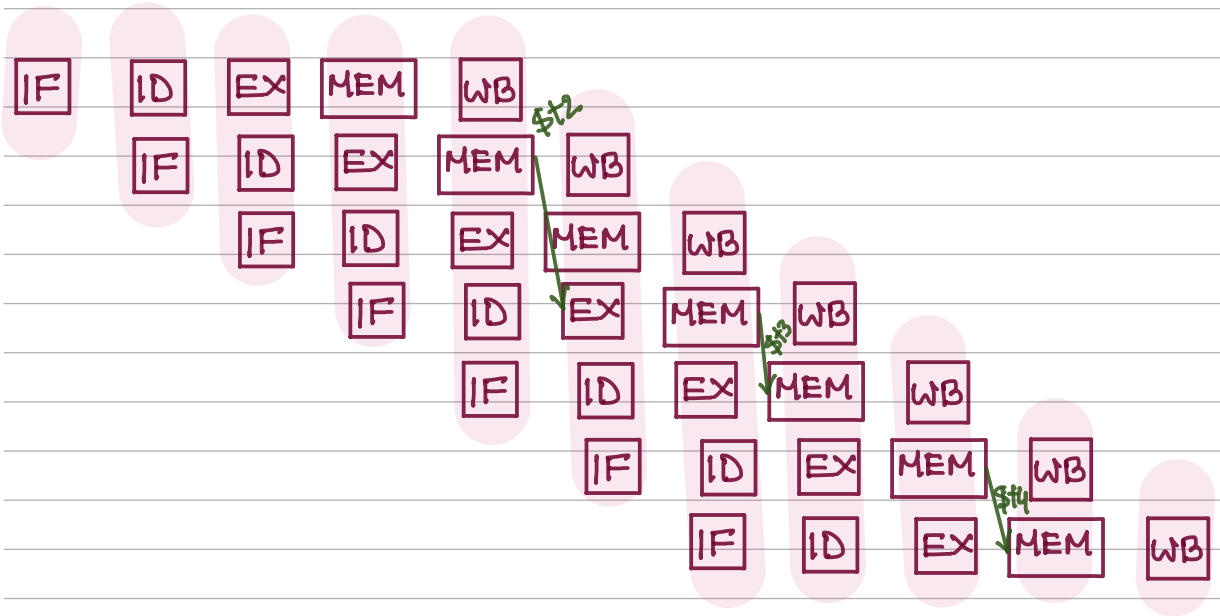

(b) How many stalls and clock cycles are required in question 5(a)? **(5)**

13 clock cycles needed
2 stalls needed

(c) Is it possible to rearrange the code in question 5(a) to reduce the required number of stalls? If not, then explain why it is not possible. If yes, then write down the rearranged code block and indicate the number of stalls and the total number of clock cycles required after the rearrangement. **(15)**

We can rearrange like this →

```
lw $t1, 0($t0)
lw $t2, 4($t0)
lw $t4, 8($t4)
add $t3, $t1, $t2
sw $t3, 12($t0)
add $t4, $t3, $t4
sw $t4, 8($t0)
```


no stalls needed here
total 11 clock cycles needed

6. (a) Assume there are *three* small caches, each consisting of eight (8) one-word blocks. One cache is **direct-mapped**, a second is **two-way set-associative**, and the third is **four-way set-associative**. For each cache organization, complete the following table for the following sequence of memory block addresses: **0**, **12**, **16**, **18**, and **0**. You are allowed to subdivide the third column "Contents of cache blocks after reference" as needed. **(20)**

Note that when it is necessary to replace an existing cache entry on a cache miss, follow the least recently used (LRU) replacement rule to determine which cache entry to replace and show the cache content after the replacement.

Direct mapped:

| Address of memory block accessed | Hit or Miss | Contents of cache blocks after reference |
|---|---|---|
| 0 | miss | 0 |
| 12 | miss | 0, 12 |
| 16 | miss | 16, 12 |
| 18 | miss | 16, 18, 12 |
| 0 | miss | 0, 18, 12 |

0 1 2 3 4 5 6 7

Two-way set associative:

| Address of memory block accessed | Hit or Miss | Contents of cache blocks after reference |
|---|---|---|
| 0 | miss | 0 |
| 12 | miss | 12 0 |
| 16 | miss | 16 12 |
| 18 | miss | 16 12, 18 |
| 0 | miss | 0 16, 18 |

0 1 2 3

Four-way set associative :

| Address of memory block accessed | Hit or Miss | Contents of cache blocks after reference | |
|---|---|---|---|
| 0 | miss | 0 | |
| 12 | miss | 12 0 | |
| 16 | miss | 16 12 0 | |
| 18 | miss | 18 16 12 0 | |
| 0 | hit | 0 18 16 12 | |
| | | 0 | 1 |

(b) Assuming a cache of 4K blocks, a 4-word block size, and a 32-bit address, find the *total* number of tag bits for a cache that is **eight-way set associative**. **(15)**

Block size = 4-word
# of rows = $2^{12}$

$n = 12 - 3 = 9$
$m = 2$

$t = 32 - (n + m + 2)$
$= 32 - (9 + 2 + 2)$
$= 19$ bit

total tag bits = $19 \times 2^{12} = 77824$

7. (a) How many *total* bits are required for a **direct-mapped** cache with 64 KB of data and 8-word blocks (i.e., 32 bytes), assuming a 32-bit address? **(20)**

block size = 8-word = 32 byte
# of rows = $2^{16}/2^5 = 2^{11}$

$m = 3$
$n = 11$
$b = 32 \times 8$

$t = 32 - (n + m + 2)$
$= 16$

total bits required = $2^n \times (1 + \text{tag-bit} + \text{index bit})$
$= 2^{11} \times (b + t + v)$ [v = 1]
$= 559104$ bits

(b) The TLB acts as a cache of the page table for the entries that map to physical pages only as shown in Figure 7b. Write down the purpose of the tag bits in TLB. **(5)**

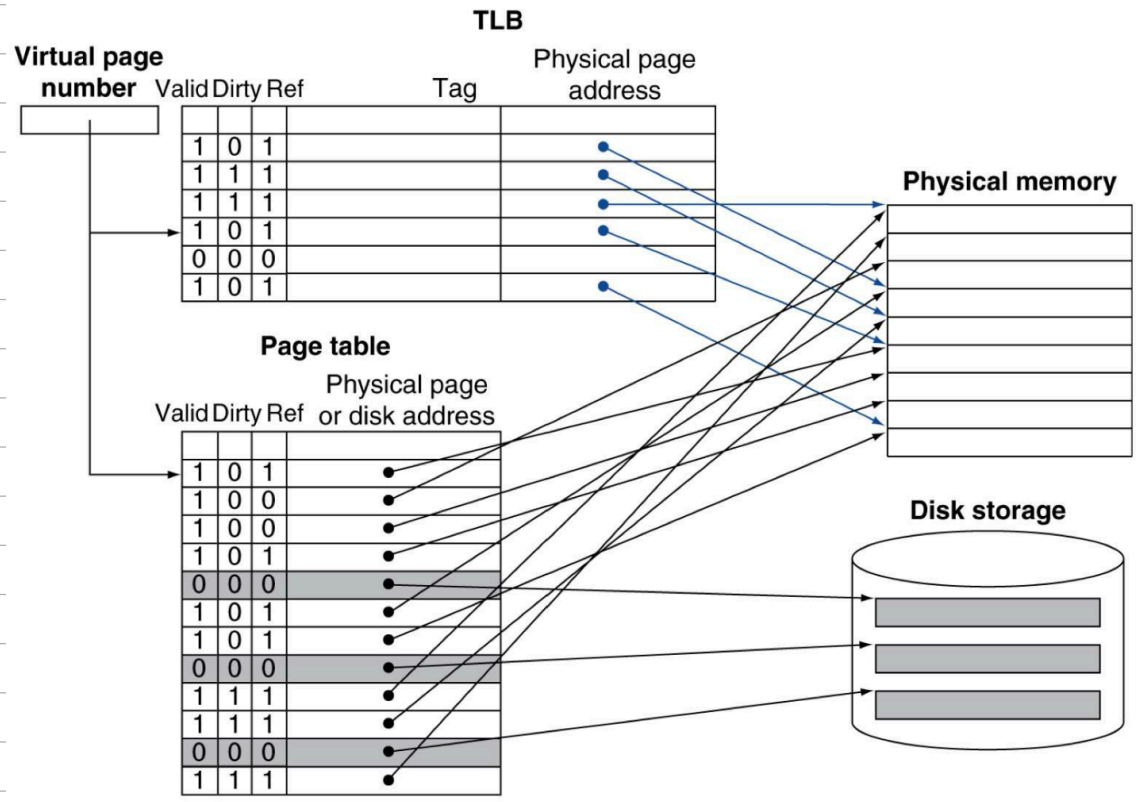



Figure 7b: TLB acts as a cache of the page table

Unlike the page table TLB is a cache.
Because TLB is a cache, it must have tag bits.
The tag bits of each entry is the virtual page number for that TLB entry.

(c) Explain how a TLB miss is handled. **(10)**

TLB miss indicates →
1. Page present, but PTE not in TLB
2. Page not present (page fault)

Case 1:
Load the PTE from memory and retry.
Case 2:
OS handles fetching the page and updating the page table. Then restart the faulting instruction.

8. (a) Figure 8a shows a single cycle datapath implementation of a subset of MIPS instruction set. For each of the following three instructions: **add**, **sw**, **beq**, determine the values (0/1) of the following control signals shown in Figure 8a. **(5x3=15)**

RegDst
Branch
MemRead
MemtoReg
MemWrite
ALUSrc
RegWrite

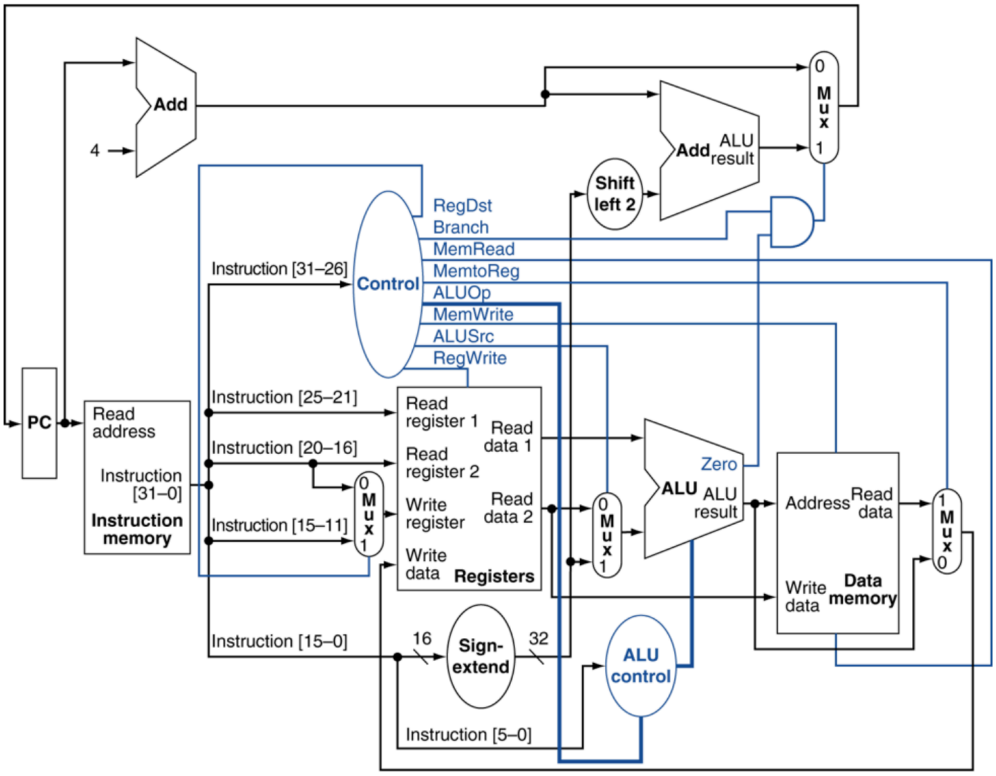
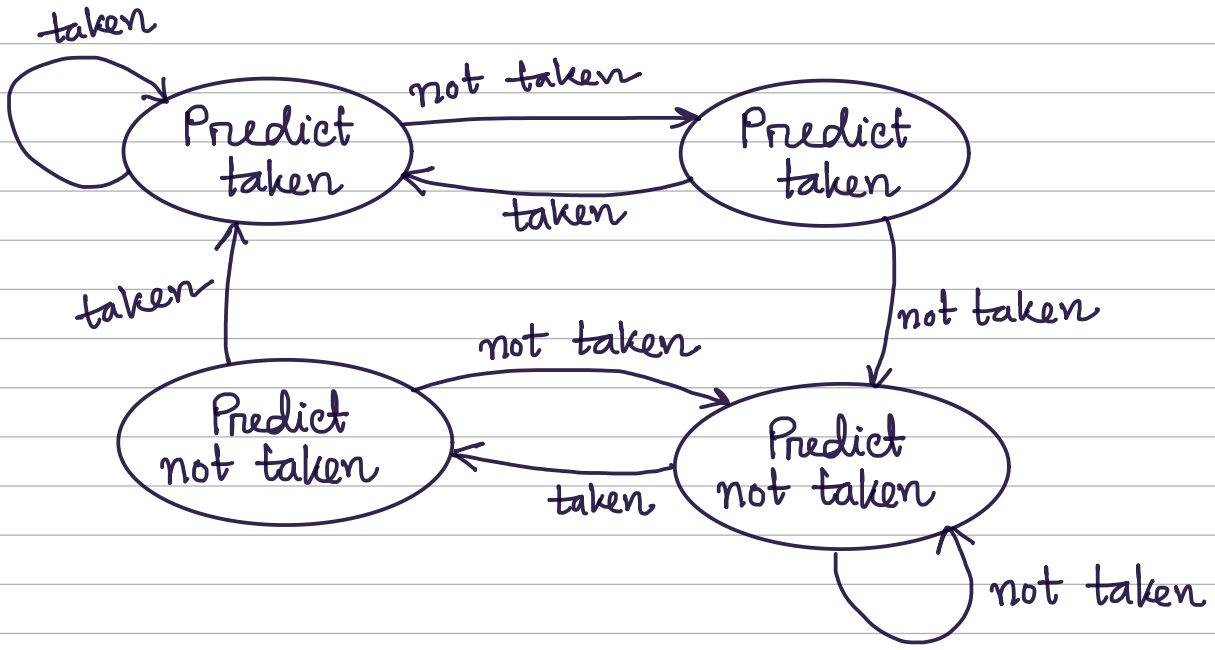



Figure 8a: Single cycle datapath with control signals

| | add | sw | beq |
|---|---|---|---|
| RegDst | 1 | X | X |
| Branch | 0 | 0 | 1 |
| MemRead | 0 | 0 | 0 |
| MemtoReg | 0 | X | X |
| MemWrite | 0 | 1 | 0 |
| ALUSrc | 0 | 1 | 0 |
| RegWrite | 1 | 0 | 0 |

(b) Draw the finite state machine for a 2-bit dynamic branch prediction scheme. **(8)**




(c) Figure 8c shows a pipelined execution of three instructions **sub**, **and**, and **or**. The pipeline registers are denoted respectively as IF/ID, ID/EX, EX/MEM, and MEM/WB. The data forwarding path between **sub** and **and** has been shown in *green* color and the data forwarding path between **sub** and **or** has been shown in *yellow* color. For each of these two data forwarding cases, write down the data forwarding conditions. **(6x2=12)**

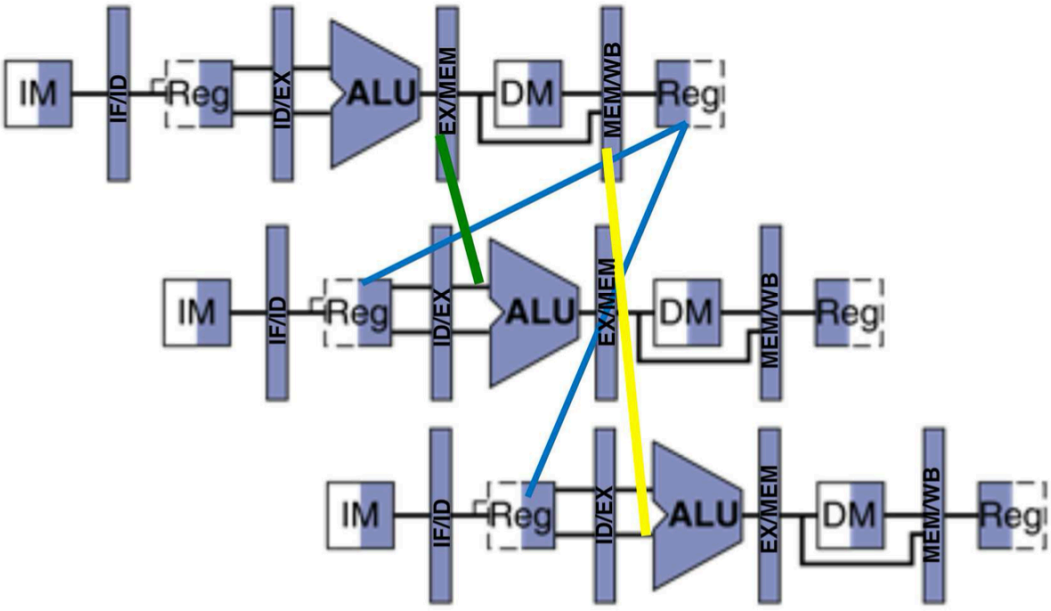



Figure 8c: Pipelined execution with data forwarding

EX hazard: (mux selector = 10)

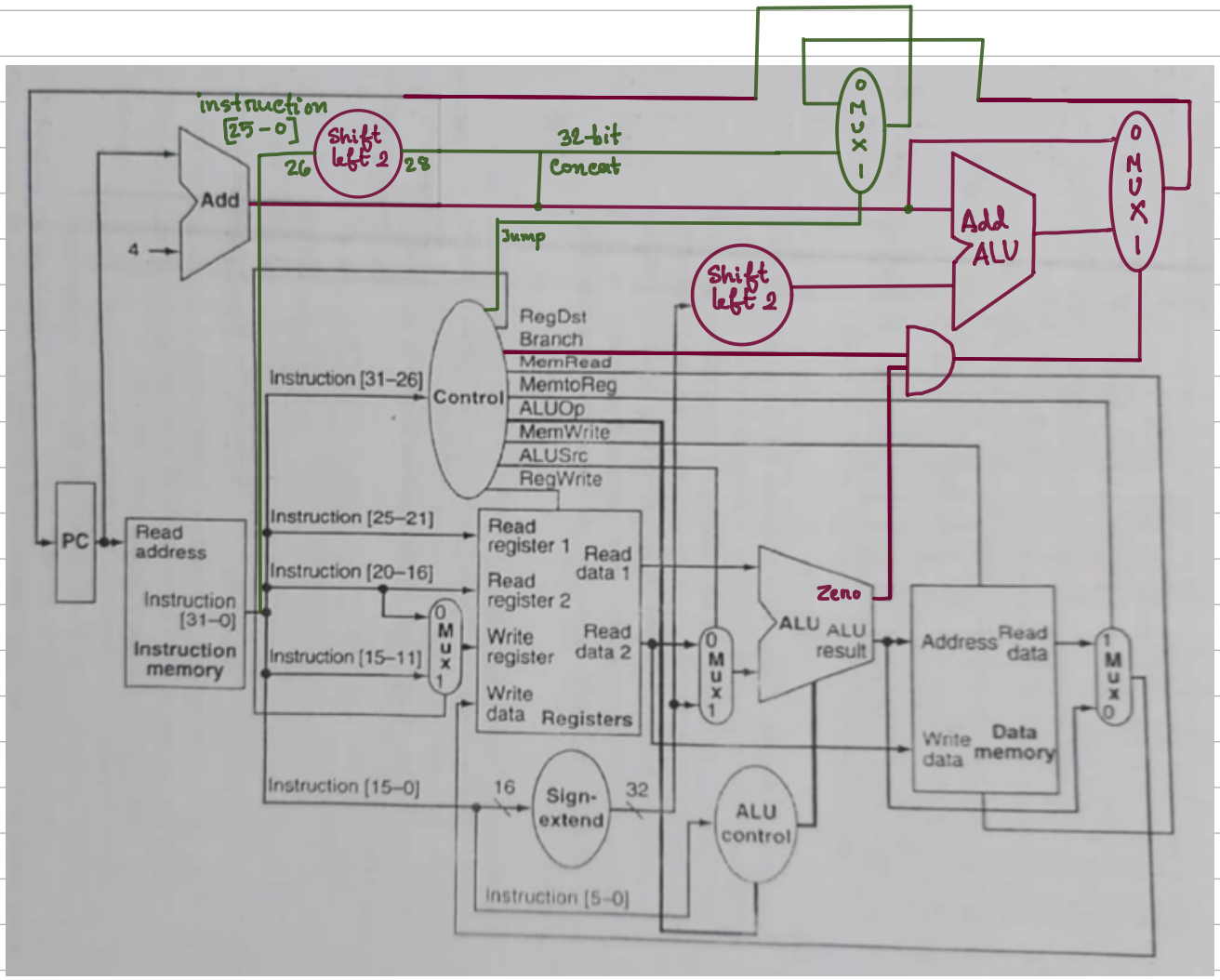
1. ForwardA = 10 ⟶
   EX/MEM.regWrite & EX/MEM.Rd ≠ 0 & EX/MEM.Rd = ID/EX.Rs
2. ForwardB = 10 ⟶
   EX/MEM.regWrite & EX/MEM.Rd ≠ 0 & EX/MEM.Rd = ID/EX.Rt

MEM hazard: (mux selector = 01)

1. ForwardA = 01 ⟶
   MEM/WB.regWrite & MEM/WB.Rd ≠ 0 &
   !(EX/MEM.regWrite & EX/MEM.Rd ≠ 0 & EX/MEM.Rd = ID/EX.Rs) &
   MEM/WB.Rd = ID/EX.Rs
2. ForwardB = 01 ⟶
   MEM/WB.regWrite & MEM/WB.Rd ≠ 0 &
   !(EX/MEM.regWrite & EX/MEM.Rd ≠ 0 & EX/MEM.Rd = ID/EX.Rt) &
   MEM/WB.Rd = ID/EX.Rt

2021-2022

1. (a) Complete the provided incomplete diagram of the MIPS single cycle datapath for the branch if equal (beq) and jump (j) instructions. **(10)**

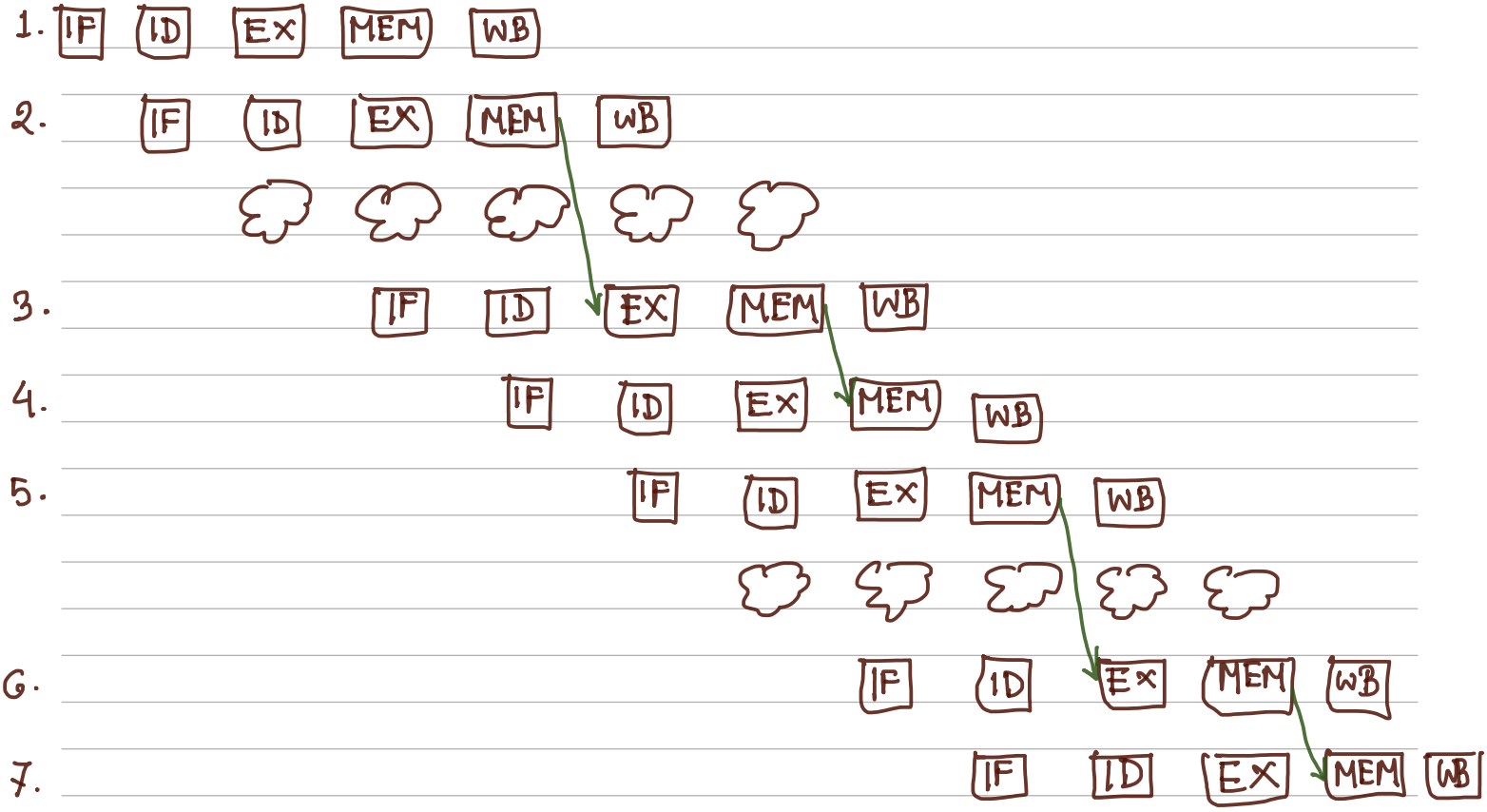



(b) Consider the following MIPS code: **(10)**

1. lw $t1, 0($t0)
2. lw $t2, 4($t0)
3. add $t3, $t1, $t2
4. sw $t3, 12($t0)
5. lw $t4, 8($t0)
6. add $t5, $t1, $t4
7. sw $t5, 16($t0)

[Slide example]

How many cycles are required to execute the above code? Can you reduce the number of cycles using code scheduling? If yes, what will be resulting number of cycles?




here, needed 13 clock cycles

by code scheduling it can be reduced.

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
lw  $t4, 8($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
add $t5, $t1, $t4
sw  $t5, 16($t0)
```

no stall needed for this

no stall needed

hence we can reduce 2 stalls.

So, 11 clock cycles will be needed -

(c) What do you mean by data hazard for branches? Give MIPS code examples with explanations for each of the following cases of data hazard for branches: (10)

(i) Needs no stall to resolve

(ii) Needs one cycle stall to resolve

(iii) Needs two cycle stalls to resolve

Data hazard for branches occurs when there is a dependency between the data produced by another instruction earlier and the data needed to predict branch in beq.

① No stall:

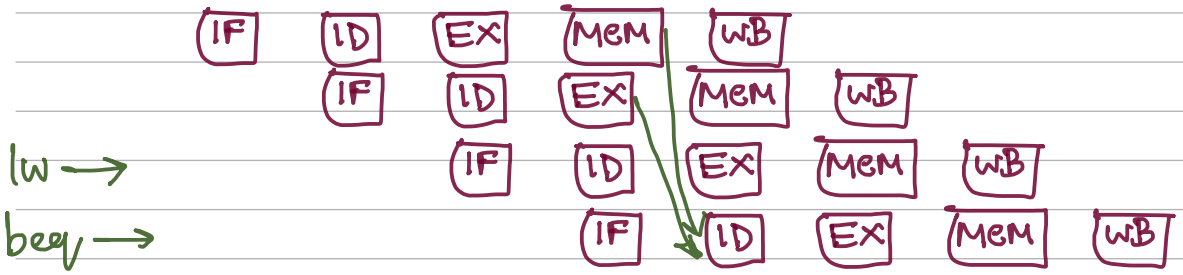
Suppose the instruction set:

```
lw   $t6, 0($t0)
add  $t1, $zero, $t2
or   $t3, $t4, $t5
beq  $t1, $t3, 8
```

data hazard

here we can re-schedule the instruction set →

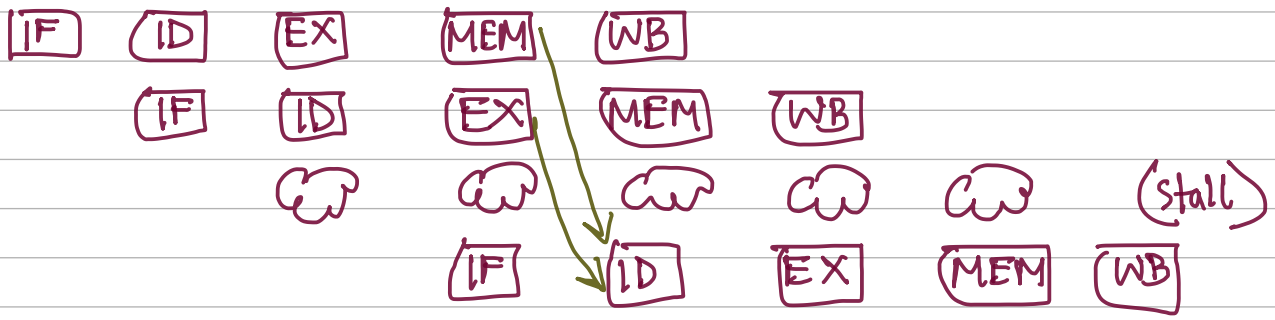
```
add  $t1, $zero, $t2
or   $t3, $t4, $t5
lw   $t6, 0($t0)   ——→ not-dependent
beq  $t1, $t3, 8
```




② One cycle stall:

```
add  $t1, $zero, $t2
or   $t3, $t4, $t5
beq  $t1, $t3, 8
```

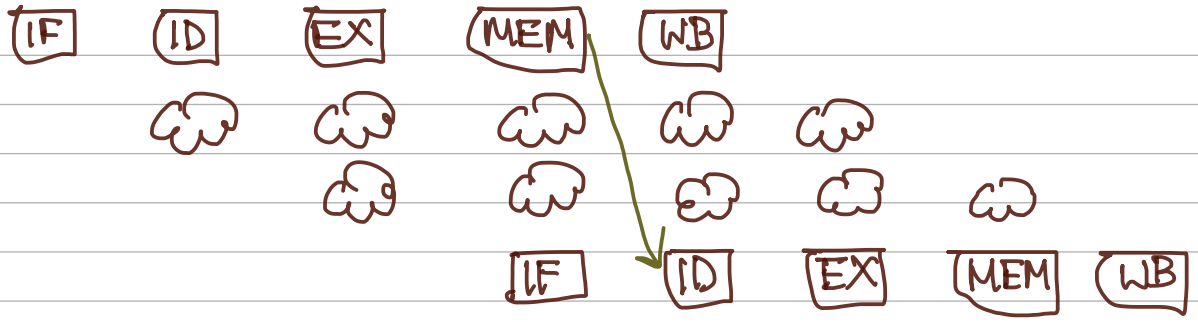
here, we can't find a non-dependent instruction for re-scheduling. Hence, need one stall.




③ two stalls needed:

```
lw  $t1, 0($t0)
beq $t1, $t2, 4
```

here the $t1 value we can get from lw instruction in MEM stage. Hence two stalls needed.




(d) Consider the following code: (5)

```
outer:
        // some code
inner:
        // some code
        beq $t0, $zero, inner
        // some code
        beq $t1, $zero, outer
```

What is the problem with the above code if we use 1 bit branch predictor? How can you solve that?

This is similar to a nested loop code snippet. the inner loop continues to branch as long as $to is zero. When this inner branch is not taken the branch predictor is set to 0 (this is the case when it gets out of the inner loop and then the outer branch instruction gets checked). Hence, every time the outer beq. instruction is checked the branch predicton is set to 0. So. we get misprediction for this branch in all the cases except the last one, when outer loop terminates.
But if we use 2-bit predictor, after terminating the inner loop, the branch predictor is still set to predict taken. Hence, it predicts the outer branch to be taken. When both of the loop terminates finally the 2-bit branch predictor is set to predict not-taken according to the last two branch result. Because 2-bit predictor only changes prediction on two successive wrong predictions.

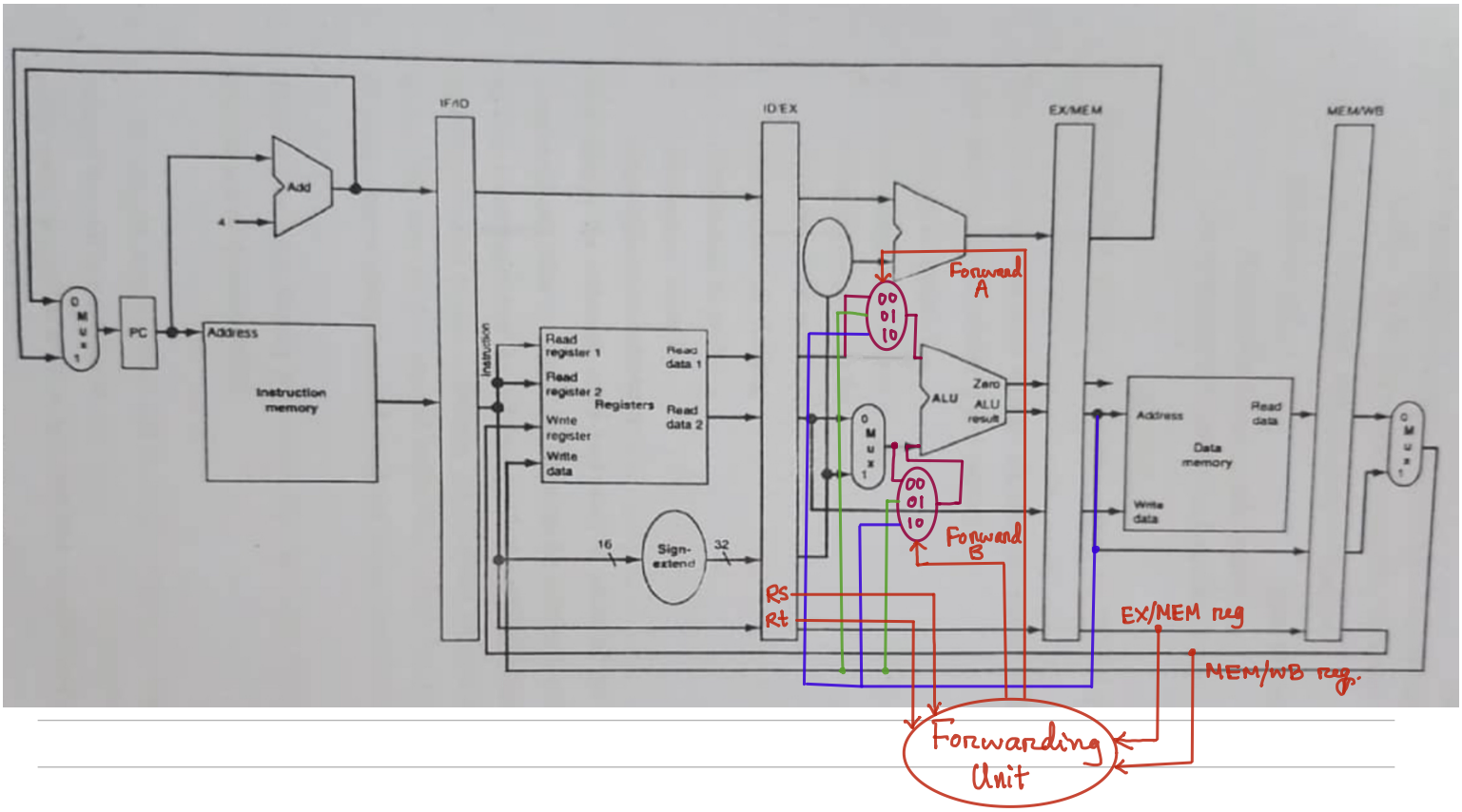
2. (a) What do you mean by forwarding for data hazard? Design a complete forwarding unit in the provided diagram with necessary equations for the MIPS pipeline considering double data hazard. (15)

To avoid data hazard we use forwarding, which means using the result when it is computed, not waiting the data to be stored in the destination.
Forwarding requires extra hardware and connections in datapath.

Double data hazard occurs when both MEM hazard and EX hazard happens.
For example:

MEM hazard
```
add $t1, $t0, $t2
add $t1, $t1, $t2   } EX hazard
add $t1, $t1, $t3
```

if we use $t1's value from first MEM hazard that will be wrong.




In the forwarding unit we need to check if EX hazard occured or not.
Hence, the conditions will be →

For EX hazard:

Case 1: ForwardA = 10

EX/MEM.RegWrite &
EX/MEM.Rd ≠ 0 &
EX/MEM.Rd = ID/EX.RS

Case 2: ForwardB = 10

EX/MEM.RegWrite &
EX/MEM.Rd ≠ 0 &
EX/MEM.Rd = ID/EX.Rt

For MEM hazard:

Case 1: ForwardA = 01

MEM/WB.regWrite & MEM/WB.Rd ≠ 0 &
! (EX/MEM.RegWrite & EX/MEM.Rd ≠ 0 & EX/MEM.Rd = ID/EX.RS) &
MEM/WB.Rd = ID/EX.RS

Case 2: ForwardB = 01

MEM/WB.regWrite & MEM/WB.Rd ≠ 0 &
! (EX/MEM.RegWrite & EX/MEM.Rd ≠ 0 & EX/MEM.Rd = ID/EX.Rt) &
MEM/WB.Rd = ID/EX.Rt

*b, c not in syllabus

3. (a) Suppose we have a processor with a base CPI of 1.0, assuming all references hit in the primary cache and a clock rate of 4 GHz. Assume a main memory access time of 200 ns, including all the miss handling. Suppose the miss rate per instruction at the primary cache is 4%. How much faster will the processor be if we add a second level cache that has a 10 ns access time for either a hit or a miss and is large enough to reduce the miss rate to main memory to 1%? How much faster will the processor be if we add a third level cache that has a 40 ns access time for either a hit or a miss and is large enough to reduce the miss rate to main memory to 0.5%? (10)

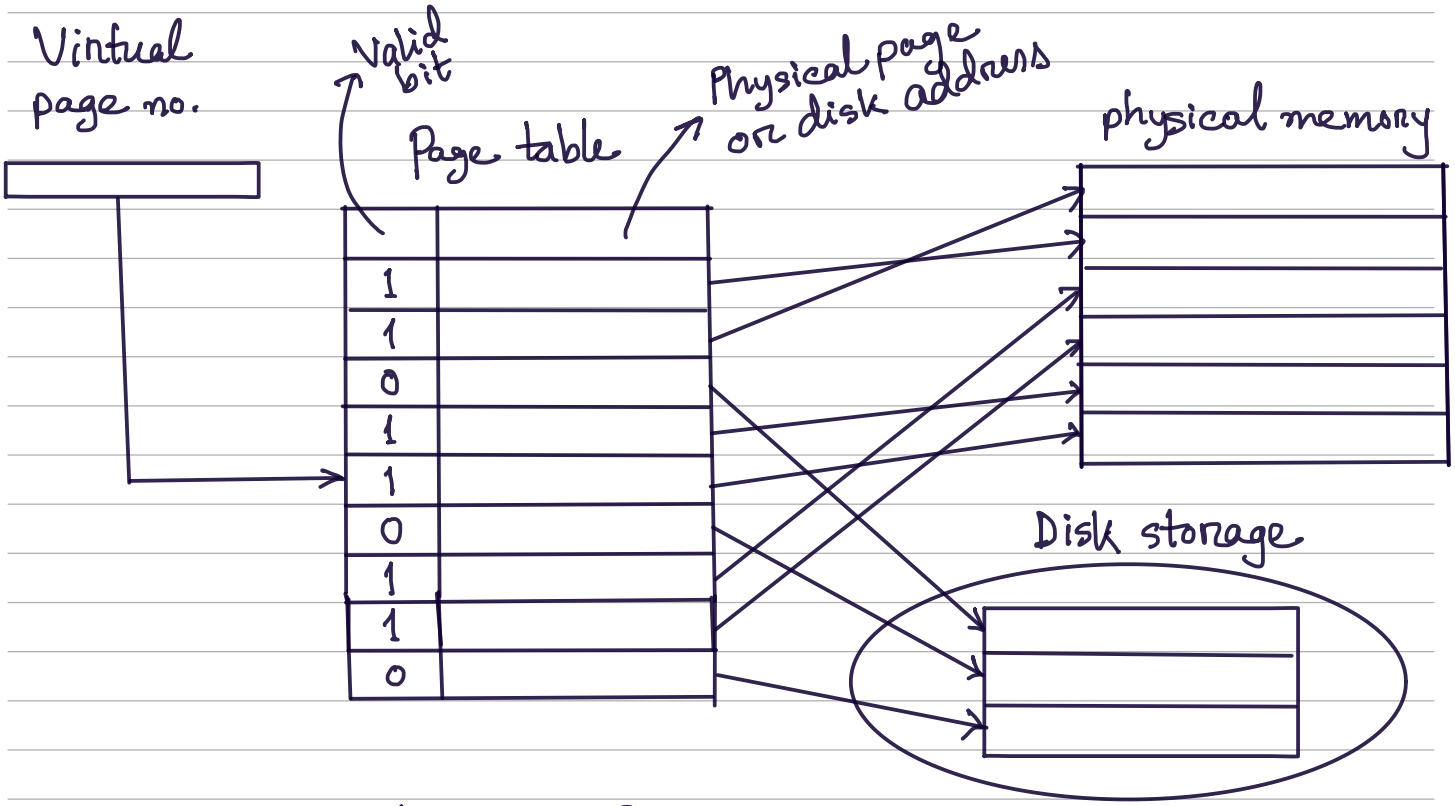
Miss penalty to main memory = 200 × 4 = 800 clock cycles
Total CPI = 1 + 0.04 × 800 = 33

After adding second level cache →
miss penalty = 10 × 4 = 40 clock cycles
Total CPI = 1 + 0.04 × 40 + 0.01 × 800 = 10.6
faster by = $\frac{33}{10.6}$ = 3.113

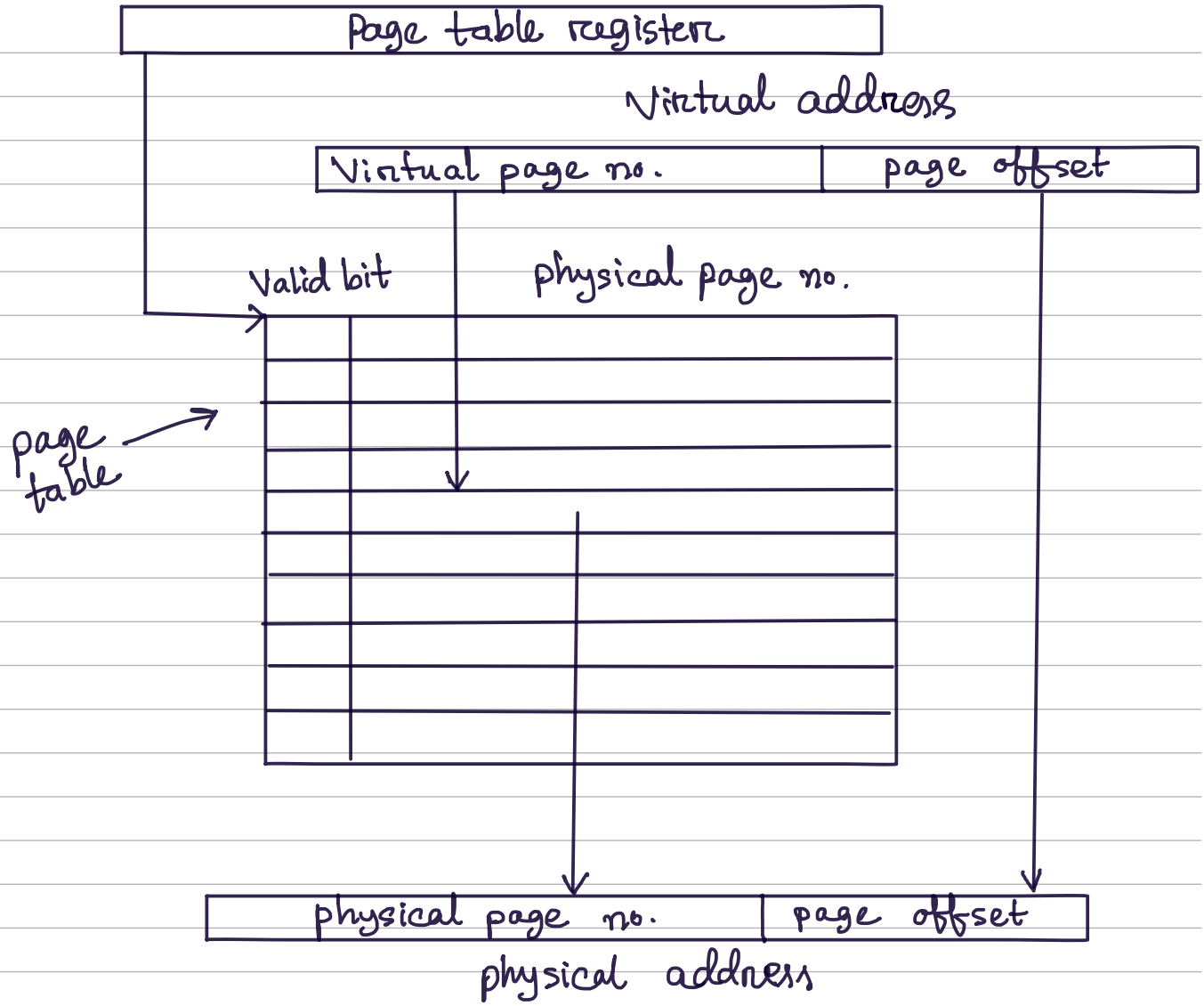
After adding third level cache →
miss penalty = 40 × 4 = 160 clock cycles
Total CPI = 1 + 0.005 × 800 + 0.01 × 40 + 0.04 × 160 = 11.8
faster by = $\frac{33}{11.8}$ = 2.797

(b) Explain with necessary diagrams how the page table works with respect to the virtual address, physical address, main memory, and disk. (10)

Page tables stores placement information.
It is an array of page table entries (PTE), indexed by virtual page number.
Page table register in CPU points to page table in physical memory.
If page is present in memory PTE stores the physical page number and other status bits.
If page is not present, PTE can refer to location in swap space on the disk.




Mapping Pages to Storage




(c) You are given the following hit/miss status of TLB, Page Table and Cache. (10)

| Serial No. | TLB | Page Table | Cache |
|---|---|---|---|
| 1 | Hit | Hit | Hit |
| 2 | Hit | Miss | Hit |
| 3 | Miss | Miss | Hit |
| 4 | Miss | Miss | Miss |
| 5 | Miss | Hit | Hit |

Table 3(c)

Explain which of the scenarios in Table 3(c) is possible or impossible and under what circumstance with proper reasoning.

1. Possible. Although the page table is never really checked if TLB hits
2. Impossible. Cannot have translation in TLB if not present in memory.
3. Impossible. Data cannot be in cache if not present in memory
4. TLB misses followed by a page fault. Data must miss the cache after retry.
5. TLB misses but data found in page table. After retry it is found in the cache.

* d not in syllabus

4. (a) Assume there are three small caches, each consisting of four one-word blocks and uses LRU replacement policy. One cache is fully associative, a second is two-way set-associative, and the third is direct-mapped. Find the number of misses for each cache organization given the following sequence of block addresses: (15)

1, 3, 2, 4, 5, 1, 3, 1, 3, 5, 3, 2

Case 1: fully associative:

# of rows = $4 = 2^2$

block size = 1 word = 4 Byte

No index needed, cache block can go anywhere in the cache

| tag | data |
|---|---|
| ~~1~~ 5 | |
| ~~3~~ 1 | |
| ~~2~~ 3 | |
| ~~4~~ 2 | |

| 1 | 3 | 2 | 4 | 5 | 1 | 3 | 1 | 3 | 5 | 3 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| m | m | m | m | m | m | m | h | h | h | h | m |

# of misses = 8

Case 2: two-way associative:

# of rows = $2^2$

block size = 1 word

index = 1

# of sets = 2

n = 1

| | tag | data |
|---|---|---|
| Set 0 | 2 | |
| | 4 | |
| Set 1 | ~~1~~ ~~3~~ 3 | |
| | ~~2~~ ~~1~~ 5 | |

| 1 | 3 | 2 | 4 | 5 | 1 | 3 | 1 | 3 | 5 | 3 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| m | m | m | m | m | m | m | h | h | m | h | h |

# of misses = 8

Case 3: direct mapped:

# of sets = 4, n = 2 index = 2

| | tag | data |
|---|---|---|
| 00 | 4 | |
| 01 | ~~1~~ ~~3~~ ~~1~~ 5 | |
| 10 | 2 | |
| 11 | 3 | |

| 1 | 3 | 2 | 4 | 5 | 1 | 3 | 1 | 3 | 5 | 3 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| m | m | m | m | m | m | h | h | h | m | h | h |

# of misses = 7

* IDK if b, c in syllabus or not ☺