Exercises on Pipelining, Data Hazards & Memory H

1. Review Problems for Quiz 2
2. Quiz 2: Friday November 18th :
   - INET section: 8:30 AM – 11:00AM
   - Section A,B: During Class Hours
   - MOSES students – check with your Proctor at MOSES
3. 8-10 Questions covering topics from HW 5,6
4. Same Format, Rules as Quiz 1

Highlights of Exercises on Data Hazards

1. How do we determine program execution time penalty from `Data Hazards` – with *and* without forwarding hardware overheads
2. How do structural hazards from use of a single Memory Array increase CPI & how is this Structural Hazard resolved by instruction scheduling?
3. Lowering CPI penalty due to Load/Store Data Use Hazards by overlapping EX and MEM stages of the RISC pipeline
4. Load-use-data hazard resolution for given instruction sequences including branches

1. Speedup from Forwarding hardware resolving data hazards

Consider a version of the pipeline from *Section 4.5 in RISC-V text* that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). Suppose that (after optimization) a typical n- instruction program requires an additional **0.**4*n NOP instructions to correctly handle data hazards.

1.1 Suppose that the cycle time of this pipeline without forwarding is 250 ps. Suppose also that adding forwarding hardware will reduce the number of NOPs from .4*n to .05*n, but increase the cycle time to 300 ps. What is the speedup of this new pipeline compared to the one without forwarding?

Speedup from Forwarding hardware resolving data hazards

Making the implicit assumption that the average CPI for the n-instruction Program is 1 and assuming no data hazards are encountered, in a Pipeline executing at cycle time of 250 ps takes

- T0 = n x 250 ps
- Assuming 0.4 NOPS per instruction, in optimized program without forwarding
- T1 = 1.4n x 250ps
- Adding forwarding hardware reduces the number of NOPS to 0.05n but also increases the cycle time to 300ps yielding total execution time of:
- T2 = 1.05 x n x 300 ps
- Therefore, speedup = T1/T2 = [1.4 x 250] /[1.05 x 300] = 1.11

Limits on minimum NOPs while still improving exec time

Consider a version of the pipeline from *Section 4.5 in RISC-V text* that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). Suppose that (after optimization) a typical n- instruction program requires an additional **0.**4*n NOP instructions to correctly handle data hazards.

1.2 Different programs will require different amounts of NOPs. How many NOPs (as a percentage of code instructions) can remain in the typical program before that program runs slower on the pipeline *with forwarding*?

Limits on minimum NOPs while still improving exec time

Without Forwarding, typical program execution time,

- T0 = 1.4n × 250ps

With Forwarding, assuming γ = number of NOPs per instruction after forwarding. Execution time:

- TF = $(1 + \gamma) \times n \times 300ps$

Maximum number of NOPs per instruction, γ for faster hardware forwarding is restricted by

- T0 ≥ TF
- $1.4n \times 250ps \geq (1 + \gamma) \times n \times 300ps$
- Solving, $\gamma \leq 0.167$

Limits on minimum NOPs while still improving exec time

Consider a version of the pipeline from *Section 4.5 in RISC-V text* that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). Suppose that (after optimization) a typical n- instruction program requires an additional **0.**4*n NOP instructions to correctly handle data hazards.

1.3 Repeat 1.2; however, this time let x represent the number of NOP instructions relative to n. (In 1.2, x was equal to 0.4) Your answer will be with respect to x.

Limits on minimum NOPs while still improving exec time

- Replacing 0.4 for x in Problem 1.2, we get
- $(1+x)n \times 250ps \geq (1 + \gamma) \times n \times 300ps$
- solving for γ, we get the limits on min NOPs as a function of the NOPs/instruction in the program without forwarding
  - More NOPs permissible with forwarding to improve performance for higher x
  - If x goes below a minimum threshold, it is pointless to use forwarding hardware

[next 2 questions] $\gamma < [T_{nf}x - (T_f - T_{nf})]/T_f$

$\gamma < (250x - 50)/300$ **[A]**

Limits on minimum NOPs while still improving exec time

Consider a version of the pipeline from *Section 4.5 in RISC-V text* that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). Suppose that (after optimization) a typical n- instruction program requires an additional **0.**4*n NOP instructions to correctly handle data hazards.

1.4 Can a program with only .075*n NOPs possibly run faster on the pipeline with forwarding? Explain why or why not.

Limits on minimum NOPs while still improving exec time

- In the best case, where forwarding results in no NOPS at all, the execution time will be 300n which is more than 250 x 1.075n
- So the given program cannot possibly run faster since the NOPs per instruction of the optimized program of 0.075 is too low to improve performance with forwarding – essentially too few data hazards to begin with

Limits on minimum NOPs while still improving exec time

Consider a version of the pipeline from *Section 4.5 in RISC-V text* that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). Suppose that (after optimization) a typical n- instruction program requires an additional **0.**4*n NOP instructions to correctly handle data hazards.

1.5 At minimum, how many NOPs (as a percentage of code instructions) must a program have before it can possibly run faster on the pipeline with forwarding?

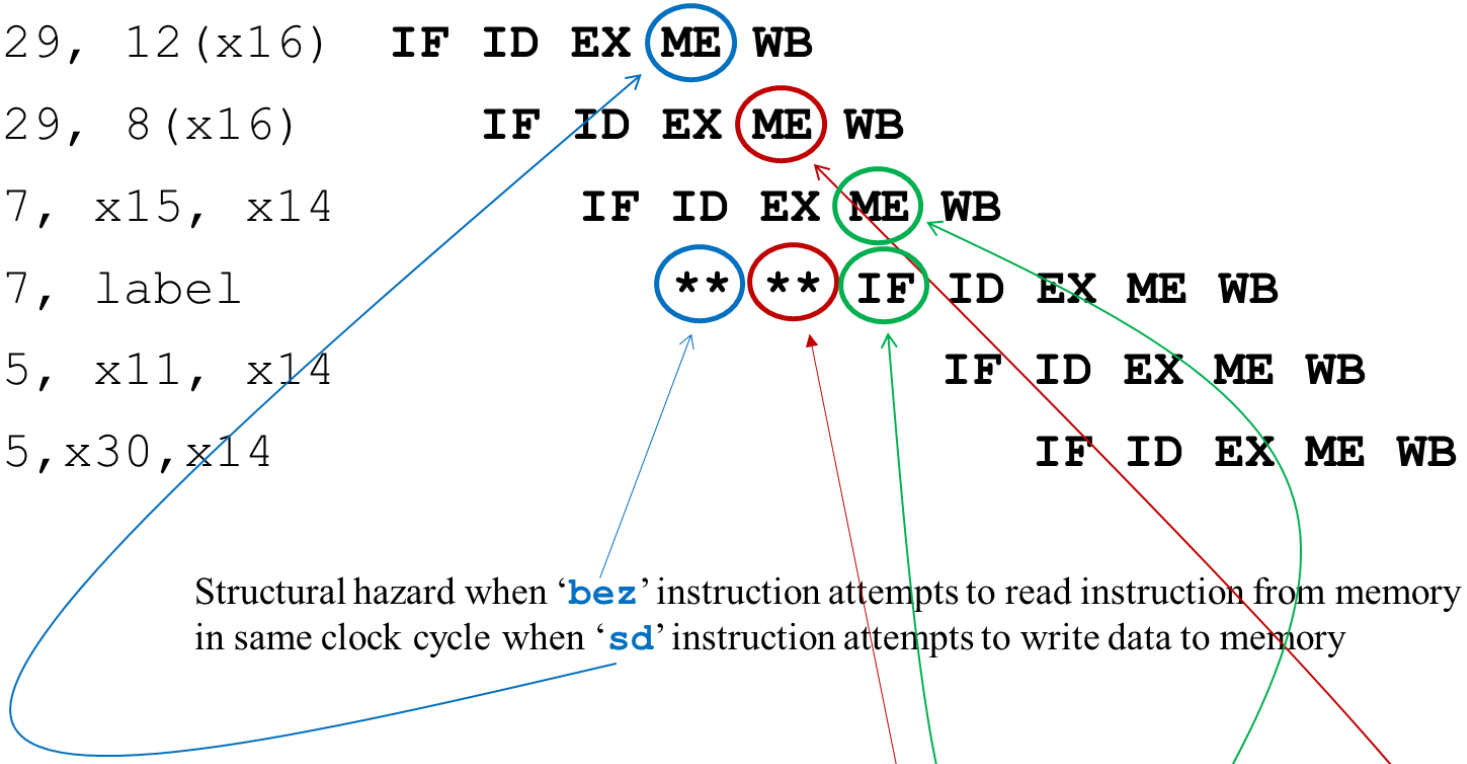
Limits on minimum NOPs while still improving exec time

- The larger the number of NOPs per instruction, **x** without forwarding, the easier it is for forwarding hardware to work and lower the number of NOPs per instruction.
- As the number of NOPS per instruction without forwarding, **x** decreases, it becomes physically impossible to go faster with forwarding if there are very few data hazards. This minimum occurrence of data hazards is when **x** is at a minimum corresponding to the case of $\gamma=0$ in equation **[A]** {slide 5}.
- This minimum value of x can be solved in **[A]** for $\gamma = 0$ as x = 0.2

Pipeline with Structural hazards

2.1 Draw a pipeline diagram to show where the given code above will stall.

- `sd  x29, 12(x16)`
- `ld  x29, 8(x16)`
- `sub x17, x15, x14`
- `bez x17, label`
- `add x15, x11, x14`
- `sub x15,x30,x14`



Structural hazard when '**bez**' instruction attempts to read instruction from memory in same clock cycle when '**sd**' instruction attempts to write data to memory

Structural hazard encountered again when '**bez**' instruction attempts to read instruction from memory in same clock cycle when '**ld**' instruction attempts to read data to memory

No structural hazard encountered in next cycle when '**bez**' instruction attempts to read instruction from memory in same clock cycle as '**sub**' instruction because **sub** instruction (R-Type) does not access memory

Pipeline with Structural hazards

2.2 In general, is it possible to reduce the number of stalls/NOPs resulting from this structural hazard by reordering code?

- The conflict for access to Memory will always be present between the first RISC pipeline stage and the Fourth – from 3 instructions upstream, *if that first instruction is a load/store instruction*.
- It is not possible to lower NOPs by reordering the code since every instruction 3 cycles after a load/store instruction must access memory for IF

Pipeline with Structural hazards

2.3 Must this structural hazard be handled in hardware? We have seen that data hazards can be eliminated by adding NOPs to the code. Can you do the same with this structural hazard? If so, explain how. If not, explain why not.

- The only way to resolve this structural hazard in hardware is by adding a separate memory array for Data and for Instructions.
- Adding NOPs to avoid the structural hazard can also resolve it but adds significant penalty to execution time

Pipeline with Structural hazards

2.4 Approximately how many stalls would you expect this structural hazard to generate in a typical program? (Use the instruction mix shown below)

R-type/I-type (non-ld)	ld	sd	beq
52%	25%	11%	12%

- 36% of the instructions will stall corresponding to all of the load/store instructions in a typical program represented in table above

Overlapping EX and MEM stages of the RISC pipeline

3. If we change load/store instructions to use a register (without an offset) as the address, these instructions no longer need to use the ALU. (See Problem 4 in HW 5) As a result, the MEM and EX stages can be overlapped and the pipeline has only four stages.

3.1 How will the reduction in pipeline depth affect the cycle time?

- Cycle time is dependent on the latency of the slowest of the 5 RISC pipeline stages not on the number of stages. Cycle time will not change by reduction of pipeline depth

Overlapping EX and MEM stages of the RISC pipeline

3.2 How might this change improve the performance of the pipeline?

- By overlapping the MEM with the EX pipeline stages, fewer pipeline stage enables the latency of an instruction to improve.
- Also, load-use-data hazards that require a NOP cycle inserted between a load instruction and an instruction that uses this data read from memory, can eliminate the need of this NOP cycle potentially lowering the Execution Time penalty in NOPS per instruction for load-use-data hazards

Overlapping EX and MEM stages of the RISC pipeline

3.3 How might this change degrade the performance of the pipeline?

- Removing the offset from `ld/sd` instructions can require `addi` to be used/paired with `ld/sd` instructions thus increasing the number of instructions in the program
- If the improvement in execution time from resolving load-use-data hazards is less than degradation from more instructions, this overlap would not be implemented

Load-use-data hazard resolution

4. Which of the two pipeline diagrams below better describes the operation of the pipeline's hazard detection unit? Why?

Choice 1:

```
ld x11, 0(x12):          IF ID EX ME WB
add x13, x11, x14:          IF ID EX .. ME WB
or x15, x16, x17:              IF ID .. EX ME WB
```

Choice 2:

```
ld x11, 0(x12):          IF ID EX ME WB
add x13, x11, x14:          IF ID .. EX ME WB
or x15, x16, x17:              IF .. ID EX ME WB
```

```
                                CC1CC2CC3CC4CC5CC6CC7CC8
ld x11, 0(x12):                 IF ID EX ME WB
add x13, x11, x14:                 IF ID EX .. ME WB
or x15, x16, x17:                     IF ID .. EX ME WB

ld x11, 0(x12):                 IF ID EX ME WB
add x13, x11, x14:                 IF ID .. EX ME WB
or x15, x16, x17:                     IF .. ID EX ME WB
```

Choice 2 because:

In the first sequence, content of register x11 becomes available only at the end of CC4 but is being used at the start of CC4 in the execute stage of the add instruction – this is a ‘load-use-data-hazard’. Insertion of a stall in CC5 is too late

In the second sequence, content of register x11 which becomes available at the end of CC4 is used at the start of CC5 (and not CC4 as above) since the EX stage has been delayed by insertion of a stall in CC4

Load-use-data hazard resolution

5. Consider the following loop.

```
LOOP: ld   x10, 0(x13)
      ld   x11, 8(x13)
      add  x12, x10, x11
      subi x13, x13, 16
      bnez x12, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the EX (as opposed to the ID) stage.

Load-use-data hazard resolution

5.1 Show a pipeline execution diagram for the first two iterations of this loop

5.2 Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the `subi` is in the IF stage. End with the cycle during which the bnez is in the IF stage.)

Load-use-data hazard resolution

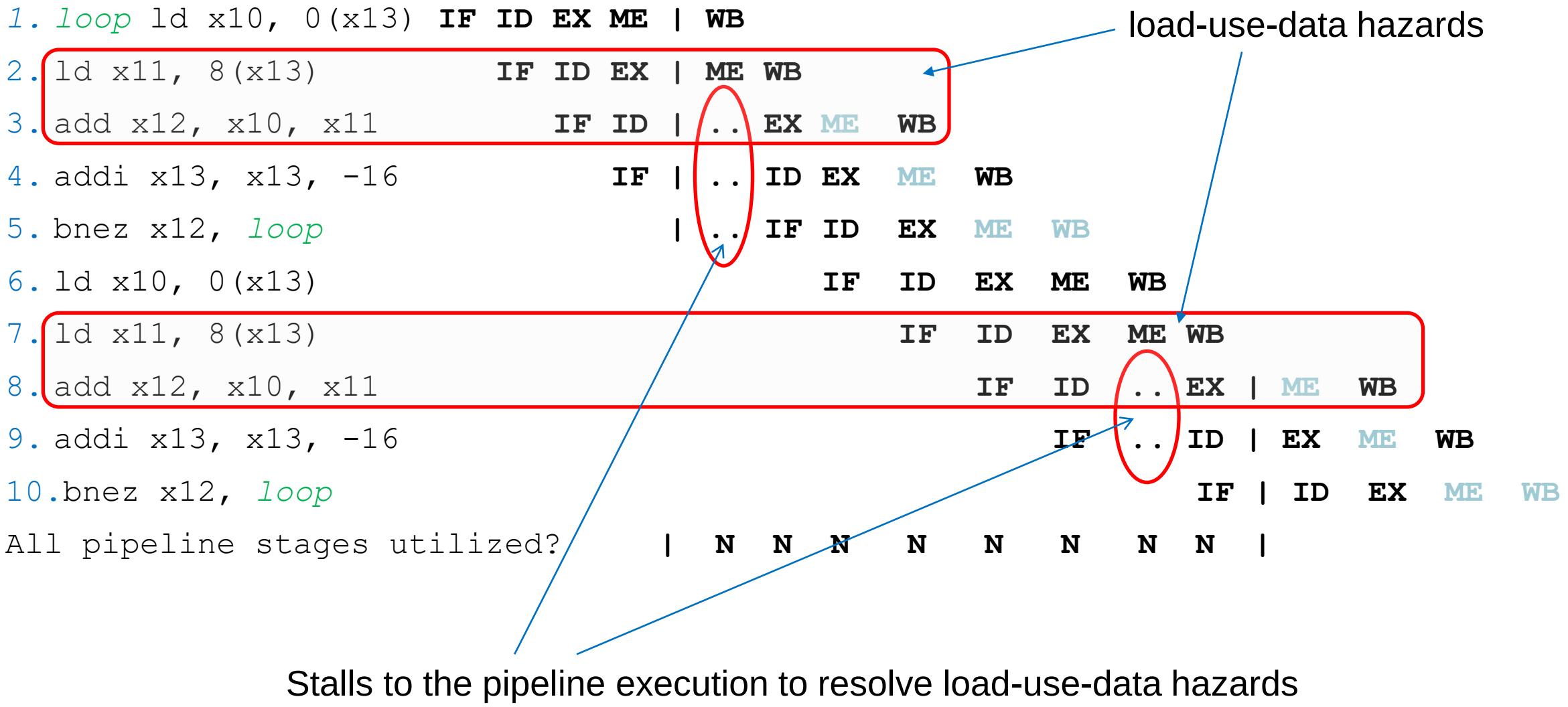
[1] Since perfect branch prediction is used, we do not lose any cycles due to branch hazards – that is, the branch is always predicted and taken correctly

[2] Availability of full forwarding support is assumed, so we can assume data hazards that can be resolved with forwarding do not stall the pipeline

[3] Load-use-hazards cannot be resolved and are identified in next slide in red boxes - resolved by stalling the pipeline

[4a] pipeline stages unused by any instruction are identified in blue

[4b] any clock cycle that does not have all of the pipeline stages utilized is identified with 'N'


1. loop ld x10, 0(x13) IF ID EX ME | WB
2. ld x11, 8(x13) IF ID EX | ME WB
3. add x12, x10, x11 IF ID | .. EX ME WB
4. addi x13, x13, -16 IF | .. ID EX ME WB
5. bnez x12, loop | .. IF ID EX ME WB
6. ld x10, 0(x13) IF ID EX ME WB
7. ld x11, 8(x13) IF ID EX ME WB
8. add x12, x10, x11 IF ID .. EX | ME WB
9. addi x13, x13, -16 IF .. ID | EX ME WB
10. bnez x12, loop IF | ID EX ME WB
All pipeline stages utilized? | N N N N N N N N |
load-use-data hazards
Stalls to the pipeline execution to resolve load-use-data hazards

RAW Data Dependencies

6. This exercise is intended to help you understand the cost/complexity/performance trade-offs of forwarding in a pipelined processor. Problems in this exercise refer to pipelined datapaths from *Figure 4.53 in RISC-V text (reproduced below)*. These problems assume that, of all the instructions executed in a processor, the following fraction of these instructions has a particular type of RAW data dependence

EX to 1st Only	MEM to 1st Only	EX to 2nd Only	MEM to 2nd Only	EX to 1st and EX to 2nd
5%	20%	5%	10%	10%

RAW Data Dependencies

6.1 For each RAW dependency listed in table previous slide, give a sequence of at least three assembly statements that exhibits that dependency.

(1) EX to 1st only:

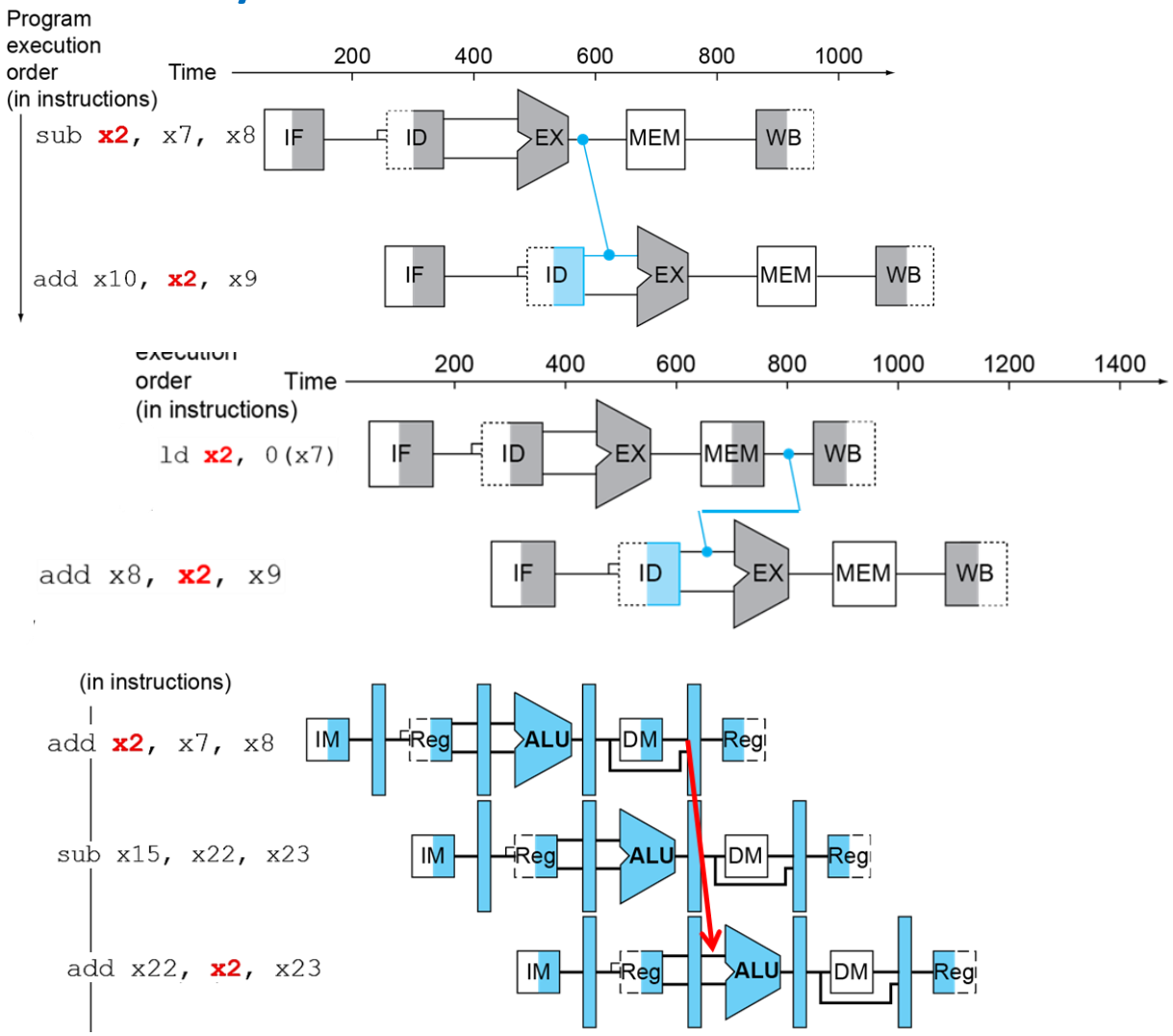
```
sub x2, x7, x8
add x10, x2, x9
add x27, x28, x29
```

(2) MEM to 1st only:

```
ld x2, 0(x7)
add x8, x2, x9
sub x15, x22, x23
```

(3) EX to 2nd only:

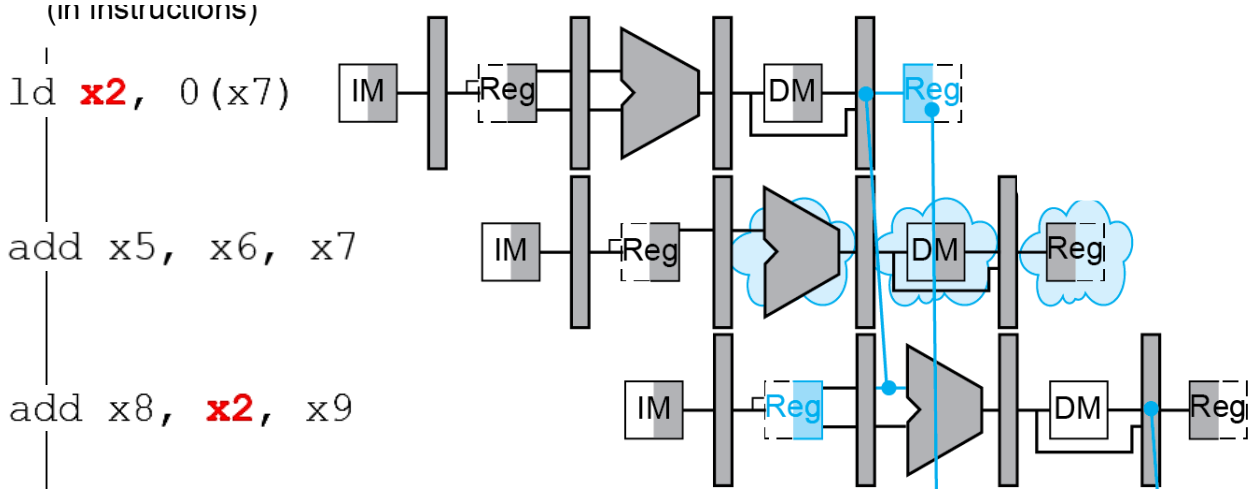
```
add x2, x7, x8
sub x15, x22, x23
add x22, x2, x23
```



RAW Data Dependencies

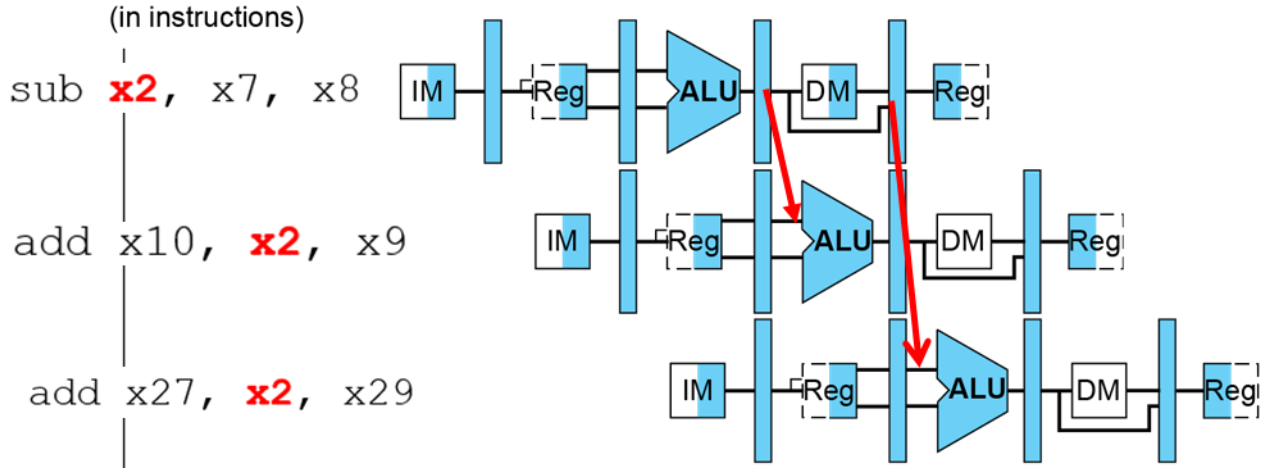
(4) MEM to 2nd only:

```
ld x2, 0(x7)
add x5, x6, x7
add x8, x2, x9
```



(5) EX to 1st and EX to 2nd :

```
sub x2, x7, x8
add x10, x2, x9
add x27, x2, x29
```



RAW Data Dependencies

6.2, 6.3 For each RAW dependency above, how many NOPs would need to be inserted to allow your code from ***6.1*** to run correctly on a pipeline with no forwarding or hazard detection? Show where the NOPs could be inserted.

RAW Data Dependencies [no forwarding]

EX to 1st only:

```
sub x2, x7, x8
NOP
NOP
add x10, x2, x9
add x27, x28, x29
```

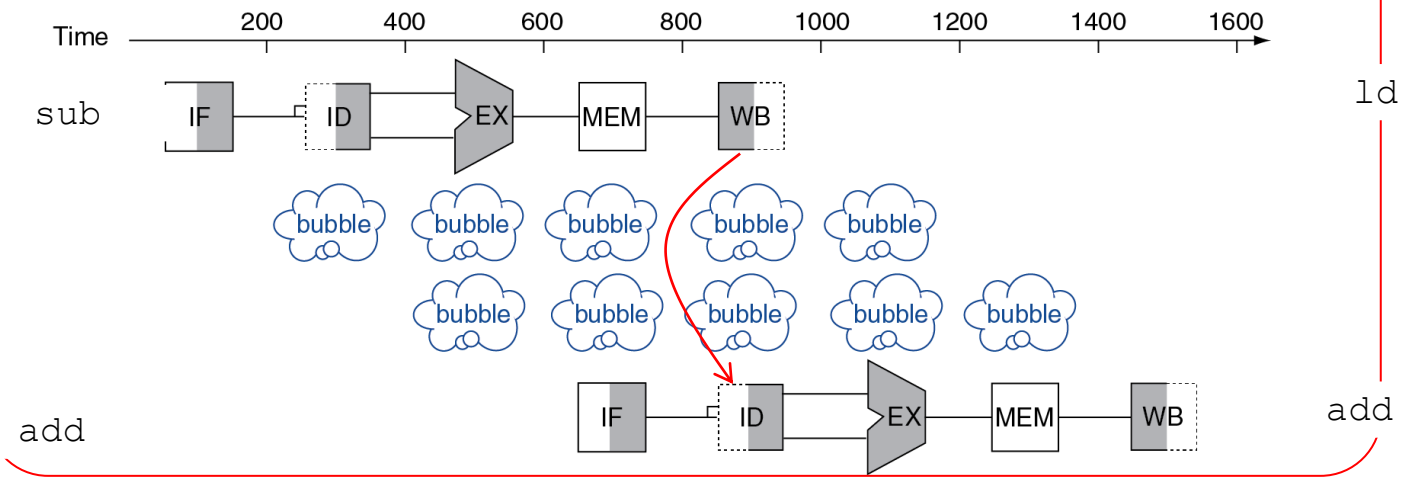
MEM to 1st only:

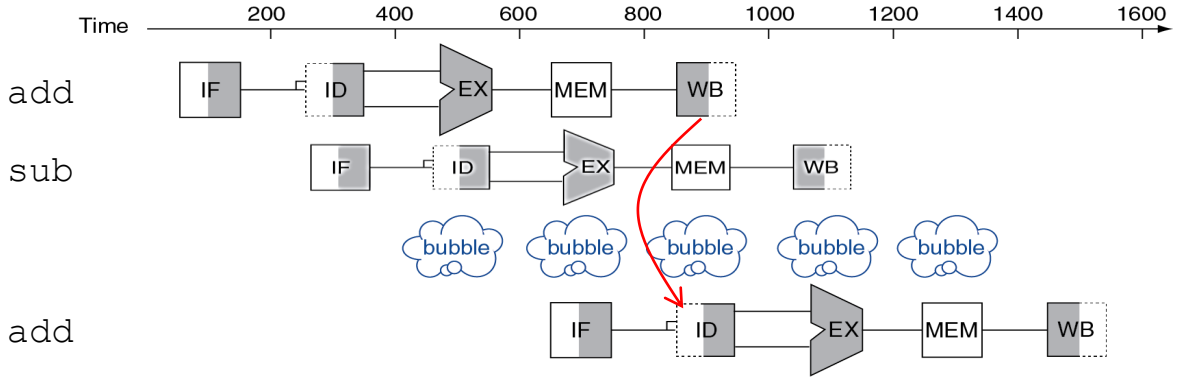
```
ld x2, 0(x7)
NOP
NOP
add x8, x2, x9
```

EX to 2nd only:

```
add x2, x7, x8
sub x15, x22, x23
NOP
add x22, x2, x23
```

Clock Cycle in which Rd is written is the earliest cycle when that register can be read by the instruction creating a hazard

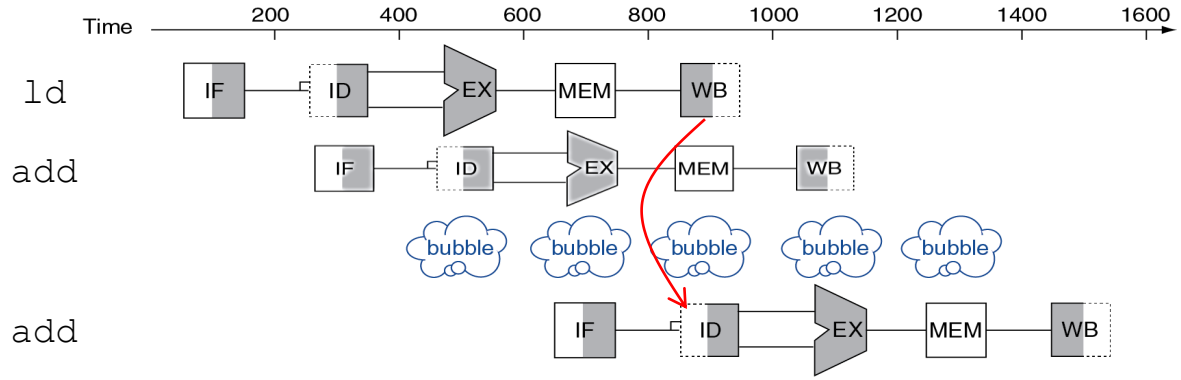




RAW Data Dependencies

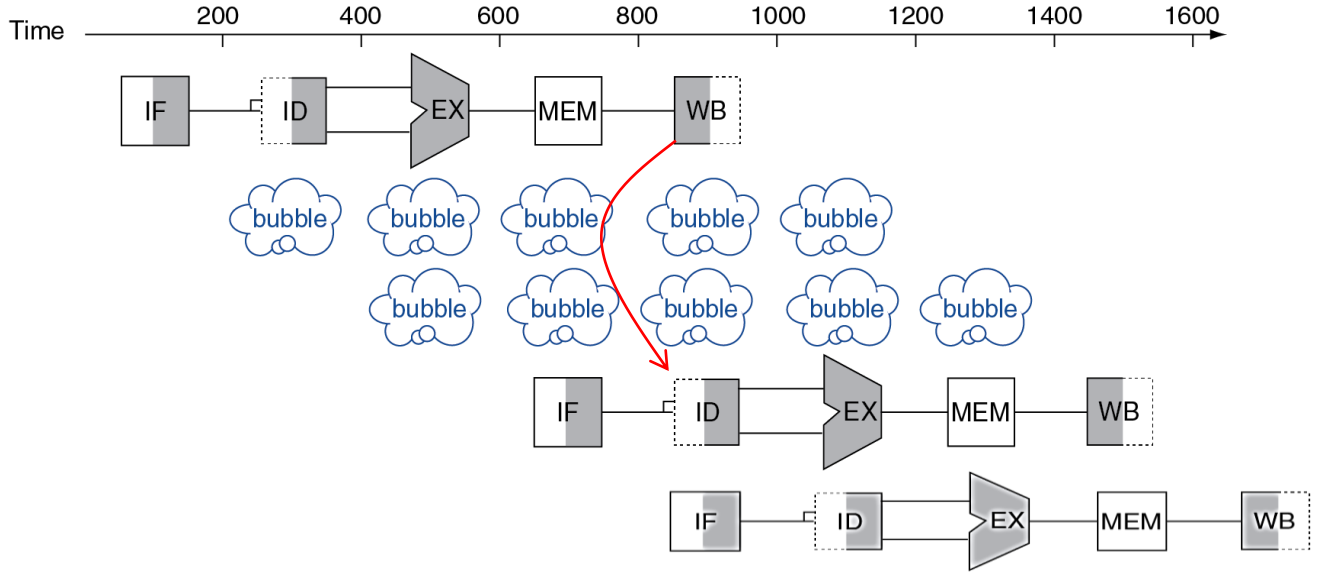
MEM to 2nd only:

```
ld x2, 0(x7)
add x5, x6, x7
NOP
add x8, x2, x9
```



EX to 1st and EX to 2nd :

```
sub x2, x7, x8
NOP
NOP
add x10, x2, x9
add x27, x2, x29
```



CPI Without Forwarding

6.4 Assuming no other hazards, what is the CPI for the program described by the table above when run on a pipeline with no forwarding? What percent of cycles are stalls? (For simplicity, assume that all necessary cases are listed above and can be treated independently.)

EX to 1st Only	MEM to 1st Only	EX to 2nd Only	MEM to 2nd Only	EX to 1st and EX to 2nd
5%	20%	5%	10%	10%

- Taking a weighted average of the number of NOPs for each from 6.2 gives

0.05*2 + 0.2*2 + 0.05*1 + 0.1*1 + 0.1*2 = 0.85 NOPs per instruction

- a CPI of 1.85.
- So, 0.85/1.85 cycles = 46%, are NOPs.

CPI improvement with Full Forwarding

6.5 What is the CPI if we use full forwarding (forward all results that can be forwarded)? What percent of cycles are stalls?

EX to 1st Only	MEM to 1st Only	EX to 2nd Only	MEM to 2nd Only	EX to 1st and EX to 2nd
5%	20%	5%	10%	10%

- The only RAW dependency that cannot be handled by load-use-data hazards with forwarding is from the `MEM stage to the next instruction`
- 20% of instructions will generate one NOP for a CPI of 1.2.
- 0.2 out of 1.2 cycles = 17% NOPs

Partial Forwarding – use EX/MEM only

6.6 Let us assume that we cannot afford to have **three-input** multiplexors that are needed for full forwarding. We have to decide if it is better to forward only from the **EX/MEM pipeline register** (next-cycle forwarding) or only from the MEM/WB pipeline register (two-cycle forwarding). What is the CPI for each option?

If we forward from the **EX/MEM register only**, we have the following stalls/NOPs

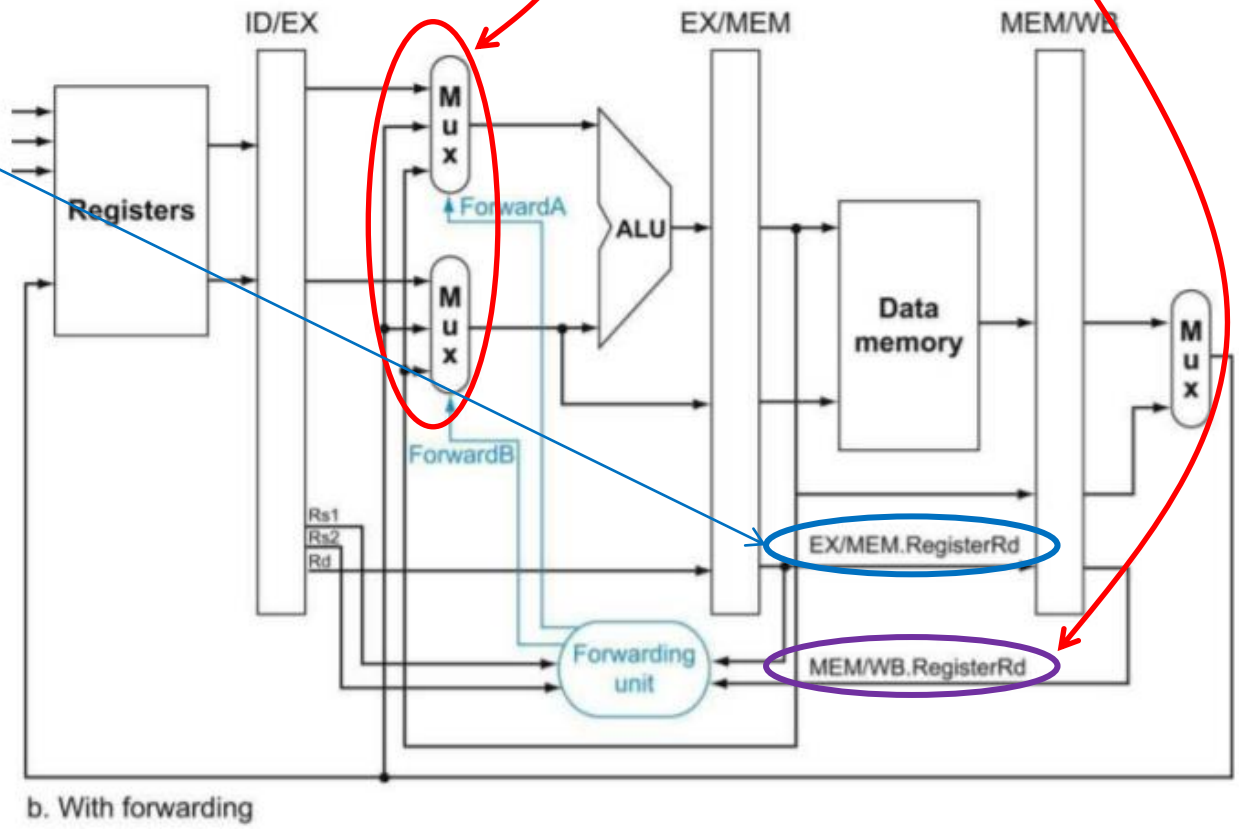
EX to 1st: **0** (eq 1)
This hazard needs the **EX/MEM** register with which the forwarding unit eliminates the NOP, so NOPs = 0

MEM to 1st: **2** (eq 2) [must insert 2 NOPs] This hazard needs the MEM/WB register (load-use-data hazard) and cannot be resolved, so 2 NOPs must be inserted

EX to 2nd: **1** (eq 3) This hazard needs the MEM/WB register and cannot be resolved so a NOP must be inserted

MEM to 2nd: **1** (eq 4) This hazard needs the **MEM/WB** register and cannot be resolved, so a NOP is inserted

EX to 1st and 2nd: **1** (eq 5) EX to 1st can be resolved with the EX/MEM register but the EX to 2nd needs the MEM/WB register and cannot be resolved so a NOP is inserted to resolve the EX to 2nd hazard



Partial Forwarding – use MEM/WB only

6.6 Let us assume that we cannot afford to have **three-input** multiplexors that are needed for full forwarding. We have to decide if it is better to forward only from the **EX/MEM pipeline register** (next-cycle forwarding) or only from the MEM/WB pipeline register (two-cycle forwarding). What is the CPI for each option?

If we forward from the **MEM/WB register**, we have the following stalls/NOPs

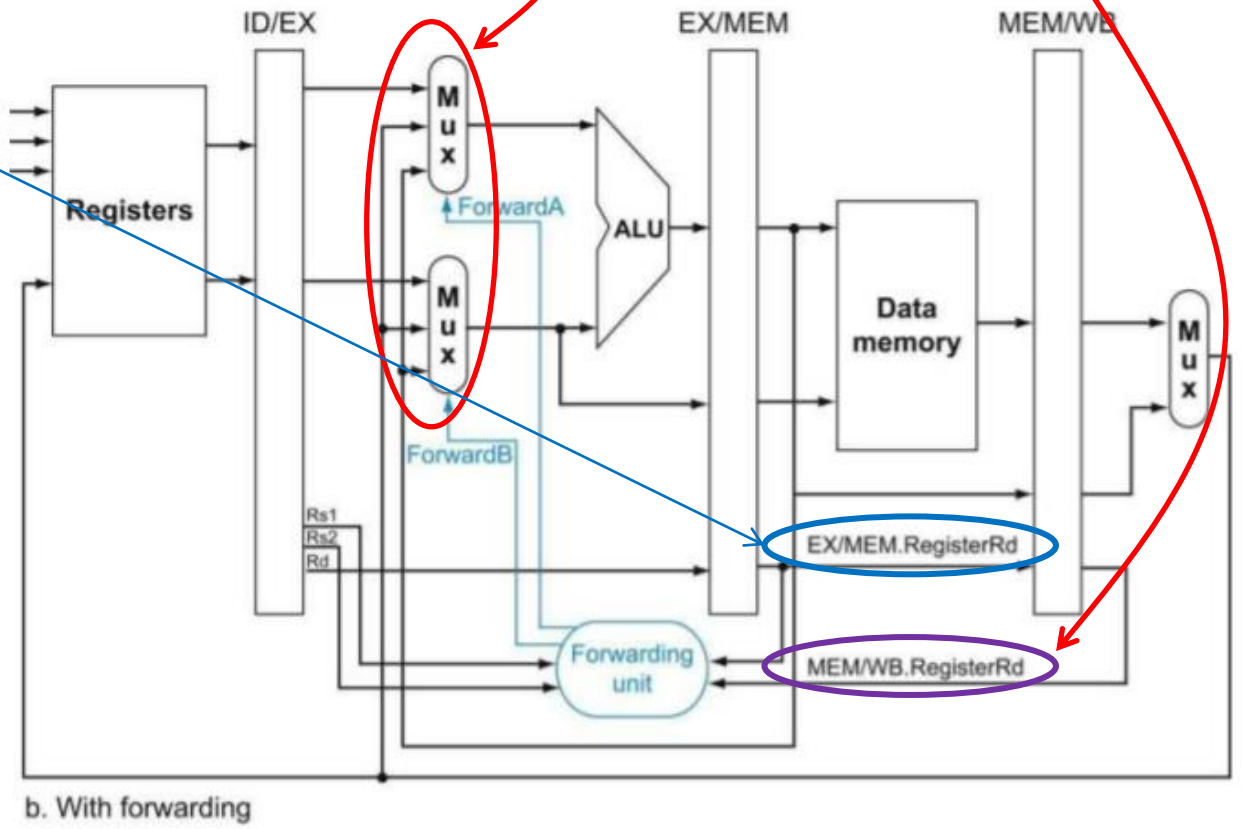
EX to 1st: **1**(eq 1)
This hazard needs the **EX/MEM** register and can be resolved with the MEM/WB register, if the MEM/WB register forwards to the ALU after 1 NOP, so NOP =1

MEM to 1st: **1** (eq 2) This hazard needs the MEM/WB register (load-use-data hazard) but 1 NOP must be inserted

EX to 2nd: **0** (eq 3) This hazard needs the MEM/WB register and is resolved with 0 NOPs

MEM to 2nd: **0** (eq 4) This hazard needs the MEM/WB register and and is resolved with 0 NOPs

EX to 1st and 2nd: **1** (eq 5) EX to 1st cannot be resolved and needs 1 NOP, but the EX to 2nd can be resolved with the MEM/WB register



b. With forwarding

CPI with Partial Forwarding

- With MEM/WB register unavailable,
- Average NOPs of 0.05*0 + 0.2*2 + 0.05*1 + 0.10*1 + 0.10*1

= 0.65 stalls/instruction

So, CPI = 1.65

- With EX/MEM register unavailable,
- Average NOPs of 0.05*1 + 0.2*1 + 0.1*1

= 0.35 stalls/instruction

So, CPI = 1.35

Speedup with Full F, Partial F, NF

6.7 For the given hazard probabilities and pipeline stage latencies, (1) what is the speedup achieved by each type of forwarding (EX/MEM, MEM/WB, for full) as compared to a pipeline that has no forwarding?

Speedup with Full F, Partial F, NF

- We calculated CPI for NF, EX/MEM only, MEM/WB only and with Full Forwarding
- Period for any of the above is the longest latency stage [FF needs 130 ps w FF unit]

	No Forwarding	EX/MEM	MEM/WB	Full Forwarding
CPI	1.85	1.65	1.35	1.2
Period	120ps	120ps	120ps	130ps
Time	222ps	198ps	162ps	156ps
Speedup	ref	1.12	1.37	1.42

Performance Limits with zero Hazards

6.8 What would be the additional speedup (relative to the fastest processor from 6.7) be if we added "timetravel" forwarding that eliminates all data hazards? Assume that the yet-to-be-invented time-travel circuitry **adds 100 ps to the latency of the full-forwarding EX stage.**

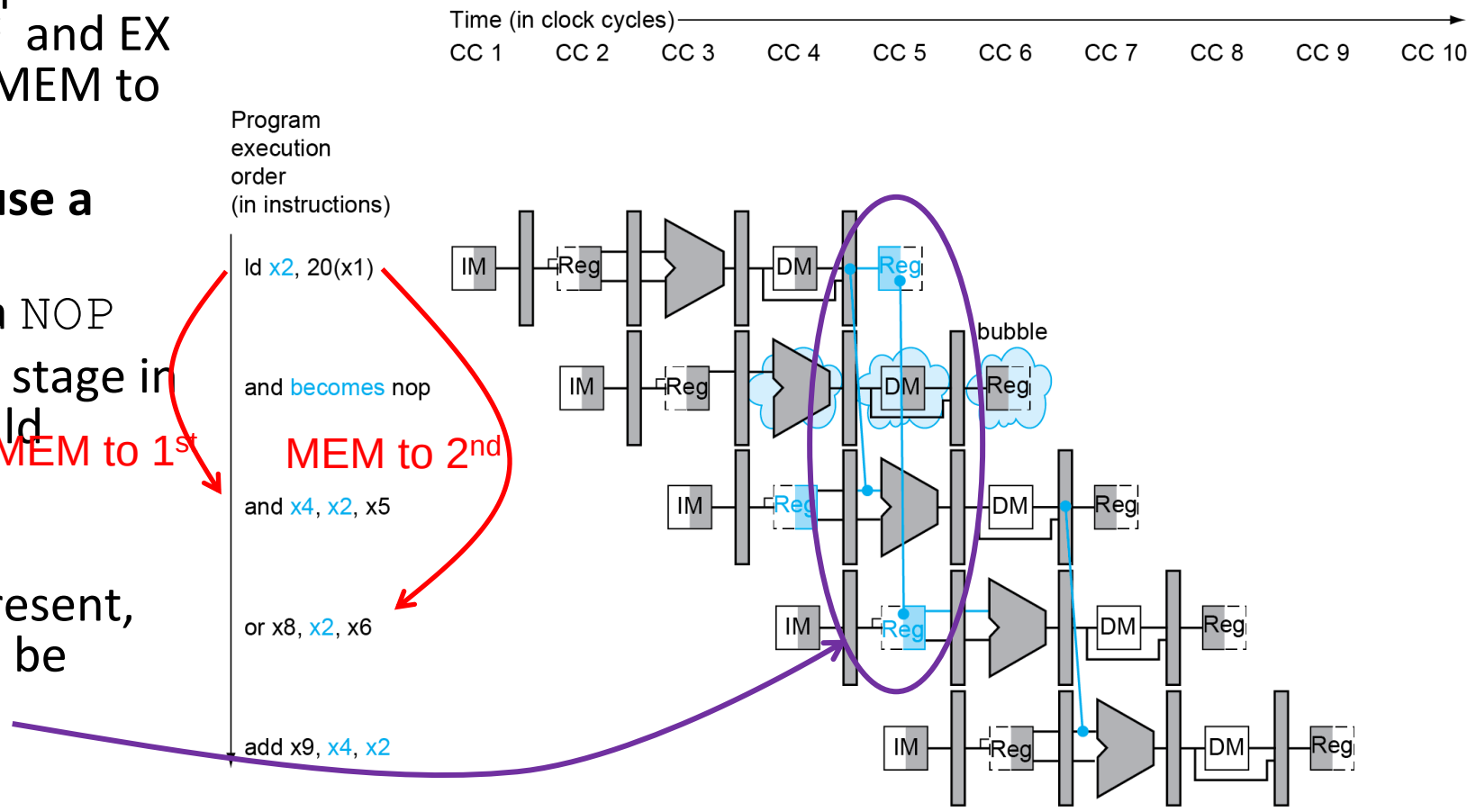
- CPI with full forwarding is 1.2
- CPI for "time travel" forwarding is 1.0 [Max with 0 Hazards]
- Clock period for full forwarding is 130
- Clock period for zero hazards forwarding is 230
- Speedup = Told/Tnew = (1.2*130)/ (1*230) = 0.68
- Zero hazard forwarding actually slows the CPU

MEM to 1st and MEM to 2nd

6.9 The table of hazard types has separate entries for “EX to 1st ” and “EX to 1st and EX to 2nd ”. Why is there no entry for “MEM to 1st and MEM to 2nd ”?

- All MEM to 1st instructions **will cause a stall** (load-use-data hazard)
  - So instruction after `ld` must be a `NOP`
  - So, 2nd instruction would have ID stage in same Clock Cycle as WB stage of ld instruction
  - This is not a hazard anymore

So, if a MEM to 1st RAW hazard is present, the MEM to 2nd RAW hazard cannot be present!



Hazard Resolution

7. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add x15, x12, x11
ld x13, 4(x15)          EX to 1st RAW Hazard
ld x12, 0(x2)
or x13, x15, x13        MEM to 2nd RAW Hazard
sd x13, 0(x15)          EX to 1st RAW Hazard
```

7.1 If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.

7.1 No Hazard Resolution – only NOP insertion

```
add x15, x12, x11
nop
nop                EX to 1st RAW Hazard resolution with 2 NOPs
ld x13, 4(x15)
ld x12, 0(x2)
nop                MEM to 2nd RAW Hazard resolution with 1 NOP
or x13, x15, x13
nop
nop                EX to 1st RAW Hazard resolution with 1 NOP
sd x13, 0(x15)
```

Code Optimization assuming NOP resolution

7.2 Now, change and/or rearrange the code to minimize the number of NOPs needed. You can assume register x17 can be used to hold temporary values in your modified code.

`not possible to reduce the number of NOPs.`

7.3 No Hazard Detection Unit

7.3 If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?

- The code executes correctly.
- Hazard detection relevant only to insert a stall for load-use-data hazards
- The given instruction sequence does not have `ld` followed by use of register written into

Highlights of Exercises on the Memory Hierarchy

- Calculating Read and Write Bandwidth dependencies on Write Policy
- Impact of Cache Size on Memory Performance: Access Time Vs Miss Rate – multi level cache within a Processor chip
- Lowering Miss Rates – Direct Mapped Vs Associative Caches

8. Memory Bandwidth for Read, Write in WT/WB cache

- Given:
  - Write-through, Write Allocate as the Write Hit & Write Miss policies
  - CPI =2 (or 0.5 Instructions/cycle), Block size = 64 Bytes
  - Reads:
    - 100% of instructions generate a Read Instruction Cache request with a 0.3% miss rate
    - 25% of instructions generate a Read Data Cache request with 2% miss rate
  - Writes:
  - 10% of Instructions generate Write requests to Data Cache

Data Reads per 1000 Instructions	Data Writes per 1000 Instructions	Instruction Cache Miss Rate	Data Cache Miss Rate	Block Size (bytes)
250	100	0.30%	2%	64

Memory Bandwidth for Read

- What is the bandwidth b/w RAM and Instruction Cache? (note: only Read ops performed on Instruction memory)
  - 0.3% of instructions read from Instruction cache register a Read miss
  - So, 0.003 x 0.5 instructions /cycle = missed instructions per cycle
  - Block of 64 Bytes corresponding to the miss must be fetched from memory
  - So, [0.003 misses/I] x [0.5 I/cycle] x [64 bytes/miss] = **0.096 Bytes/cycle**
- What is the Read bandwidth b/w RAM and Data Cache
  - 25% of instructions generate a read request to Data cache
  - [0.25 Reads/I] x [0.5 I/cycle] = 0.125 Reads/cycle
  - 2% of these miss: So, [0.02 misses/Read]x[0.125 Reads/cycle] = 0.0025 Read misses/cycle
  - Each miss requests 1 block (64 Bytes), So: [0.0025 Read miss/cycle]x64 Bytes = **0.16 Bytes/cycle**

Memory Bandwidth for Write

- What is the Write bandwidth b/w RAM and Data Cache
  - 10% of instructions generate a write request to Data cache
  - [0.10 Writes/I] x [0.5 I/cycle] = 0.05 Writes/cycle
- Since this is a Write-through hit policy, all words written to data cache must be written to RAM
  - So, [0.05 Writes/cycle]x[8 Bytes/word] x [1 word/write-through] = **0.4Bytes/cycle** write bandwidth consumed
- Since this is a Write Allocate Miss policy, A write miss (rate = 2%) must also Read from RAM the Block corresponding to the missing entry (to be written into) into the Data Cache
  - [0.05 Writes/cycle]x[0.02 misses/write]x[64Bytes/miss] = **0.064 Bytes/cycle**
- Assuming each write miss attempts to write only 1 word to the Block brought in from RAM, into the RAM to update it:
  - [0.05 Writes/cycle]x[0.02 misses/write]x[8Bytes/word]x[1 word/miss] = **0.008 Bytes/cycle** (included in above calculation of BW for all words written to RAM w Write-through

Total Bandwidth

- **Read**: **0.096** (Instruction memory) + **0.16** (data memory) + **0.064** (Write-miss in Write-through cache with Write Allocate) Bytes/cycle = 0.32 Bytes/cycle
- **Write**: **0.408 Bytes/cycle** (all Word Writes to Data Cache written to RAM)

8.2 Assume a Write Back Cache Hit Policy

- For a write-back, write-allocate cache, assuming 30% of replaced data cache blocks are dirty, what are the **read** and **write** bandwidths needed for a CPI of 2?
- With a write-back, write allocate cache*, data are only written to memory on a cache miss*.
- But, it is written to memory *on every cache miss* (both read and write), because any line could have dirty data when evicted, even if the eviction is caused by a read request
- assuming *30% of replaced data cache blocks are dirty* what are the read and write bandwidths needed assuming the same CPI?

Write Data Bandwidth

- # of Read & Write Accesses to Cache:
- [0.5 inst/cycle] x [0.10 Write Data Accesses/instruction + 0.25 Read Data Accesses/instruction] = 0.175 Accesses/cycle
- [0.175 Accesses/cycle] x [0.02 misses /Access] = 0.0035 misses/cycle

Of these misses, 30% are 'dirty' – that is they need to be updated in RAM before being evicted from Cache

- [0.0035 misses/cycle] x [0.3 blocks/miss] = 0.00105 blocks/cycle
- [0.00105 blocks/cycle] x [64 bytes/block] = =**0.0672 bytes/cycle**

Problem 9:

Briefly explain why many of the transistors on a modern microprocessor chip are devoted to Level 1, Level 2, and sometimes Level 3 cache.

1. Processing logic is faster than off-chip, 1-transistor DRAM – and this performance gap has been continually growing - Accessing DRAM has a severe Miss penalty in number of core clock cycles
2. Lower Miss Penalty by doing 2 things:
   - Add levels to the Memory hierarchy with smaller penalty in core clock cycles using faster on-chip memory (SRAM) that can be accessed within a few cycles of core clock rate
   - Increase the size of added levels of the Hierarchy to lower their miss rates and ensuing penalties to go off-chip to DRAM
3. Size of Processor chip goes up (and its cost) substantially. So does leakage power from the large number of SRAM bitcells now on-chip in the L2, L3

Problem 9a

- Some versions of the Pentium 4 microprocessor have two 8 Kbyte, Level 1 caches – one for data and one for instructions. However, a design team also considered another option – a single, 16 Kbyte cache that holds both instructions and data where
  - Each block will hold 32 bytes of data (not including tag, valid bit, etc.)
  - The cache would be 2-way set associative
  - Physical addresses are 32 bits
  - Data is addressed to the word and words are 32 bits

Solution

- How many blocks would be in this cache?
  - 2^{14} bytes in the cache (16KBytes) / 32 bytes/block = **512 blocks**
- How many bits of tag are stored with each block entry?
- 32 address Bits are dedicated to the offset, index and tag
- Index:
  - # of sets: 512 /2 = 256 = 2^8
  - Therefore **8** bits of index are needed
- Offset:
  - # of words per block = 32 / 4 = 8
  - 2^3 = 8
  - Therefore **3** bits for Block offset
- Tag
  - 32 – 3 – 8 = **21** bits of tag

Problem 9b

- Each instruction fetch means a reference to the instruction cache and 35% of all instructions reference data memory. With the first implementation:
  - The average miss rate in the L1 instruction cache was 2%
  - The average miss rate in the L1 data cache was 10%
  - In both cases, the miss penalty is 9 CCs
- For the new unified design, the average miss rate is 3% for the cache as a whole, and the miss penalty is again 9 CCs.
- Which design is better and by how much?

Solution

- Miss penalty Separate Instruction (8KB) and Data (8KB) Cache

= (1)(.02)(9) + (0.35)(.1)(9) = .18 + .063 = 0.495

- Miss penalty unified 16KB Instruction & Data Cache

= (.03)(9) = 0.270

Unified Cache is the right design choice

10 Cache Size Impact on Performance

- Access time proportional to cache size
  - more bits, longer wires, more decode gate delays etc.
- Miss rate lower for larger cache size
  - more likely to find data in the cache if it is bigger, even though it is slower
- So **what optimal value** of each enables best Cache performance?

	L1 Size	L1 Miss Rate	L1 Hit Time
P1	2 KiB	8.0%	0.66 ns
P2	4 KiB	6.0%	0.90 ns

- Assume: RAM takes 70ns to access, 36% of all instructions access data cache. Table above is data for 2 processors P1, P2

10.1 Clock rates for P1, P2

- Assuming L1 access determines cycle time for P1, P2 (that is assuming no need to go to RAM) what are the clock rates for P1, P2?
- Clock rate *is generally limited by the L1 cache* – since the access to the L1 must complete within 1 clock cycle.
- Smaller L1s permit faster clocks, so:
- Cycle times for P1 and P2 : **P1: 1.515 GHz; P2: 1.11 GHz**

10.2 AMAT for P1, P2 (without L2)

- What is AMAT for P1, P2 assuming the RAM Latency given and hit rates in table?
- Average Memory Access Time (in cycles)
- = # of cycles for a hit + Miss Rate x Miss Penalty Miss penalty
  - for P1 = 70ns/0.66ns = 107 cycles Miss penalty
  - for P2 = 70ns/0.90ns = 78 cycles
  - P1: AMAT = 1 + 0.08 x 107 cycles = 9.56 cycles or 6.31 ns
  - P2: AMAT = 1 + 0.06 x 78 cycles = 5.68 cycles or 5.11 ns

10.3 CPI for P1 and P2?

- base CPI = 1.0 without any memory stalls
- Total CPI for P1 and P2 are determined by calculating:
- For **P1:**
  - Each instruction requires 1 cycle for a hit 1 cycle
  - If the instruction fetch from ***L1 Instruction memory*** misses at Miss Rate of 8%, the instruction incurs an additional delay of 0.08 x 107 cycles = 8.56 cycles
  - 36% of the instructions also require access to ***L1 Data Memory*** at a Miss rate of 8% which incur an additional delay of + 0.36 x 0.08 x 107 cycles = 3.08 cycles
  - So, 1 + 0.08 x 107 + 0.36 x 0.08 x 107 = ***12.64 cycles @ 0.66ns/cycle = 8.34ns***
- For **P2:**
  - Each instruction requires 1 cycle for a hit 1 cycle
  - If the instruction fetch from ***L1 Instruction memory*** misses at Miss Rate of 6%, the instruction incurs an additional delay of 0.06 x 78 cycles = 4.68 cycles
  - 36% of the instructions also require access to Data Memory at a Miss rate of 6% which incur an additional delay of + 0.36 x 0.06 x 78 cycles = 1.68 cycles
  - So, 1 + 0.06 x 78 + 0.36 x 0.06 x 78 = ***7.36 cycles @ 0.66ns/cycle = 6.63ns***

10.4 AMAT for P1 with the addition of an L2 cache

L2 Size	L2 Miss Rate	L2 Hit Time
1 MiB	95%	5.62 ns

- An L2 access requires 9 cycles (5.62ns/0.66ns).
- All memory accesses require at least one cycle.
- 8% of memory accesses miss in the L1 cache and make an L2 access, which takes 9 cycles.
- 95% of all L2 access are misses and require a 107 cycle RAM lookup.
- AMAT = 1 + .08[9 + 0.95*107] = 9.85 cycles – worse than without the L2 (9.56 cycles)

10.4 AMAT for P2 with the addition of an L2 cache

- An L2 access requires 7 cycles (5.62ns/0.9ns).
- All memory accesses require at least one cycle.
- 6% of memory accesses miss in the L1 cache and make an L2 access, which takes 7 cycles.
- 95% of all L2 access are misses and require a 78 cycle RAM lookup.
- AMAT = 1 + .06[7 + 0.95*78] = 5.87 cycles – worse than without the L2 (5.68 cycles)

L2 Size	L2 Miss Rate	L2 Hit Time
1 MiB	95%	5.62 ns

Problem 11

- Consider an Intel P4 microprocessor with a 16 Kbyte unified L1 cache. The miss rate for this cache is 3% and the hit time is 2 CCs. The processor also has an 8 Mbyte, on-chip L2 cache. 95% of the time, data requests to the L2 cache are found. If data is not found in the L2 cache, a request is made to a 4 Gbyte main memory. The time to service a memory request is 100,000 CCs. On average, it takes 3.5 CCs to process a memory request. How often is data found in main memory?

Solution

- Average memory access time = Hit Time + (Miss Rate x Miss Penalty)
- Average memory access time = $\text{Hit Time}_{L1} + (\text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1})$
- $\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + (\text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$
- $\text{Miss Penalty}_{L2} = \text{Hit Time}_{Main} + (\text{Miss Rate}_{Main} \times \text{Miss Penalty}_{Main})$

$3.5 = 2 + 0.03\ (15 + 0.05\ (200 + X\ (100{,}000)))$

$3.5 = 2 + 0.03\ (15 + 10 + 5000X)$

$3.5 = 2 + 0.03\ (25 + 5000X)$

$3.5 = 2 + 0.75 + 150X$

$3.5 = 2.75 + 150X$

$0.75 = 150X$

$X = .005$

Problem 12: DM Vs Associative Caches

- Briefly explain the advantages and disadvantages (in 4-5 sentences or a bulleted list) of using a direct mapped cache instead of an 8-way set associative cache
  - A direct mapped cache should have a faster hit time; there is only one block that data for a physical address can be mapped to
  - If there are successive reads to 2 separate addresses that map to the same cache block, then there will be a cache miss – degrading performance in DM.
  - In the 8-way set associative caches, a block can map to one of 8 blocks within a set. Thus, both above references would be hits and there would be no conflict misses.
  - A set associative cache will take a bit longer to search – could decrease clock rate.

Problem 12a:

- Assume you have a 2-way set associative cache.
  - Words are 4 bytes
  - Addresses are to the byte
  - Each block holds 512 bytes
  - There are 1024 blocks in the cache
- If you reference a 32-bit physical address – and the cache is initially empty how many data words are brought into the cache with this reference?
  - The entire block will be filled
  - If words are 4 bytes long and each block holds 512 bytes, there are $2^9/2^2$ words in the block i.e. there are 2^7 or 128 words in each block

Problem 12b

- Which set does the data that is brought in go to if the physical address F A B 1 2 3 8 9 (in hex) is supplied to the cache?
- We need to determine what the index bits are. From 9a, we know the offset is 9 bits (byte addressable) – so we will need to break up the hex address into binary:
- 1111 1010 1011 0001 0010 0011 1000 1001 [block offset in blue]
- 2-way set associative, so # Blocks/2 = # Index = 1024/2 = 512
- $512 = 2^9$ so, 9 bits [in red] used to identify cache index
- The value in red indicates the address maps to the 145^{th} set