Homework and Solutions Part 2

Haifeng Liu

- 4.16-4.20，4.22.1，4.23，4.24，4.25，4.27，4.28
- 5.2.2，5.3.1，5.5，5.6，5.10，5.11.2，5.13，5.16.1，5.17，5.20

4.16 In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250 ps	350 ps	150 ps	300 ps	200 ps

Also, assume that instructions executed by the processor are broken down as follows:

ALU/Logic	Jump/Branch	Load	Store
45%	20%	20%	15%

4.16.1 [5] <§4.5> What is the clock cycle time in a pipelined and non-pipelined processor?

4.16.2 [10] <§4.5> What is the total latency of an ld instruction in a pipelined and non-pipelined processor?

4.16.3 [10] <§4.5> If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

4.16.4 [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the data memory?

4.16.5 [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?

- ➤ 4.16.1   Pipelined: 350;
  non-pipelined: 1250
- ➤ 4.16.2   Pipelined: 350*5=1750;
  non-pipelined: 1250
- ➤ 4.16.3   Split the ID stage.
  This reduces the clock-cycle time to 300ps.
- ➤ 4.16.4   35%.
- ➤ 4.16.5   65%

4.17 [10] <§4.5> What is the minimum number of cycles needed to completely execute n instructions on a CPU with a k stage pipeline? Justify your formula.

- n + k − 1. Let's look at when each instruction is in the WB stage. In a k-stage pipeline, the 1st instruction doesn't enter the WB stage until cycle k. From that point on, at most one of the remaining n − 1 instructions is in the WB stage during every cycle. This gives us a minimum of k + (n − 1) = n + k − 1 cycles.

4.18 [5] <§4.5> Assume that x11 is initialized to 11 and x12 is initialized to 22. Suppose you executed the code below on a version of the pipeline from Section 4.5 that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of registers x13 and x14 be?

```
addi   x11, x12, 5
add    x13, x11, x12
addi   x14, x11, 15
```

- x13 = 33 and x14 = 26

4.19 [10] <§4.5> Assume that x11 is initialized to 11 and x12 is initialized to 22. Suppose you executed the code below on a version of the pipeline from Section 4.5 that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of register x15 be? Assume the register file is written at the beginning of the cycle and read at the end of a cycle. Therefore, an ID stage will return the results of a WB state occurring during the same cycle. See Section 4.7 and Figure 4.51 for details.

```
addi    x11, x12, 5
add     x13, x11, x12
addi    x14, x11, 15
add     x15, x11, x11
```

➧ x15 = 54 (The code will run correctly because the result of the first instruction is written back to the register file at the beginning of the 5th cycle, whereas the final instruction reads the updated value of x1 during the second half of this cycle.)

4.20 [5] <§4.5> Add `NOP` instructions to the code below so that it will run correctly on a pipeline that does not handle data hazards.

```
addi    x11, x12, 5
add     x13, x11, x12
addi    x14, x11, 15
add     x15, x13, x12
```

addi x11, x12, 5

NOP

NOP

add x13, x11, x12

addi x14, x11, 15

NOP

add x15, x13, x12

4.22 [5] <$4.5> Consider the fragment of RISC-V assembly below:
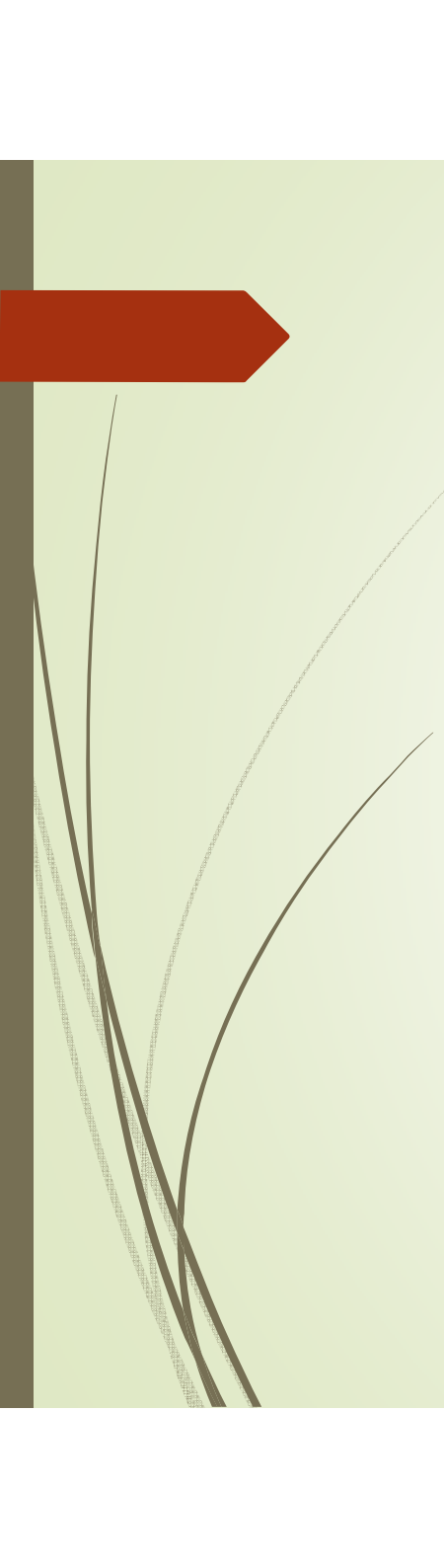
```
sd    x29, 12(x16)
ld    x29, 8(x16)
sub   x17, x15, x14
beqz  x17, label
add   x15, x11, x14
sub   x15, x30, x14
```

Suppose we modify the pipeline so that it has only one memory (that handles both instructions and data). In this case, there will be a structural hazard every time a program needs to fetch an instruction during the same cycle in which another instruction accesses data.

4.22.1 [5] <$4.5> Draw a pipeline diagram to show were the code above will stall.

- 4.22.1 Stalls are marked with **:

```
sd   x29, 12(x16)   IF ID EX  ME WB
ld   x29, 8(x16)        IF  ID  EX  ME WB
sub x17, x15, x14           IF   ID   EX  ME WB
bez x17, label                   **    **   IF    ID  EX ME WB
add x15, x11, x14                                 IF  ID  EX  ME WB
sub x15,x30,x14                                        IF   ID   EX  ME WB
```

4.22.2 [5] <§4.5> In general, is it possible to reduce the number of stalls/NOPs resulting from this structural hazard by reordering code?

4.22.3 [5] <§4.5> 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.

4.22.4 [5] <§4.5> Approximately how many stalls would you expect this structural hazard to generate in a typical program? (Use the instruction mix from Exercise 4.8.)

- 4.22.2 Reordering code won't help. Every instruction must be fetched; thus, every data access causes a stall. Reordering code will just change the pair of instructions that are in conflict.

- 4.22.3 You can't solve this structural hazard with NOPs, because even the NOPs must be fetched from instruction memory.

- 4.22.4 36%. Every data access will cause a stall.

4.23 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 Exercise 4.15.) As a result, the MEM and EX stages can be overlapped and the pipeline has only four stages.

4.23.1 [10] <§4.5> How will the reduction in pipeline depth affect the cycle time?

4.23.2 [5] <§4.5> How might this change improve the performance of the pipeline?

4.23.3 [5] <§4.5> How might this change degrade the performance of the pipeline?

- 4.23.1 The clock period won't change because we aren't making any changes to the slowest stage.

- 4.23.2 Moving the MEM stage in parallel with the EX stage will eliminate the need for a cycle between loads and operations that use the result of the loads. This can potentially reduce the number of stalls in a program.

- 4.23.3 Removing the off set from ld and sd may increase the total number of instructions because some ld and sd instructions will need to be replaced with a addi/ld or addi/sd pair.

4.24 [10] <§4.7> 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
```

Time (in clock cycles)

CC 1　CC 2　CC 3　CC 4　CC 5　CC 6　CC 7　CC 8　CC 9　CC 10

Program execution order (in instructions)
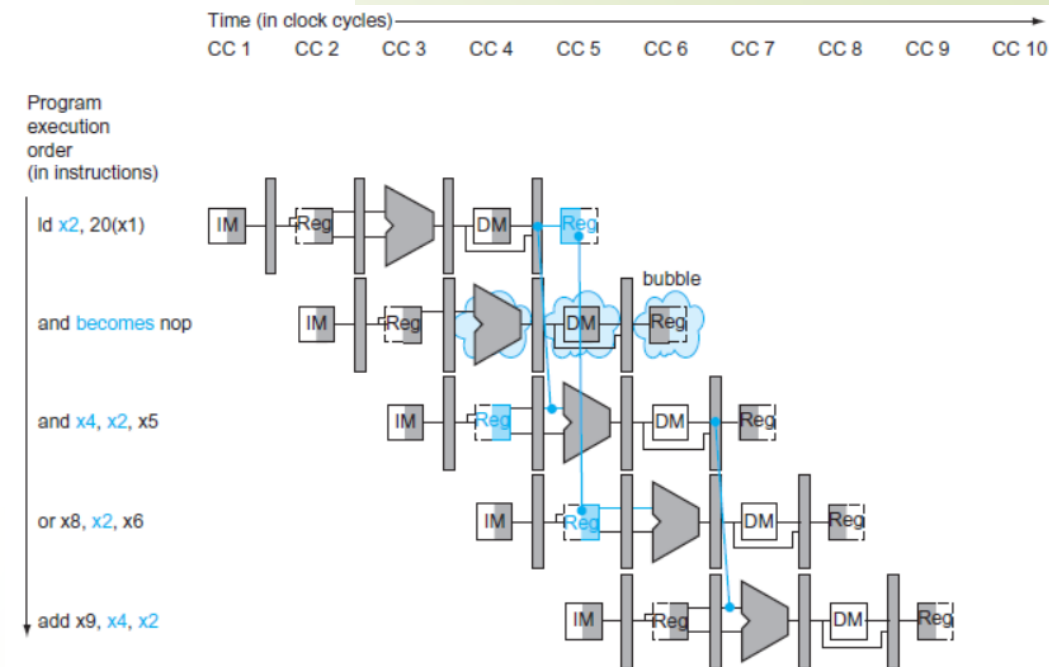
- ➡ The second one. A careful examination of Figure 4.57 shows that the need for a stall is detected during the ID stage. It is this stage that prevents the fetch of a new instruction, effectively causing the add to repeat its ID stage.

4.25 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.

4.25.1 [10] <§4.7> Show a pipeline execution diagram for the first two iterations of this loop.

> 4.25.1 … indicates a stall. ! indicates a stage that does not do useful work.

```
ld x10, 0(x13)      IF ID EX ME | WB
ld x11, 8(x13)         IF  ID  EX |  ME  WB
add x12, x10, x11         IF   ID |  ..   EX  ME!  WB
subi  x13, x13, 16             IF |  ..    ID   EX  ME!  WB
bnez x12, LOOP                    |  ..   IF   ID   EX    ME! WB!
ld x10, 0(x13)                                   IF    ID    EX   ME   WB
ld x11, 8(x13)                                        IF     ID   EX   ME   WB
add x12, x10, x11                                         IF    ID    ..    EX  | ME! WB
subi x13, x13, 16                                             IF      ..    ID | EX    ME! WB
bnez x12, LOOP                                                         IF | ID    EX    ME! WB!
Completely busy               | N    N   N    N     N    N     N     N |
```

4.25.2 [10] <§4.7> 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.)

- 4.25.2 In a particular clock cycle, a pipeline stage is not doing useful work if it is stalled or if the instruction going through that stage is not doing any useful work there. As the diagram above shows, there are not any cycles during which every pipeline stage is doing useful work.

4.27 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)
ld   x12, 0(x2)
or   x13, x15, x13
sd   x13, 0(x15)
```

4.27.1 [5] <§4.7> If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.

➡ 4.27.1 add x15, x12, x11

    nop

    nop

    ld x13, 4(x15)

    ld x12, 0(x2)

    nop

    or x13, x15, x13

    nop

    nop

    sd x13, 0(x15)

4.27.2 [10] <§4.7> 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.

4.27.3 [10] <§4.7> If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?

- 4.27.2 It is not possible to reduce the number of NOPs.

- 4.27.3 The code executes correctly. We need hazard detection only to insert a stall when the instruction following a load uses the result of the load. That does not happen in this case.

4.27.4 [20] <§4.7> If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.59.

4.27.4

Cycle	1	2	3	4	5	6	7	8	
add	IF	ID	EX	ME	WB				
ld		IF	ID	EX	ME	WB			
ld			IF	ID	EX	ME	WB		
or				IF	ID	EX	ME	WB	
sd					IF	ID	EX	ME	WB

Because there are no stalls in this code, PCWrite and IF/IDWrite are always 1 and the mux before ID/EX is always set to pass the control values through.

(1) ForwardA = X; ForwardB = X (no instruction in EX stage yet)

(2) ForwardA = X; ForwardB = X (no instruction in EX stage yet)

(3) ForwardA = 0; ForwardB = 0 (no forwarding; values taken from registers)

(4) ForwardA = 2; ForwardB = 0 (base register taken from result of previous instruction)

(5) ForwardA = 1; ForwardB = 1 (base reguster taken from result of two instructions previous)

(6) ForwardA = 0; ForwardB = 2 (rs1 = x15 taken from register; rs2 = x13 taken from result of 1st ld—two instructions ago)

(7) ForwardA = 0; ForwardB = 2 (base register taken from register file. Data to be written taken from previous instruction)

Datapath with Hazard Detection

Forwarding Conditions

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Set Forwarding Signals

- EX hazard

  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))
    ForwardA = 10

  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))
    ForwardB = 10

- MEM hazard

  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs1))
    ForwardA = 01

  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs2))
    ForwardB = 01

Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs1, IF/ID.RegisterRs2
- Load-use hazard when
  - ID/EX.MemRead and
    ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
    (ID/EX.RegisterRd = IF/ID.RegisterRs2))
- If detected, stall and insert one bubble

4.27.5 [10] <§4.7> If there is no forwarding, what new input and output signals do we need for the hazard detection unit in Figure 4.59? Using this instruction sequence as an example, explain why each signal is needed.

- The hazard detection unit additionally needs the values of rd that comes out of the MEM/WB register. The instruction that is currently in the ID stage needs to be stalled if it depends on a value produced by (or forwarded from) the instruction in the EX or the instruction in the MEM stage. So we need to check the destination register of these two instructions. The Hazard unit already has the value of rd from the EX/MEM register as inputs, so we need only add the value from the MEM/WB register. No additional outputs are needed. We can stall the pipeline using the three output signals that we already have. The value of rd from EX/MEM is needed to detect the data hazard between the add and the following ld . The value of rd form MEM/WB is needed to detect the data hazard between the first ld instruction and the or instruction.

4.27.6 [20] <§4.7> For the new hazard detection unit from 4.26.5, specify which output signals it asserts in each of the first five cycles during the execution of this code.

4.27.6

Cycle	1	2	3	4	5	6
add	IF	ID	EX	ME	WB	
ld		IF	ID	–	–	EX
ld			IF	–	–	ID

(1) PCWrite = 1; IF/IDWrite = 1; control mux = 0

(2) PCWrite = 1; IF/IDWrite = 1; control mux = 0

(3) PCWrite = 1; IF/IDWrite = 1; control mux = 0

(4) PCWrite = 0; IF/IDWrite = 0; control mux = 1

(5) PCWrite = 0; IF/IDWrite = 0; control mux = 1

4.28 The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

R-type	beqz/bnez	jal	ld	sd
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:
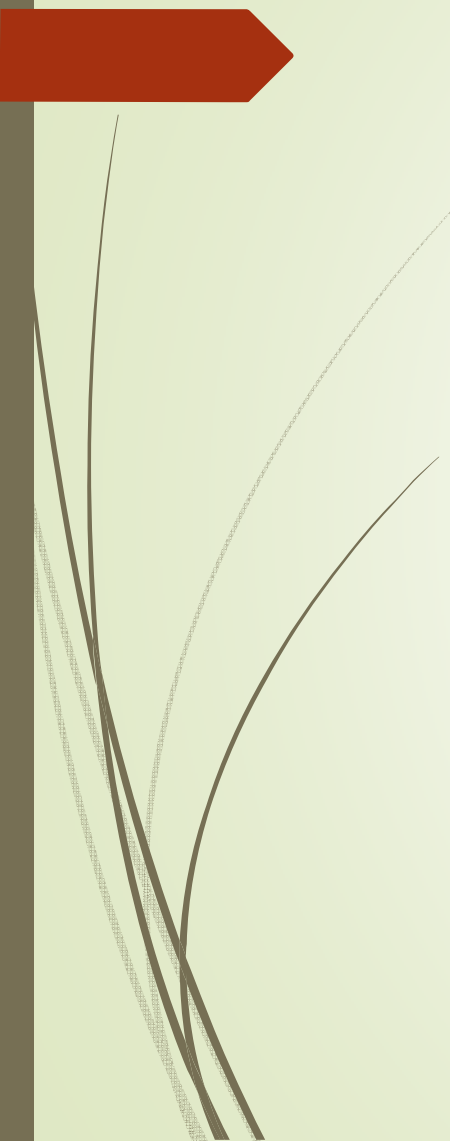
Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

4.28.1 [10] <§4.8> Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the ID stage and applied in the EX stage that there are no data hazards, and that no delay slots are used.

4.28.2 [10] <§4.8> Repeat 4.28.1 for the "always-not-taken" predictor.

4.28.3 [10] <§4.8> Repeat 4.28.1 for the 2-bit predictor.

- 4.28.1 The CPI increases from 1 to 1.4125.

- 4.28.2 The CPI increases from 1 to 1.3375. $(1 + (.25)(1 - .55) = 1.1125)$

- 4.28.3 Th e CPI increases from 1 to 1.1125. $(1 + (.25)(1 - .85) = 1.0375)$

4.28.4 [10] <§4.8> With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions to some ALU instruction? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

4.28.5 [10] <§4.8> With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions in a way that replaced each branch instruction with two ALU instructions? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

4.28.6 [10] <§4.8> Some branch instructions are much more predictable than others. If we know that 80% of all executed branch instructions are easy-to-predict loop-back branches that are always predicted correctly, what is the accuracy of the 2-bit predictor on the remaining 20% of the branch instructions?

- 4.28.4 The speedup is approximately 1.019.

- 4.28.5 The "speedup" is .91.

- 4.28.6 The predictor is 25% accurate on the remaining branches. We know that 80% of branches are always predicted correctly and the overall accuracy is 0.85. Thus, 0.8*1 + 0.2*x = 0.85. Solving for x shows that x = 0.25.
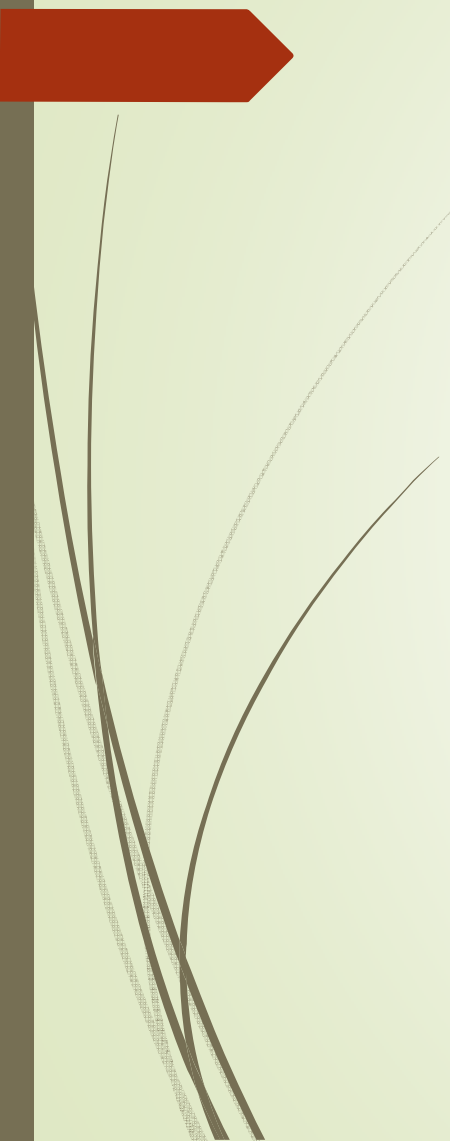
5.2 Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 64-bit memory address references, given as word addresses.

    0x03, 0xb4, 0x2b, 0x02, 0xbf, 0x58, 0xbe, 0x0e, 0xb5,
    0x2c, 0xba, 0xfd

5.2.2 [10] <§5.3> For each of these references, identify the binary word address, the tag, the index, and the offset given a direct-mapped cache with two-word blocks and a total size of eight blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

▶ Solution:

Word Address	Binary Address	Tag	Index	Offset	Hit/Miss
0x03	0000 0011	0	1	1	M
0xb4	1011 0100	b	2	0	M
0x2b	0010 1011	2	5	1	M
0x02	0000 0010	0	1	0	H
0xbf	1011 1111	b	7	1	M
0x58	0101 1000	5	4	0	M
0xbe	1011 1110	b	6	0	H
0x0e	0000 1110	0	7	0	M
0xb5	1011 0101	b	2	1	H
0x2c	0010 1100	2	6	0	M
0xba	1011 1010	b	5	0	M
0xfd	1111 1101	f	6	1	M

5.3 By convention, a cache is named according to the amount of data it contains (i.e., a 4 KiB cache can hold 4 KiB of data); however, caches also require SRAM to store metadata such as tags and valid bits. For this exercise, you will examine how a cache's configuration affects the total amount of SRAM needed to implement it as well as the performance of the cache. For all parts, assume that the caches are byte addressable, and that addresses and words are 64 bits.

5.3.1 [10] <§5.3> Calculate the total number of bits required to implement a 32 KiB cache with two-word blocks.

- Total size is 364,544 bits = 45,568 bytes

    Each word is 8 bytes; each block contains two words; thus, each block contains $16 = 2^4$ bytes.

    The cache contains 32KiB = 2^{15} bytes of data.

    Thus, it has $2^{15}/2^4 = 2^{11}$ lines of data.

    Each 64-bit address is divided into: (1) a 3-bit word off set, (2) a 1-bit block off set, (3) an 11-bit index (because there are 2^{11} lines), and (4) a 49-bit tag (64 − 3 − 1 − 11 = 49).

    The cache is composed of: $2^{15} * 8$ bits of data + $2^{11}*49$ bits of tag + $2^{11}*1$ valid bits = 364,544 bits.

5.5 For a direct-mapped cache design with a 64-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
63–10	9–5	4–0

5.5.1 [5] <§5.3> What is the cache block size (in words)?

5.5.2 [5] <§5.3> How many blocks does the cache have?

5.5.3 [5] <§5.3> What is the ratio between total bits required for such a cache implementation over the data storage bits?

Beginning from power on, the following byte-addressed cache references are recorded.

Address											
Hex	00	04	10	84	E8	A0	400	1E	8C	C1C	B4	884
Dec	0	4	16	132	232	160	1024	30	140	3100	180	2180

- 5.5.1 Each cache block consists of four 8-byte words. The total off set is 5 bits. Three of those 5 bits is the word off set (the off set into an 8-byte word). The remaining two bits are the block off set. Two bits allows us to enumerate 2^2 = 4 words.

- 5.5.2 There are five index bits. This tells us there are 2^5 = 32 lines in the cache.

- 5.5.3 The ratio is 1.21. The cache stores a total of 32 lines * 4 words/block * 8 bytes/ word = 1024 bytes = 8192 bits. In addition to the data, each line contains 54 tag bits and 1 valid bit. Thus, the total bits required = 8192 + 54*32 + 1*32 = 9952 bits.

5.5.4 [20] <§5.3> For each reference, list (1) its tag, index, and offset, (2) whether it is a hit or a miss, and (3) which bytes were replaced (if any).

5.5.5 [5] <§5.3> What is the hit ratio?

5.5.6 [5] <§5.3> List the final state of the cache, with each valid entry represented as a record of <index, tag, data>. For example,

```
<0, 3, Mem[0xC00]-Mem[0xC1F]>
```

▸ 5.5.4:

Byte Address	Binary Address	Tag	Index	Offset	Hit/Miss	Bytes Replaced
0x00	0000 0000 0000	0x0	0x00	0x00	M	
0x04	0000 0000 0100	0x0	0x00	0x04	H	
0x10	0000 0001 0000	0x0	0x00	0x10	H	
0x84	0000 1000 0100	0x0	0x04	0x04	M	
0xe8	0000 1110 1000	0x0	0x07	0x08	M	
0xa0	0000 1010 0000	0x0	0x05	0x00	M	
0x400	0100 0000 0000	0x1	0x00	0x00	M	0x00-0x1F
0x1e	0000 0001 1110	0x0	0x00	0x1e	M	0x400-0x41F
0x8c	0000 1000 1100	0x0	0x04	0x0c	H	
0xc1c	1100 0001 1100	0x3	0x00	0x1c	M	0x00-0x1F
0xb4	0000 1011 0100	0x0	0x05	0x14	H	
0x884	1000 1000 0100	0x2	0x04	0x04	M	0x80-0x9f

▸ 5.5.5: 4/12 = 33%.

▸ 5.5.6:

```
<index, tag, data>
<0, 3, Mem[0xC00]-Mem[0xC1F]>
<4, 2, Mem[0x880]-Mem[0x89f]>
<5, 0, Mem[0x0A0]-Mem[0x0Bf]>
<7, 0, Mem[0x0e0]-Mem[0x0ff]>
```

5.6 Recall that we have two write policies and two write allocation policies, and their combinations can be implemented either in L1 or L2 cache. Assume the following choices for L1 and L2 caches:

L1	L2
Write through, non-write allocate	Write back, write allocate

5.6.1 [5] <§§5.3, 5.8> Buffers are employed between different levels of memory hierarchy to reduce access latency. For this given configuration, list the possible buffers needed between L1 and L2 caches, as well as L2 cache and memory.

5.6.2 [20] <§§5.3, 5.8> Describe the procedure of handling an L1 write-miss, considering the components involved and the possibility of replacing a dirty block.

5.6.3 [20] <§§5.3, 5.8> For a multilevel exclusive cache configuration (a block can only reside in one of the L1 and L2 caches), describe the procedures of handling an L1 write-miss and an L1 read-miss, considering the components involved and the possibility of replacing a dirty block.

- 5.6.1 The L1 cache has a low write miss penalty while the L2 cache has a high write miss penalty. A write buffer between the L1 and L2 cache would hide the write miss latency of the L2 cache. The L2 cache would benefit from write buffers when replacing a dirty block, since the new block would be read in before the dirty block is physically written to memory.

- 5.6.2 On an L1 write miss, the word is written directly to L2 without bringing its block into the L1 cache. If this results in an L2 miss, its block must be brought into the L2 cache, possibly replacing a dirty block, which must first be written to memory.

- 5.6.3 After an L1 write miss, the block will reside in L2 but not in L1. A subsequent read miss on the same block will require that the block in L2 be written back to memory, transferred to L1, and invalidated in L2.

5.10 In this exercise, we will look at the different ways capacity affects overall performance. In general, cache access time is proportional to capacity. Assume that main memory accesses take 70 ns and that 36% of all instructions access data memory. The following table shows data for L1 caches attached to each of two processors, P1 and P2.

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

5.10.1 [5] <$5.4> Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

5.10.2 [10] <$5.4> What is the Average Memory Access Time for P1 and P2 (in cycles)?

5.10.1

- Solution:

P1	1.515 GHz
P2	1.11 GHz

5.10.2

P1	6.31 ns	9.56 cycles
P2	5.11 ns	5.68 cycles

For P1 all memory accesses require at least one cycle (to access L1). 8% of memory accesses additionally require a 70 ns access to main memory. This is 70/0.66 = 106.06 cycles. However, we can't divide cycles; therefore, we must round up to 107 cycles. Thus, the Average Memory Access time is 1 + 0.08*107 = 9.56 cycles, or 6.31 ps.

For P2, a main memory access takes 70 ns. This is 70/0.66 = 77.78 cycles. Because we can't divide cycles, we must round up to 78 cycles. Thus the Average Memory Access time is 1 + 0.06*78 = 5.68 cycles, or 6.11 ps.

5.10.3 [5] <§5.4> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 and P2? Which processor is faster? (When we say a "base CPI of 1.0", we mean that instructions complete in one cycle, unless either the instruction access or the data access causes a cache miss.)

For the next three problems, we will consider the addition of an L2 cache to P1 (to presumably make up for its limited L1 cache capacity). Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.

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

5.10.3

P1	12.64 CPI	8.34 ns per inst
P2	7.36 CPI	6.63 ns per inst

For P1, every instruction requires at least one cycle. In addition, 8% of all instructions miss in the instruction cache and incur a 107-cycle delay. Furthermore, 36% of the instructions are data accesses. 8% of these 36% are cache misses, which adds an additional 107 cycles.

$1 + .08 * 107 + .36 * .08 * 107 = 12.64$

With a clock cycle of 0.66 ps, each instruction requires 8.34 ns.

Using the same logic, we can see that P2 has a CPI of 7.36 and an average of only 6.63 ns/instruction.

5.10.4 [10] <§5.4> What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

5.10.5 [5] <§5.4> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 with the addition of an L2 cache?

5.10.6 [10] <§5.4> What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P1 without an L2 cache?

5.10.7 [15] <§5.4> What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P2 without an L2 cache?

- Solution:

- 5.10.4   AMAT = 9.85 cycles Worse

- 5.10.5   13.04

- 5.10.6   Because the clock cycle time and percentage of memory instructions is the same for both versions of P1, it is sufficient to focus on AMAT. We want AMAT with L2 < AMAT with L1 only 1 + 0.08[9 + m*107] < 9.56. This happens when m< .916.

- 5.10.7   We want P1's average time per instruction to be less than 6.63 ns. This means that we want (CPI_P1 * 0.66) < 6.63. Thus, we need CPI_P1 < 10.05, CPI_P1 = AMAT_P1 + 0.36(AMAT_P1 – 1)

    Thus, we want AMAT_P1+ 0.36(AMAT_P1-1) < 10.05 This happens when AMAT_P1< 7.65. Finally, we solve for 1+ 0.08[9+ m*107] < 7.65 and find that m< 0.693. This miss rate can be at most 69.3%.

5.11 This exercise examines the effect of different cache designs, specifically comparing associative caches to the direct-mapped caches from Section 5.4. For these exercises, refer to the sequence of word address shown below.

```
0x03, 0xb4, 0x2b, 0x02, 0xbe, 0x58, 0xbf, 0x0e, 0x1f,
0xb5, 0xbf, 0xba, 0x2e, 0xce
```

5.11.2 [10] <§5.4> Trace the behavior of the cache from Exercise 5.11.1. Assume a true LRU replacement policy. For each reference, identify

- the binary word address,

- the tag,

- the index,

- the offset

- whether the reference is a hit or a miss, and

- which tags are in each way of the cache after the reference has been handled.

5.11.2 T(x) is the tag at index x.

Word Address	Binary Address	Tag	Index	Offset	Hit/Miss	Way 0	Way 1	Way 2
0x03	0000 0011	0x0	1	1	M	T(1)=0		
0xb4	1011 0100	0xb	2	0	M	T(1)=0 T(2)=b		
0x2b	0010 1011	0x2	5	1	M	T(1)=0 T(2)=b T(5)=2		
0x02	0000 0010	0x0	1	0	H	T(1)=0 T(2)=b T(5)=2		
0xbe	1011 1110	0xb	7	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b		
0x58	0101 1000	0x5	4	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5		
0xbf	1011 1111	0xb	7	1	H	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5		
0x0e	0000 1110	0x0	7	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=0	
0x1f	0001 1111	0x1	7	1	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=0	T(7)=1
0xb5	1011 0101	0xb	2	1	H	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=0	T(7)=1
0xbf	1011 1111	0xb	7	1	H	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=0	T(7)=1
0xba	1011 1010	0xb	5	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=2 T(5)=b	T(7)=1
0x2e	0010 1110	0x2	7	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=2 T(5)=b	T(7)=1
0xce	1100 1110	0xc	7	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=2 T(5)=b	T(7)=c

5.13 *Mean time between failures* (MTBF), *mean time to replacement* (MTTR), and *mean time to failure* (MTTF) are useful metrics for evaluating the reliability and availability of a storage resource. Explore these concepts by answering the questions about a device with the following metrics:

MTTF	MTTR
3 Years	1 Day

5.13.1 [5] <§5.5> Calculate the MTBF for such a device.

5.13.2 [5] <§5.5> Calculate the availability for such a device.

5.13.3 [5] <§5.5> What happens to availability as the MTTR approaches 0? Is this a realistic situation?

5.13.4 [5] <§5.5> What happens to availability as the MTTR gets very high, i.e., a device is difficult to repair? Does this imply the device has low availability?

- 5.13.1   3 years and 1 day = 1096 days = 26304 hours

- 5.13.2   1095/1096 = 99.90875912%

- 5.13.3   Availability approaches 1.0. With the emergence of inexpensive drives, having a nearly 0 replacement time for hardware is quite feasible. However, replacing file systems and other data can take significant time. Although a drive manufacturer will not include this time in their statistics, it is certainly a part of replacing a disk.

- 5.13.4   MTTR becomes the dominant factor in determining availability. However, availability would be quite high if MTTF also grew measurably. If MTTF is 1000 times of MTTR, then the specific value of MTTR is not significant.

5.16 As described in Section 5.7, virtual memory uses a page table to track the mapping of virtual addresses to physical addresses. This exercise shows how this table must be updated as addresses are accessed. The following data constitute a stream of virtual byte addresses as seen on a system. Assume 4 KiB pages, a four-entry fully associative TLB, and true LRU replacement. If pages must be brought in from disk, increment the next largest page number.

Decimal	4669	2227	13916	34587	48870	12608	49225
hex	0x123d	0x08b3	0x365c	0x871b	0xbee6	0x3140	0xc049

TLB

Valid	Tag	Physical Page Number	Time Since Last Access
1	0xb	12	4
1	0x7	4	1
1	0x3	6	3
0	0x4	9	7

Page table

Index	Valid	Physical Page or in Disk
0	1	5
1	0	Disk
2	0	Disk
3	1	6
4	1	9
5	1	11
6	0	Disk
7	1	4
8	0	Disk
9	0	Disk
a	1	3
b	1	12

5.16.1 [10] <§5.7> For each access shown above, list

■ whether the access is a hit or miss in the TLB,

■ whether the access is a hit or miss in the page table,

■ whether the access is a page fault,

■ the updated state of the TLB.

Address	Virtual Page	TLB H/M	TLB		
			Valid	Tag	Physical Page
4669 0x123d	1	TLB miss PT hit PF	1	b	12
			1	7	4
			1	3	6
			1 (last access 0)	1	13
2227 0x08b3	0	TLB miss PT hit	1 (last access 1)	0	5
			1	7	4
			1	3	6
			1 (last access 0)	1	13
13916 0x365c	3	TLB miss PT hit	1 (last access 1)	0	5
			1	7	4
			1 (last access 2)	3	6
			1 (last access 0)	1	13
34587 0x871b	8	TLB miss PT hit PF	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 2)	3	6
			1 (last access 0)	1	13
48870 0xbee6	b	TLB miss PT hit	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 2)	3	6
			1 (last access 4)	b	12
12608 0x3140	3	TLB miss PT hit	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	b	12
49225 0xc040	c	TLB miss PT hit PF	1 (last access 6)	c	15
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	b	12

5.17 There are several parameters that affect the overall size of the page table. Listed below are key page table parameters.

Virtual Address Size	Page Size	Page Table Entry Size
32 bits	8 KiB	4 bytes

5.17.1 [5] <§5.7> Given the parameters shown above, calculate the maximum possible page table size for a system running five processes.

5.17.2 [10] <§5.7> Given the parameters shown above, calculate the total page table size for a system running five applications that each utilize half of the virtual memory available, given a two-level page table approach with up to 256 entries at the 1st level. Assume each entry of the main page table is 6 bytes. Calculate the minimum and maximum amount of memory required for this page table.

5.17.3 [10] <§5.7> A cache designer wants to increase the size of a 4 KiB virtually indexed, physically tagged cache. Given the page size shown above, is it possible to make a 16 KiB direct-mapped cache, assuming two 64-bit words per block? How would the designer increase the data size of the cache?

- 5.17.1 The tag size is $32 - \log_2(8192) = 32 - 13 = 19$ bits. All five page tables would require $5 \times (2^{19} \times 4)$ bytes = 10 MB.

- 5.17.2 In the two-level approach, the 2^{19} page table entries are divided into 256 segments that are allocated on demand. Each of the second-level tables contains $2^{(19-8)} = 2048$ entries, requiring $2048 \times 4 = 8$ KB each and covering 2048×8 KB = 16 MB (2^{24}) of the virtual address space. Chapter 5 Solutions S-19 If we assume that "half the memory" means 2^{31} bytes, then the minimum amount of memory required for the second-level tables would be $5 \times (2^{31}/2^{24})*8$ KB = 5 MB. The first-level tables would require an additional $5 \times 128 \times 6$ bytes = 3840 bytes. The maximum amount would be if all 1st-level segments were activated, requiring the use of all 256 segments in each application. This would require $5 \times 256 \times 8$ KB = 10 MB for the second-level tables and 7680 bytes for the first-level tables.

- 5.17.3 The page index is 13 bits (address bits 12 down to 0). A 16 KB direct-mapped cache with two 64-bit words per block would have 16-byte blocks and thus 16 KB/16 bytes = 1024 blocks. Thus, it would have 10 index bits and 4 off set bits and the index would extend outside of the page index. The designer could increase the cache's associativity. This would reduce the number of index bits so that the cache's index fits completely inside the page index.

5.20 In this exercise, we will examine how replacement policies affect miss rate. Assume a two-way set associative cache with four one-word blocks. Consider the following word address sequence: 0, 1, 2, 3, 4, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0.

Consider the following address sequence: 0, 2, 4, 8, 10, 12, 14, 16, 0

5.20.1 [5] <§§5.4, 5.8> Assuming an LRU replacement policy, which accesses are hits?

5.20.2 [5] <§§5.4, 5.8> Assuming an MRU (*most recently used*) replacement policy, which accesses are hits?

5.20.3 [5] <§§5.4, 5.8> Simulate a random replacement policy by flipping a coin. For example, "heads" means to evict the first block in a set and "tails" means to evict the second block in a set. How many hits does this address sequence exhibit?

5.20.4 [10] <§§5.4, 5.8> Describe an optimal replacement policy for this sequence. Which accesses are hits using this policy?

5.20.5 [10] <§§5.4, 5.8> Describe why it is difficult to implement a cache replacement policy that is optimal for all address sequences.

5.20.6 [10] <§§5.4, 5.8> Assume you could make a decision upon each memory reference whether or not you want the requested address to be cached. What effect could this have on miss rate?

- 5.20.1 There are no hits.

- 5.20.2 Direct mapped 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 M M M M M M M M M H H M M M M M H H M

- 5.20.3 Answers will vary.

- 5.20.4 MRU is an optimal policy.

- 5.20.5 The best block to evict is the one that will cause the fewest misses in the future. Unfortunately, a cache controller cannot know the future! Our best alternative is to make a good prediction.

- 5.20.6 If you knew that an address had limited temporal locality and would conflict with another block in the cache, choosing not to cache it could improve the miss rate. On the other hand, you could worsen the miss rate by choosing poorly which addresses to cache.