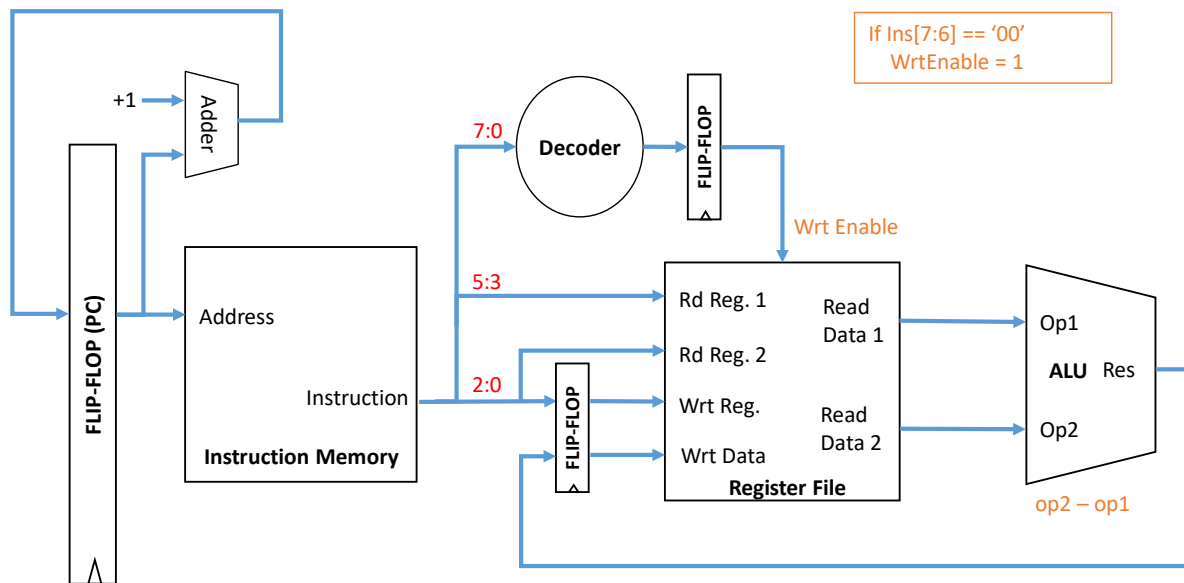


## Homework 2

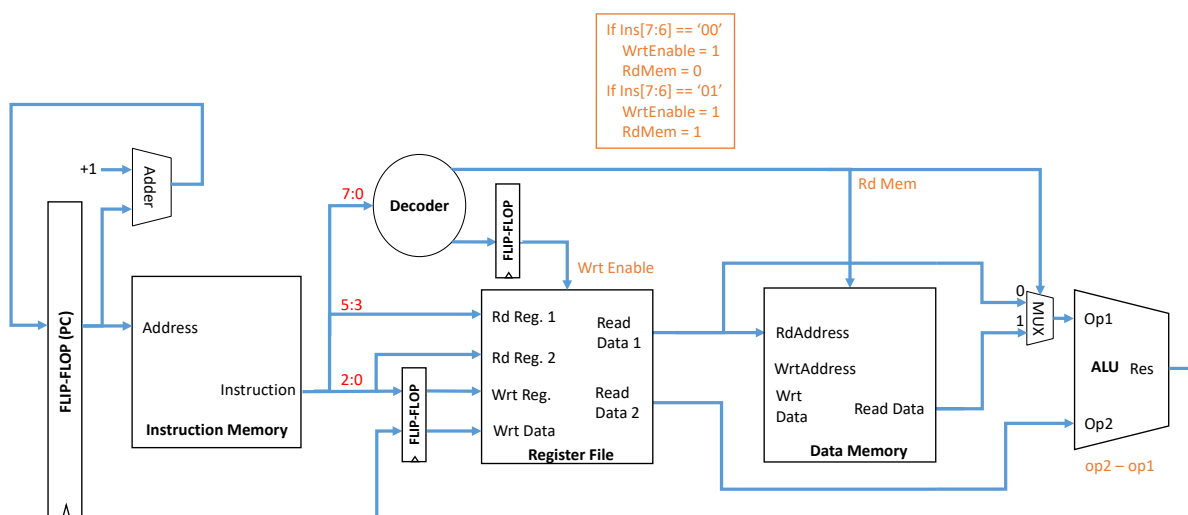
CS-GY 6133

Bo Yao, by677  
Tianyu Gu, tg1553



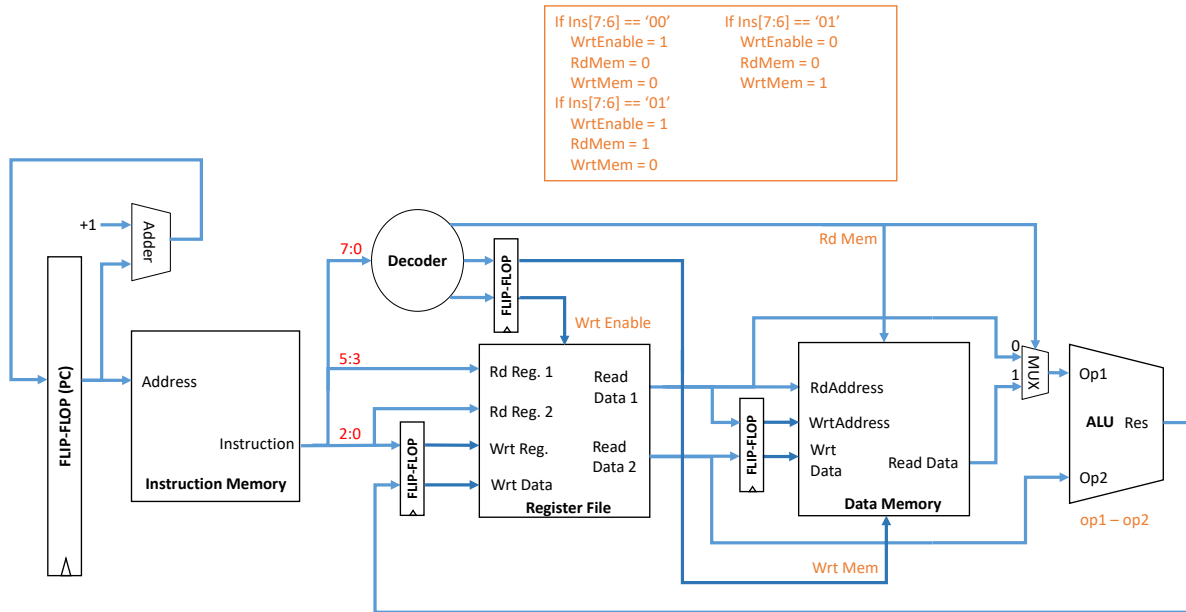
### A1.2

Supporting 'sub', 'subm' instructions



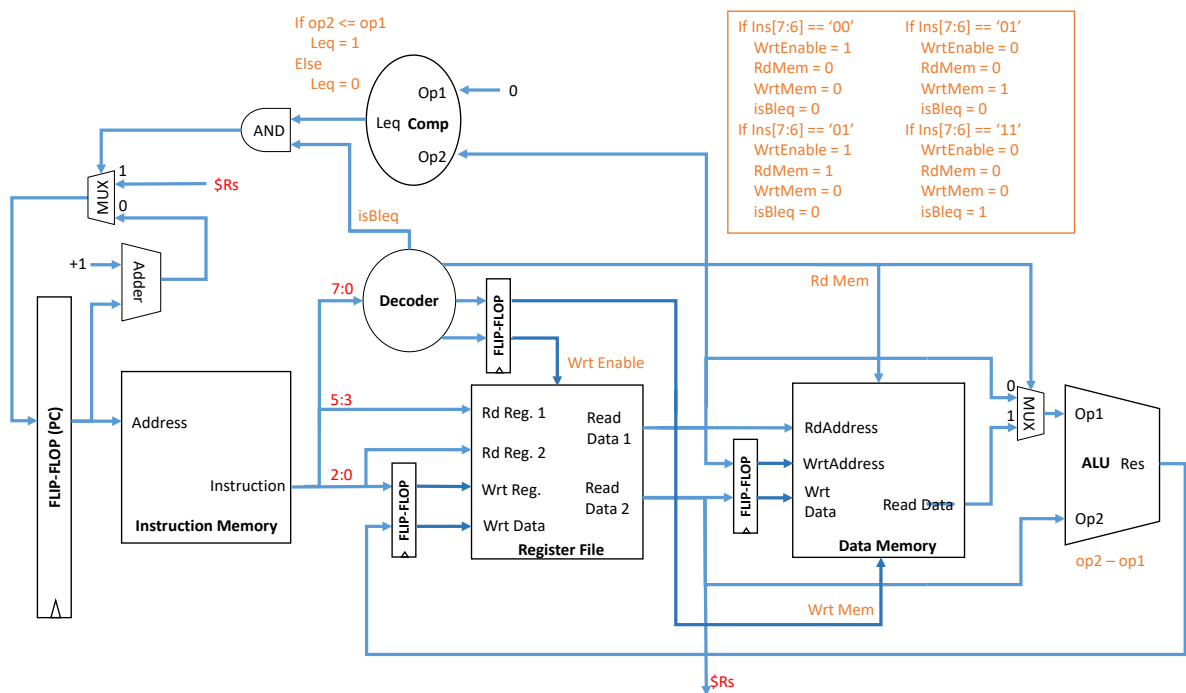
### A1.3

Supporting 'sub', 'subm', 'st' instructions



### A1.4

Supporting 'sub', 'subm', 'st', 'bleq' instructions



### A1.5

The path 'IM-RF-DM-MUX-ALU' takes the longest time (4.5ns). So the maximum frequency is 222.2 MHz.

## Q2. Pipelined CSGY6133

The table below shows how these three instructions access the registers in different states. As we known, the RAW hazard only happens when the first instruction writes back to the same register/memory-block that the second instruction reads. Therefore, we only have to consider about the *rs* of 'sub' and 'subm' in the first instruction. As to 'st' instruction, the data will be update immediately at next positive clock edges. It's like processing 'RF' and 'WB' at the same time. So 'st' won't cause RAW hazard at all.

	Read		Writeback	
	Mem	EX	Mem	WB
<b>sub</b> <i>rd, rs</i>	/	<i>rd, rs</i>	/	<i>rs</i>
<b>subm</b> <i>rd, rs</i>	<i>rd</i>	<i>rs, Mem[rd]</i>	/	<i>rs</i>
<b>st</b> <i>rd, rs</i>	<i>rd, rs</i>	/	<i>Mem[rd]</i>	/

The table below shows all possible pairs that will cause the RAW hazard.

1.	<b>sub</b> \$1, \$2	RAW Hazard	Resolving
2.	<b>sub</b> \$2, \$3	Yes	EX-EX forwarding
	<b>sub</b> \$3, \$2	Yes	EX-EX forwarding
	<b>sub</b> \$2, \$2	Yes	EX-EX forwarding
	<b>subm</b> \$2, \$3	Yes	Stalling before EX, EX-Mem forwarding
	<b>subm</b> \$3, \$2	Yes	EX-EX forwarding
	<b>subm</b> \$2, \$2	Yes	Stalling before EX, EX-Mem forwarding <sup>1</sup>
	<b>st</b> \$2, \$3	Yes	Stalling before EX, EX-Mem forwarding
	<b>st</b> \$3, \$2	Yes	Stalling before EX, EX-Mem forwarding
	<b>st</b> \$2, \$2	Yes	Stalling before EX, EX-Mem forwarding

1.	<b>subm</b> \$1, \$2	RAW Hazard	Resolving
2.	<b>sub</b> \$2, \$3	Yes	EX-EX forwarding
	<b>sub</b> \$3, \$2	Yes	EX-EX forwarding
	<b>sub</b> \$2, \$2	Yes	EX-EX forwarding
	<b>subm</b> \$2, \$3	Yes	Stalling before EX, EX-Mem forwarding
	<b>subm</b> \$3, \$2	Yes	EX-EX forwarding
	<b>subm</b> \$2, \$2	Yes	Stalling before EX, EX-Mem forwarding <sup>1</sup>

<b>st</b> \$2, \$3	Yes	Stalling before EX, EX-Mem forwarding
<b>st</b> \$3, \$2	Yes	Stalling before EX, EX-Mem forwarding
<b>st</b> \$2, \$2	Yes	Stalling before EX, EX-Mem forwarding

**Notes:** 1. **sub** \$1, \$2  
**subm** \$2, \$2


If both  $rd$  and  $rs$  in 'subm' have RAW dependencies, like the example above, theoretically both EX-EX and EX-Mem forwarding are required in two sequential clock cycles. But practically only one EX-Mem forwarding can resolve the RAW dependencies at all. We can forward  $rs$  (\$2 in example) to the Mem state resolving the RAW dependency of  $rd$  first. Then use the flip-flop after the Mem state, to forward  $rs$  to the EX state in the next clock cycle resolving the RAW dependency of  $rs$  in that state.

### Timing diagrams:

1. All timing diagrams of RAW dependencies which require an EX-EX forwarding to resolve the hazard are the same. The diagram is shown below:

EX-EX Forwarding


Cycle	1	2	3	4	5	6
<b>sub</b> \$1, \$2	IF	RF/ID	Mem	EX	WB	
<b>sub</b> \$2, \$3		IF	RF/ID	Mem	EX	WB



2. All timing diagrams of RAW dependencies which require an EX-Mem forwarding to resolve the hazard are the same. The diagram is shown below:

EX-Mem Forwarding (with stalling)

Cycle	1	2	3	4	5	6	7
<b>sub</b> \$1, \$2	IF	RF/ID	Mem	EX	WB		
<b>st</b> \$2, \$3		IF	RF/ID	Mem	Mem	EX	WB



3. As to the situation where both EX-EX and EX-Mem forwarding are required, the timing diagram is shown below. The orange arrow illustrates the path while using only the EX-Mem forwarding.

EX-EX and EX-Mem Forwarding (with stalling)

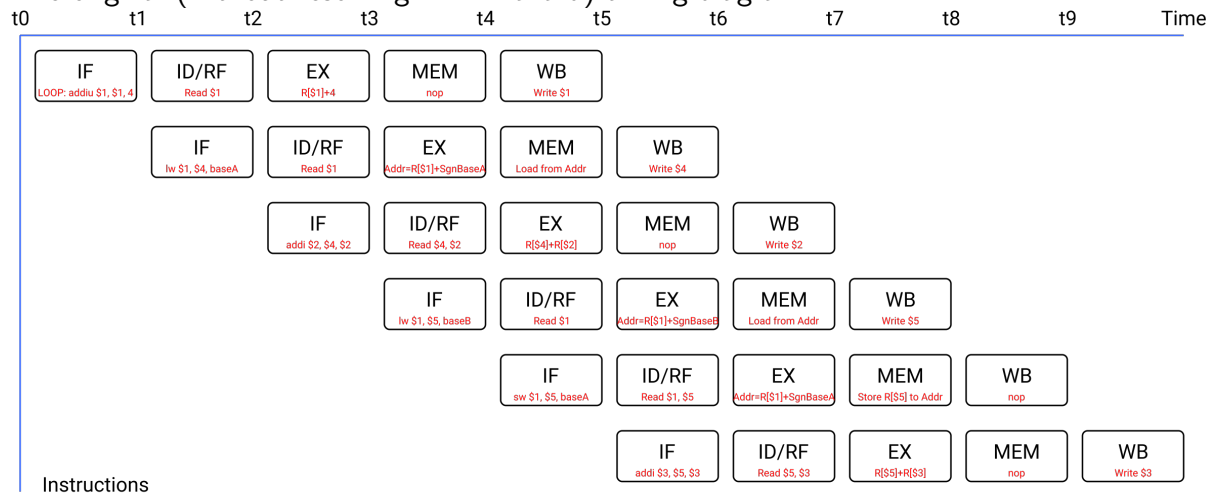
Cycle	1	2	3	4	5	6	7
<b>sub</b> \$1, \$2	IF	RF/ID	Mem	EX	WB		
<b>sub</b> \$2, \$3		IF	RF/ID	Mem	Mem	EX	WB



### Q3. Pipelined MIPS

#### A3.1

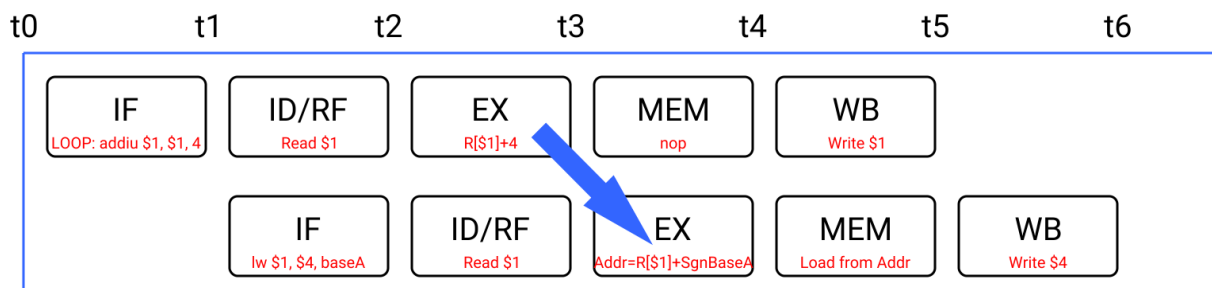
The original (without resolving RAM hazard) timing diagram:



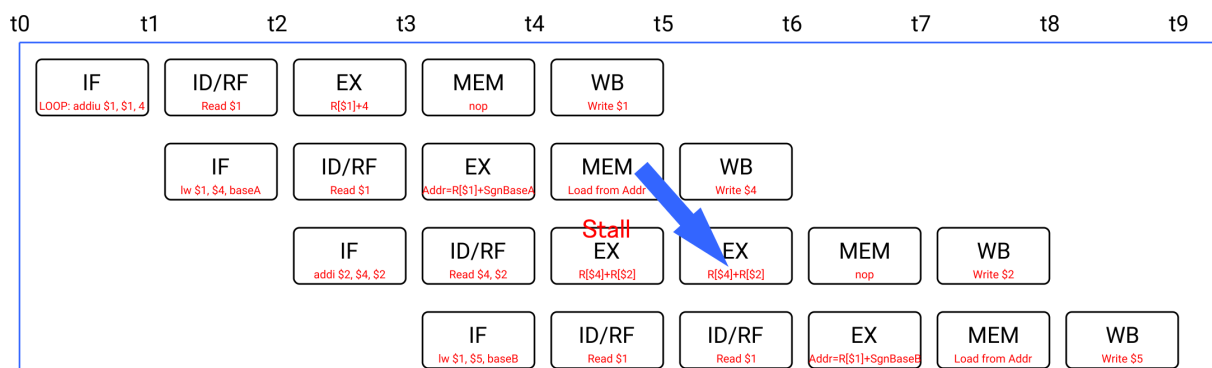
Possible RAM hazards:

1. The Write \$1 in the Instruction 1 (LOOP: ...) and the Read \$1 in the Instruction 2.
2. The Write \$4 in the Instruction 2 and the Read \$4, \$2 in the Instruction 3.
3. The Write \$5 in the Instruction 4 and the Read \$1, \$5 in the Instruction 5.
4. The Write \$5 in the Instruction 4 and the Read \$5 in the Instruction 6.

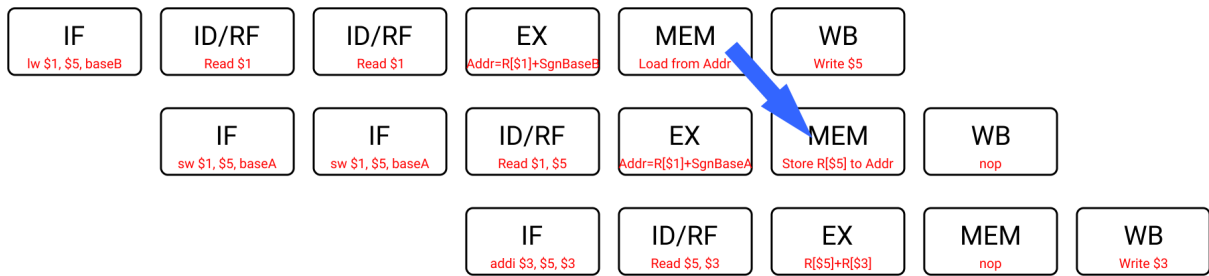
Solution for 1: EX-EX forwarding



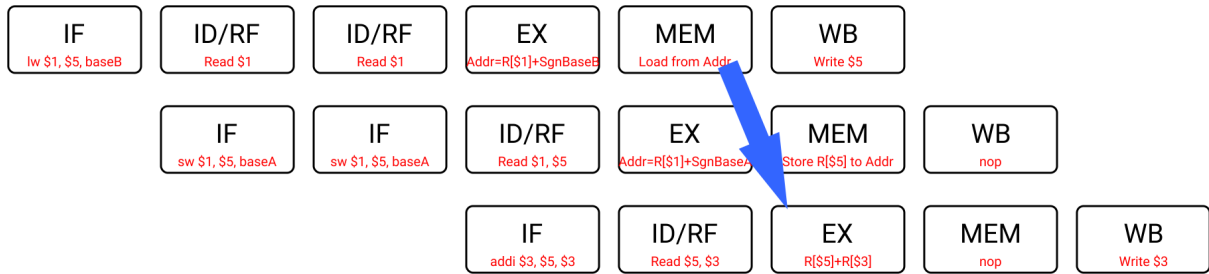
Solution for 2: Stall and MEM-EX forwarding



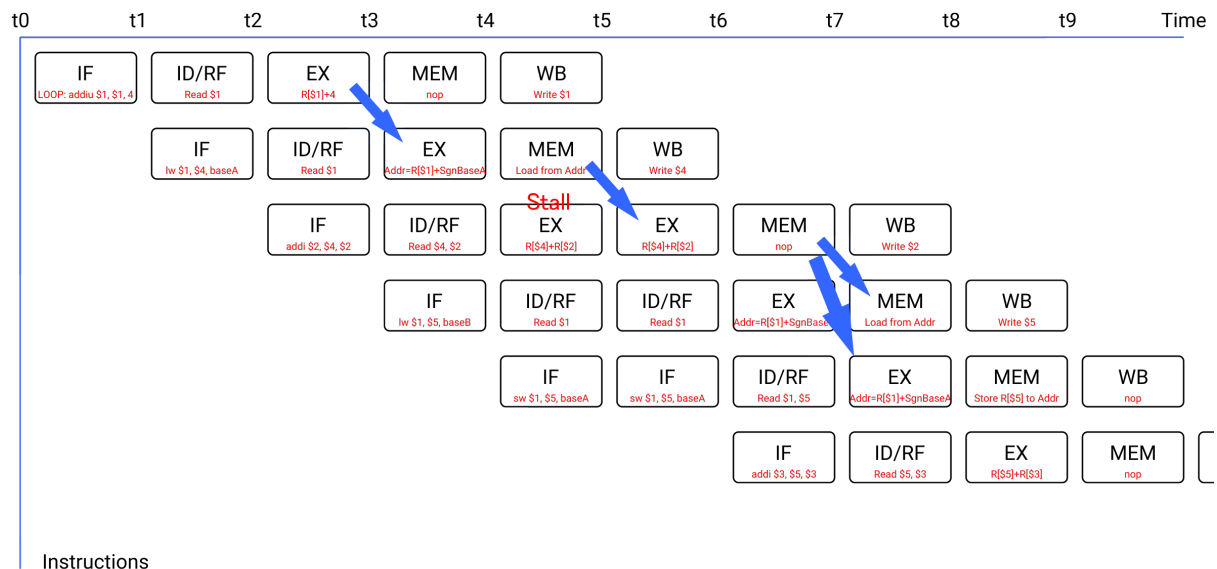
Solution for 3: MEM-MEM forwarding



Solution for 4: MEM-EX forwarding



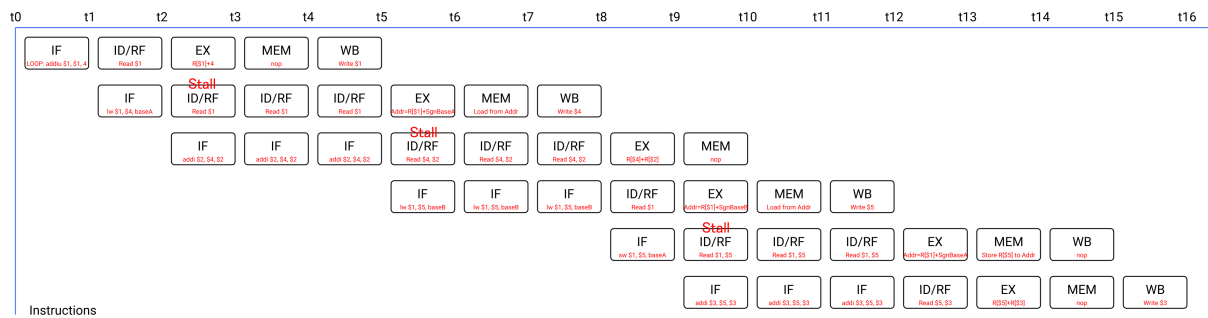
### A3.2



### A3.3

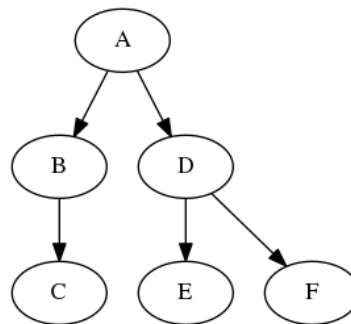
If only consider first six instructions, the clocking cycles are from time  $t_0$  to time  $t_{10}$ . It's 10 clock cycles. Then add the *bne* instruction and two *nops*. The next round iteration start at time  $t_{10}$ . In the last iteration, the final *bne* end 2 clock cycles after the *sw* instruction and no *nops* needed. Thus, the total clocking cycles are:  $10 \times 10 + 2 = 102$ .

## A3.4



## A3.5

Name the first six instruction (exclude the *bne* instruction)  $A, B, \dots, F$ , respectively and since only stalls allowed, store to memory/load from memory will not cause RAW hazard, so we can take each instruction as an abstract object only read and write from some register is taken into consideration. It reads from some register and writes to some registers two clock cycles later. If there is a dependency on its reading from the previous writing, then it has to wait for the previous writing time. Consider each instruction's semantic, the following sequence must be preserved to keep it do the same function:



Also because only stalls allowed, it's impossible that an instruction will and its previous instruction form RAW hazard, only consecutive instructions can form. Thus we use a simple program to fully list all possible arrangement. It's OK in this only five instructions need to be arrange (A is known to be the first) but more efficient algorithm is need for practical use. The final result is F C is put last and the first three can be B D E, D B E and D E B: Thus the results are:

```

LOOP: addiu $1, $1, 4
lw $1, $4, baseA
lw $1, $5, baseB
sw $1, $5, baseA
addi $3, $5, $3
addi $2, $4, $2
bne $1, $6, LOOP

```

or:

```

LOOP: addiu $1, $1, 4
lw $1, $5, baseB
lw $1, $4, baseA
sw $1, $5, baseA

```

```

addi $3, $5, $3
addi $2, $4, $2
bne $1, $6, LOOP

```

or:

```

LOOP: addiu $1, $1, 4
lw $1, $5, baseB
sw $1, $5, baseA
lw $1, $4, baseA
addi $3, $5, $3
addi $2, $4, $2
bne $1, $6, LOOP

```

All three tie in running time and can be two clock cycles faster than original sequence.

## Q4. Caches

### A4.1

1GB byte-addressable memory need 30 bits address.

(a) 8KB cache with 1 byte blocks can contain 8192 blocks.  $8192 = 2^{13}$  so we need 13 bits for index and 17 bits tag, 0 bit offset.

(b) 8KB cache with 4 byte blocks can contain 2048 blocks.  $2048 = 2^{11}$  so we need 11 bits for index and 2 bits offset. The rest 17 bits for tag.

(c) 8 Byte blocks so offset is 3 bits. 2-way and 8 byte so  $8 \times 1024 \div (8 \times 2) = 512 = 2^9$  blocks per way so we need 9 bits for index and  $30 - 9 - 3 = 18$  bits for tag.

(d) 16 Byte blocks so offset is 4 bits. 4-way and 16 byte so  $8 \times 1024 \div (16 \times 4) = 128 = 2^7$  blocks per way so we need 7 bits for index and  $30 - 7 - 4 = 19$  bits for tag.

(e) For Fully-associative cache, there are as many ways as blocks. 8KB cache with 8 byte blocks contains  $8 \times 1024 \div 8 = 1024$  blocks. So fully-associative cache has 1024-way set associative. 8 byte blocks so offset is 3 bits. One block per way so 0 bit for index and  $30 - 3 = 27$  bits for tag.

To sum up:

Question	Offset bits	Index bits	Tag bits
(a)	0	13	17
(b)	2	11	17
(c)	3	9	18
(d)	4	7	19
(e)	3	0	27



**A4.2**

Accessing Address	Hit or Miss State	Memory State after this access
<b>0</b>	Compulsory Miss	0 1
<b>1</b>	Hit	Not Changed
<b>2</b>	Compulsory Miss	0 1
		2 3
<b>3</b>	Hit	Not Changed
<b>4</b>	Compulsory Miss	0 1
		2 3
		4 5
<b>5</b>	Hit	Not Changed
<b>6</b>	Compulsory Miss	0 1
		2 3
		4 5
		6 7
<b>7</b>	Hit	Not Changed
<b>8</b>	Capacity Miss	8 9
		2 3
		4 5
		6 7
<b>9</b>	Hit	Not Changed
<b>0</b>	Capacity Miss	0 1
		2 3
		4 5
		6 7
<b>1</b>	Hit	Not Changed
<b>2</b>	Hit	Not Changed
<b>3</b>	Hit	Not Changed
<b>4</b>	Hit	Not Changed
<b>5</b>	Hit	Not Changed
<b>6</b>	Hit	Not Changed
<b>7</b>	Hit	Not Changed
<b>8</b>	Capacity Miss	8 9
		2 3
		4 5
		6 7
<b>9</b>	Hit	Not Changed

## A4.3

Accessing Address	Hit or Miss State	Memory State after this access	Victim Cache State												
0	Compulsory Miss	<table><tr><td>0</td><td>1</td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>	0	1							<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>				
0	1														
1	Hit in main cache	Not Changed	Not Changed												
2	Compulsory Miss	<table><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>	0	1	2	3					Not Changed				
0	1														
2	3														
3	Hit in main cache	Not Changed	Not Changed												
4	Compulsory Miss	<table><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td></tr><tr><td></td><td></td></tr></table>	0	1	2	3	4	5			Not Changed				
0	1														
2	3														
4	5														
5	Hit in main cache	Not Changed	Not Changed												
6	Compulsory Miss	<table><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td></tr></table>	0	1	2	3	4	5	6	7	Not Changed				
0	1														
2	3														
4	5														
6	7														
7	Hit in main cache	Not Changed	Not Changed												
8	Conflict Miss	<table><tr><td>8</td><td>9</td></tr><tr><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td></tr></table>	8	9	2	3	4	5	6	7	<table><tr><td>0</td><td>1</td></tr><tr><td></td><td></td></tr></table>	0	1		
8	9														
2	3														
4	5														
6	7														
0	1														
9	Hit in main cache	Not Changed	Not Changed												
0	Hit in the victim cache	<table><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td></tr></table>	0	1	2	3	4	5	6	7	<table><tr><td>8</td><td>9</td></tr><tr><td></td><td></td></tr></table>	8	9		
0	1														
2	3														
4	5														
6	7														
8	9														
1	Hit in main cache	Not Changed	Not Changed												
2	Hit in main cache	Not Changed	Not Changed												
3	Hit in main cache	Not Changed	Not Changed												
4	Hit in main cache	Not Changed	Not Changed												
5	Hit in main cache	Not Changed	Not Changed												
6	Hit in main cache	Not Changed	Not Changed												
7	Hit in main cache	Not Changed	Not Changed												
8	Hit in the victim cached	<table><tr><td>8</td><td>9</td></tr><tr><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td></tr></table>	8	9	2	3	4	5	6	7	<table><tr><td>0</td><td>1</td></tr><tr><td></td><td></td></tr></table>	0	1		
8	9														
2	3														
4	5														
6	7														
0	1														
9	Hit in main cache	Not Changed	Not Changed												