

HOMEWORK V

DUE : November 17, 2011

READ :

- i) Related portions of Chapter 4 (except Section 4.4)
- ii) Related portions of Appendix B
- iii) Related portions of Appendix C
- iv) Related portions of Appendix D

ASSIGNMENT : There are **five** questions

Solve all homework and exam problems as shown in class and past exam solutions.

1) Consider the EMY CPU design with nine (9) integer instructions. We decide to expand the design by adding a MIPS instruction that is already in the MIPS architecture : **ADDI** (Add Immediate). Modify the EMY CPU completely to run the ADDI instruction.

Assume that the EMY CPU is a multicycle CPU which is **microprogrammed** and executes those nine instructions in the EMY CPU Handout. You will completely modify the CPU of the handout for the **new** instruction : ADDI ! That is, you will modify the **high-level state diagram**, the **datapath**, the **low-level state diagram** and the **control unit** so that it can execute ADDI.

In order to do that you will follow steps III(b) and IV in the Digital System Design Basics hand-out. That is, you need to modify the **high-level state diagram** (not in terms of buses) in parallel with the modification of the **datapath**. Then, you will modify the **low-level state diagram**. If you decide to add new states, start at state 16. Finally, you will modify the **microprogrammed control unit**. You will state, if you need to change the microinstruction format and/or the hardware of the control unit. Then, you will write down the modified portion of the microcode to execute the ADDI instruction. Since, it is the microcode, you will show only 1's and 0's.

2) Consider the following CISC instruction : DECM. It decrements a memory location. The syntax, semantics, format and other information about the instruction are as follows :

- The **syntax** of the new instruction : DECM Disp(Rs)
- The **semantics** of the new instruction : $M[Rs + Disp^+] <--- M[Rs + Disp^+] - 1$

- The format is the **I format** :

Opcode	Rs	Rt	DOImm
6	5	5	16

Opcode = 17
Rt is **not** used

Three arguments are used by the instruction : The destination and the first source arguments are memory arguments. We use the **2-byte signed displacement** addressing mode for them. The other source argument is always $(-1)_{10}$. We use the **Implied** addressing mode for it.

Modify the EMY CPU completely ! Assume that the CPU is **microprogrammed** and executes those nine instructions in the EMY CPU Handout. It asks you to completely modify the CPU of the handout for a **new** instruction : DECM ! That is, you will modify the **high-level state diagram**, the **datapath**, the **low-level state diagram** and the **control unit** so that it can execute the DECM instruction.

You will state, if you need to change the microinstruction format and/or the hardware of the control unit. Then, you will write down the **modified** portion of the **microcode** to execute the DECM instruction. Since, it is the microcode, you will show only 1's and 0's. If you decide to add **new** states, start at state 16.

3) Consider the following EMY mnemonic machine language program :

```

400200 LW          R8, 0(R4)          # R4 initially has 10000000
400204 LW          R9, 4(R4)
400208 LW          R10, 0(R8)
40020C LW          R11, 4(R8)
400210 ADDI        R9, R9, (-1)10
400214 ADD         R13, R11, R10
400218 BNE         R9, R0, (-5)10
40021C ADDI        R8, R8, 8
400220 JR          R31
400224 SW          R13, 8(R4)
---              ---
10000000 10000400
10000004 2
10000008 ?

```

The EMY CPU is pipelined as explained in class : It has forwardings, delayed branches and write-in-the-first-half, read-in-the-second-half register usage. Assume also that there is a perfect memory, with **no** stalls. Finally, assume that the ADDI takes five clock periods.

The above code is written for the **pipelined** EMY CPU since independent instructions are placed after the LW and BEQ instructions.

The JR instruction takes two clock periods as branch instructions do. The delayed branch concept is also applied to the JR. Thus, the instruction that immediately follows the JR is executed then the instruction pointed by the JR effective address is executed (the instruction pointed by R31).

Show the execution of the loop for which you show all the **necessary** forwarding, register-reading-writing and resolved data hazard types as done in class.

4) Consider the following EMY mnemonic machine language program :

```

400000  LW      R9, 0(R8)          # R8 has 10000000 initially
400004  ADDI   R9, R9, 1
400008  SW      R9, 0(R8)
40000C  ADDI   R8, R8, 4
400010  ADDI   R10, R10, (-1)10  # R10 is already initialized
400014  BNE   R10, R0, (-6)10

```

The EMY CPU is pipelined as explained in class : It has forwardings, delayed branches and write-in-the-first-half, read-in-the-second-half register usage. Assume also that there is a perfect memory, with **no** stalls. Finally, assume that the ADDI takes five clock periods.

The code above is written for the **unpipelined** EMY CPU since there are **no** independent instructions after the LW and BEQ instructions.

i) Reorder the instructions to place independent instructions after the load instruction and in the delayed branch slot. Note that the number of instructions after the reordering will be still six.

ii) Show the execution of the loop for **two** iterations for which you show all the **necessary** forwarding, register-reading-writing and resolved data hazard types as done in class.

iii) Calculate the number of clock periods it takes to run the loop for **100** iterations.

5) Consider the following EMY program :

```

400100  LW      R8, 0(R4)          # R4 has 10000000 initially. R8 gets 0
400104  LW      R9, 1000(R4)      # Read from array B
400108  LW      R10, 2000(R4)    # Read from array C
40010C  ADD     R11, R10, R9
400110  SW      R11, 3000(R4)    # Store to array A
400114  ADDI   R4, R4, 4
400118  ADDI   R8, R8, (-1)10
40011C  SLTI   R12, R8, 0
400120  BEQ   R12, R0, (-8)10
400124  JR     R31

```

The EMY CPU is pipelined as explained in class : It has forwardings, delayed branches and write-in-the-first-half, read-in-the-second-half register usage. Assume also that there is a perfect memory, with **no** stalls. Finally, assume that the ADDI takes five clock periods.

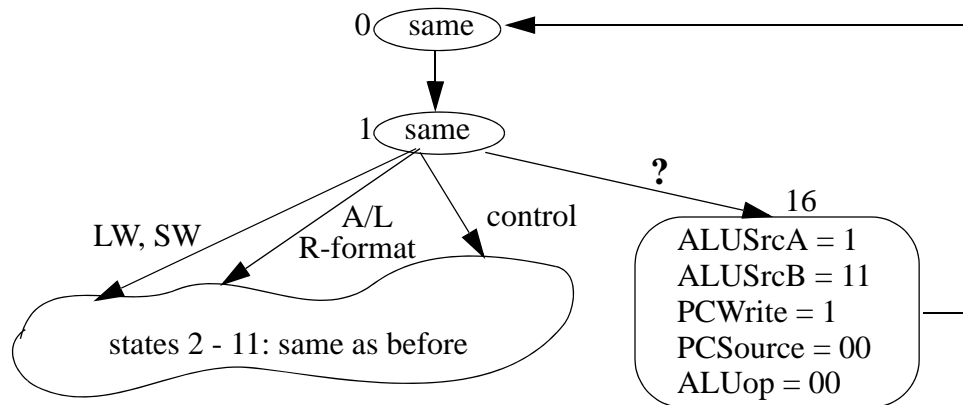
The code is written for the **unpipelined** EMY CPU since there are no independent instructions after the LW and BEQ instructions. Note that, the ADDI and SLTI instructions take five clock periods as R-type A/L instructions do.

i) Reorder the instructions to place as many independent instructions as possible after the load instruction and in delayed branch/jr slots.

ii) Clearly show in which clock period the last cycle of the instruction that follows the JR instruction is performed as done in class, i.e. together with all the **necessary** forwarding, register-reading-writing and resolved data hazard types as done in class.

RELEVANT QUESTIONS AND ANSWERS

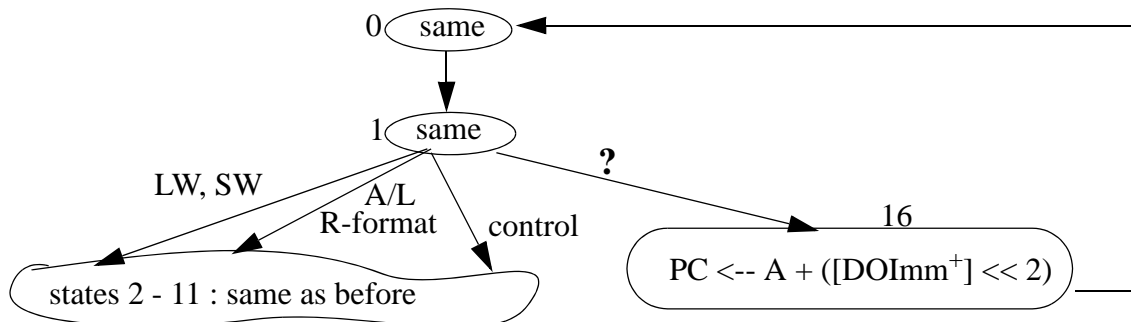
Q1) The EMY CPU is designed as shown in class. Its control unit is microprogrammed ! The CPU is modified due to an architectural change and the resulting low-level state diagram is as follows :

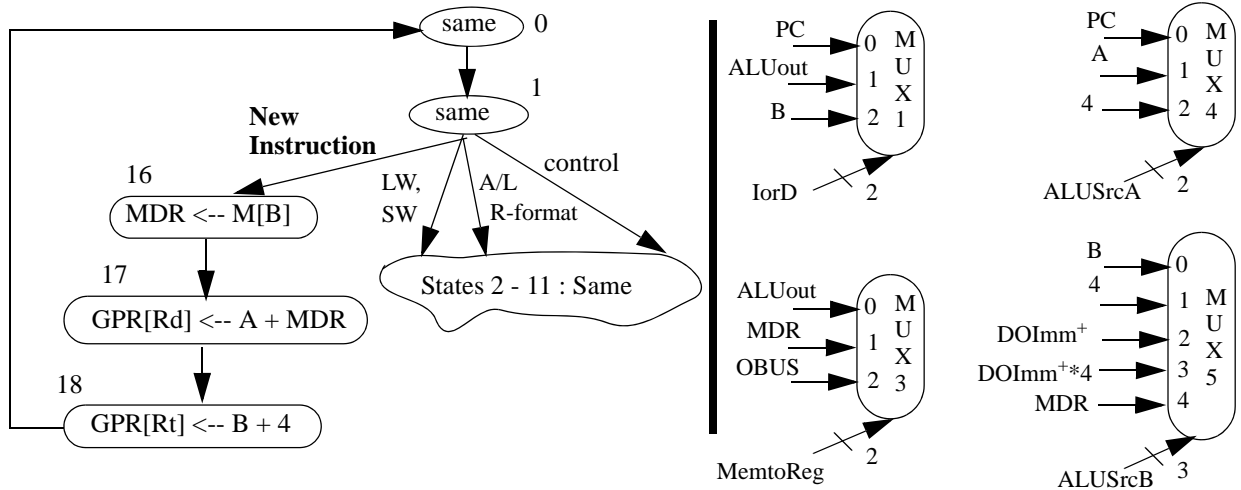


- , What is the corresponding high-level state diagram ?
- , Draw the modified control unit which is still microprogrammed.

Then, you will write down the **modified** portion of the **microcode** to execute the ADDI instruction. Since, it is the microcode, you will show only 1's and 0's.

A1) The modified portion of the high-level state diagram is as follows :





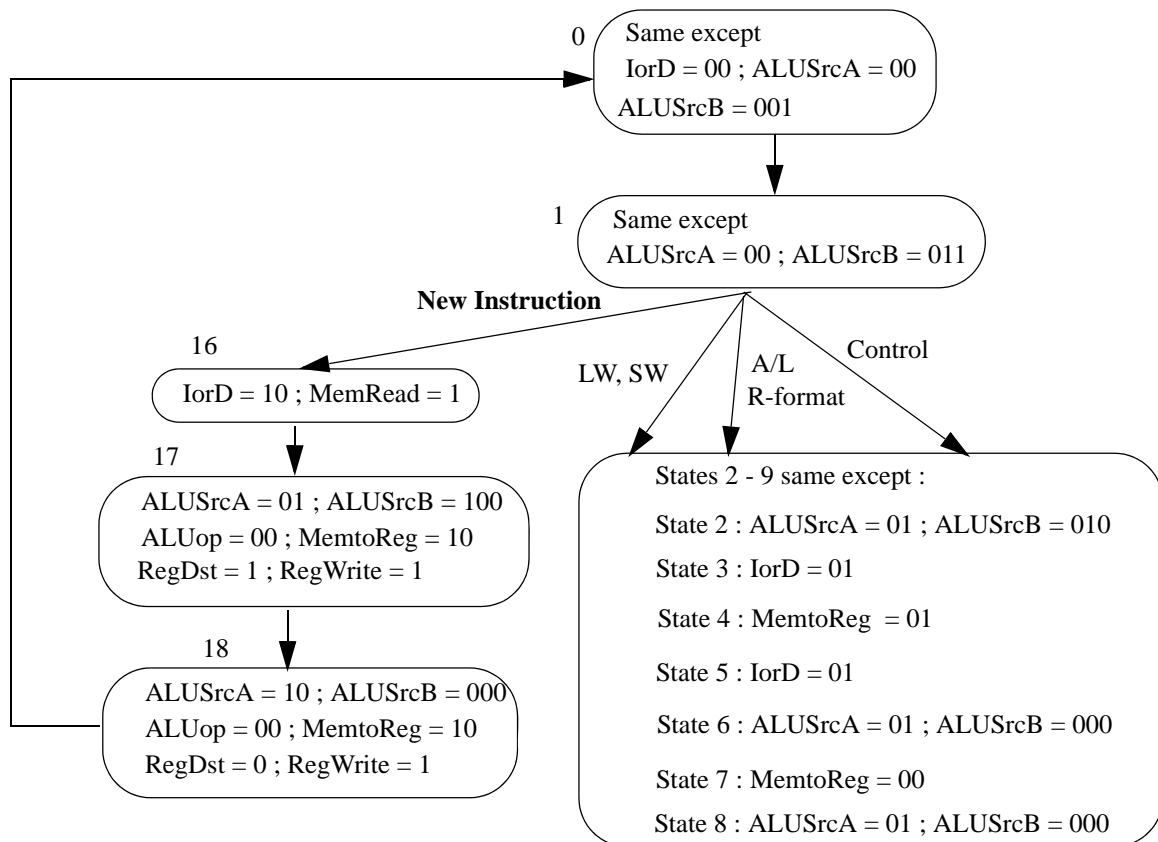
a) Modify the EMY **low-level** state diagram accordingly.

b) Assume that the EMY CPU is **microprogrammed**.

i) Draw the modified control unit which is still microprogrammed.

ii) Show the microcode for micromemory locations 16, 17 and 18. Are original micromemory locations different ? Why ?

A2) a) The modified low-level state diagram is as follows :

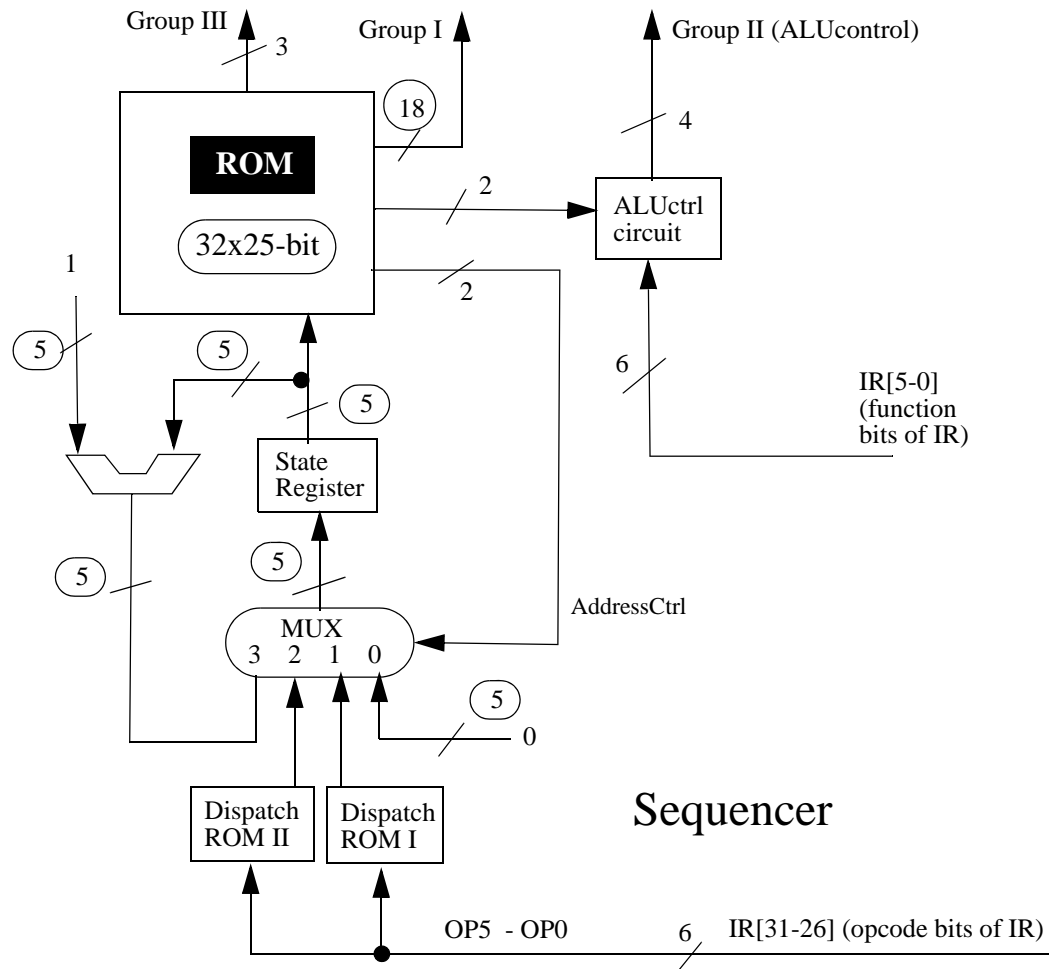


b) i) Since we have states 16, 17 and 18, there are 32 locations in the micromemory.

We have increased the number of control signals by four : IorD, ALUSrcA, ALUSrcB and MemtoReg have one new bit each.

Then, each micromemory location has $21 + 4 = 25$ bits. Therefore, the new size is 32×25 -bit.

The new microprogrammed Control Unit is shown below. The changes on the Control Unit are due to having having new microinstructions. All the changes on the sequencer are circled to quickly locate them. Dispatch ROMs are now 64×5 -bit. In addition, Dispatch ROM I is programmed in a location corresponding to the opcode value with the content of $(16)_{10}$.

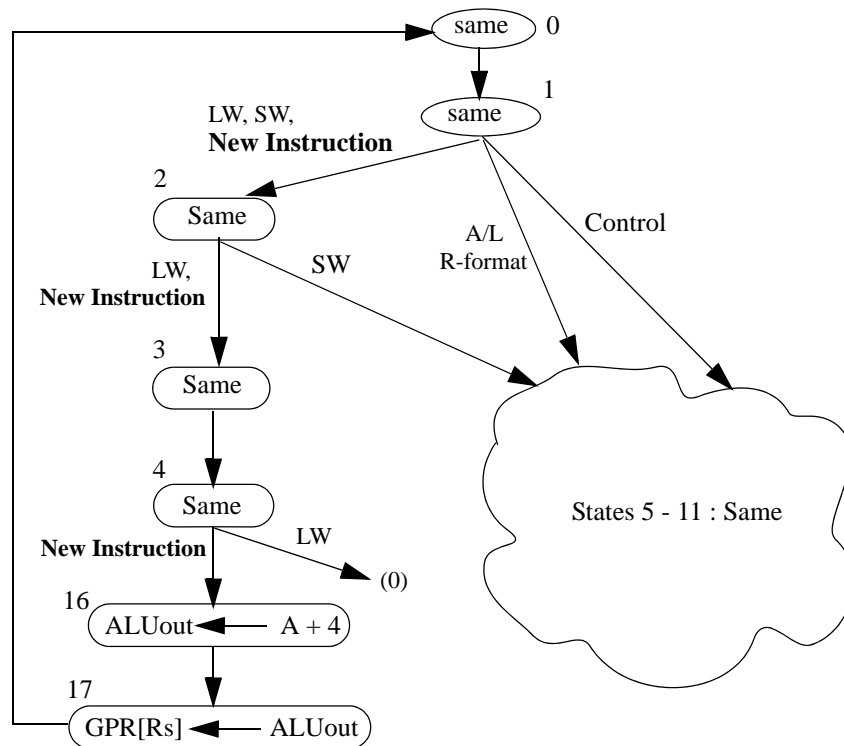


ii) The microcode for micromemory locations 16, 17 and 18 are as follows :

Loc	PCWrite	PCWriteCond	IorD	MemRead	MemWrite	IRWrite	MemtoReg	PCSource	ALUop	ALUSrcB	ALUSrcA	RegWrite	RegDst	AddrCtrl
16	0	0	10	1	0	0	00	00	00	000	00	0	0	11
17	0	0	00	0	0	0	10	00	00	100	01	1	1	11
18	0	0	00	0	0	0	10	00	00	000	10	1	0	00

The original micromemory locations (0 - 11) are different since each location has 25 bits, not 21 bits. That is, there are new bits in each location.

Q3) Assume that the EMY **high-level** state diagram has been modified to be able to run a new instruction as shown below :



a) Modify the EMY **datapath**. How long does it take to run the new instruction ?

b) Modify the EMY **low-level** state diagram accordingly.

c) Assume that the EMY CPU is **microprogrammed**.

i) What is the **new** size of the micromemory ? Explain.

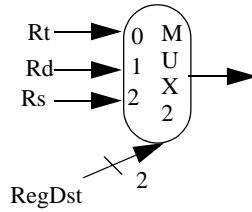
ii) Show the **microcode** for micromemory locations 16 and 17.

d) Do you think the new instruction can be used for the program below ? Why ?

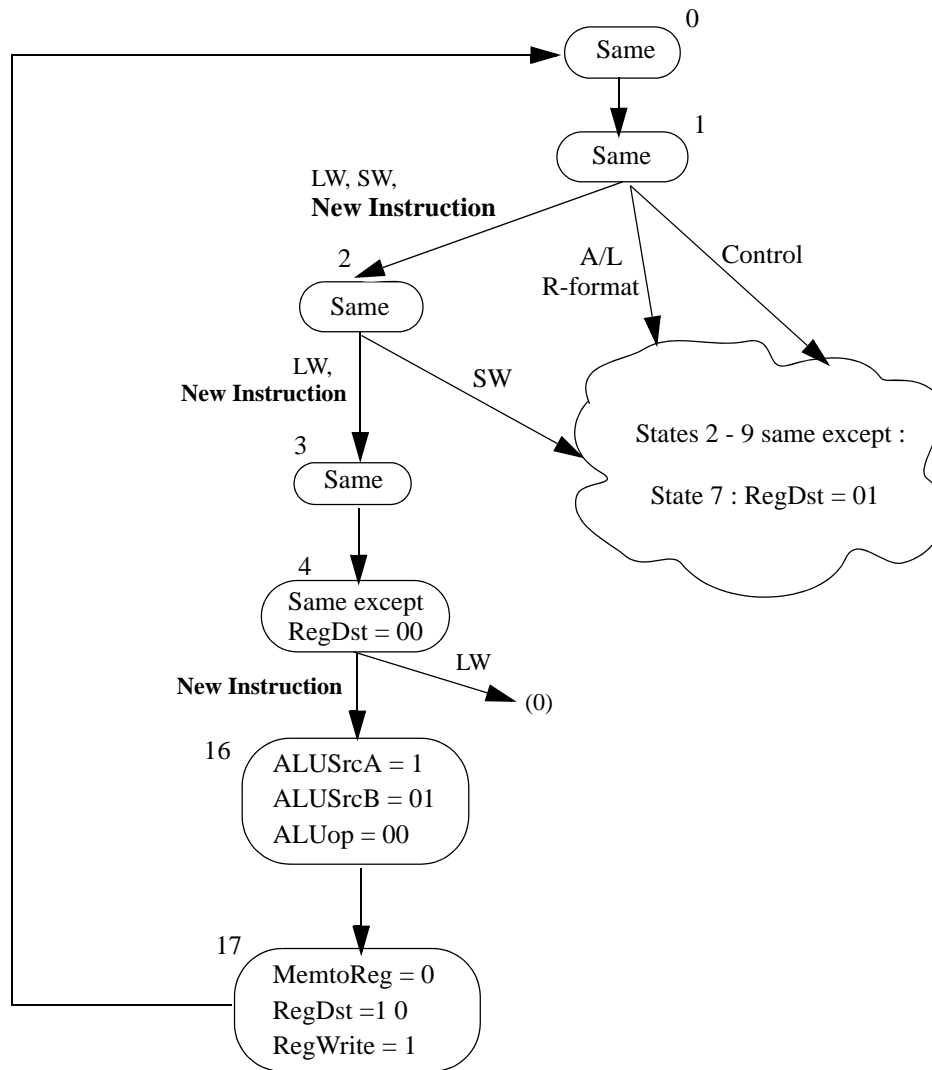
400000	LW	R8, 0(R9)	# R9 points at array A and initially has 10000000
400004	ADDI	R8, R8, (-1) ₁₀	
400008	OR	R8, R8, R10	# R10 is already initialized with a value
40000C	SLL	R8, R8, 2	
400010	SW	R8, 0(R9)	
400014	ADDI	R9, R9, 4	
400018	ADDI	R11, R11, (-1) ₁₀	# R11 is the loop-end counter
40001C	BNE	R11, R0, (-8) ₁₀	

A3) a) The new instruction takes **7** clock periods to run since we trace states 0, 1, 2, 3, 4, 16 and 17.

The modified datapath is as follows :



b) The modified low-level state diagram is as follows :



c)

i) Since we have states 16, and 17, there are 32 locations in the micromemory.

We have increased the number of control signals by two. RegDst has one new bit. The other new bit is for Address Control. It must have three bits to select one of five different possibilities for the MUX. The new possibility is a new Dispatch ROM to separate LW from the new instruction in state 4. Then, each micromemory location has $21 + 2 = 23$ bits. Therefore, the new size is 32×23 -bit

ii) The microcode for micromemory locations 16 and 17 are as follows :

Loc	PCWrite	PCWriteCond	IorD	MemRead	MemWrite	IRWrite	MemtoReg	PCSource	ALUop	ALUSrcB	ALUSrcA	RegWrite	RegDst	AddrCtrl
16	0	0	0	0	0	0	0	00	00	01	1	0	00	011
17	0	0	0	0	0	0	0	00	00	00	0	1	10	000

d) The new instruction can be used by the program in Question 1 since it loads a memory location to GPR Rt and then adds four to register Rs so that the next data access can be performed without using an explicit ADDI instruction on Rs. Below, the new instruction is described and the program in Question 1 is rewritten ;

i) The instruction loads a memory location to register and then advances (increments) the pointer register automatically.

ii) The **syntax** of the new instruction Load Word with autoIncrement : `LWI Rt, Disp(Rs)++`

The **semantics** of the new instruction : `Rt <--- M[Rs + Disp+] then Rs <--- Rs + 4`

The format is the **I format** :

Opcode	Rs	Rt	DOImm
6	5	5	16

Five arguments are used by the instruction :

The two destination arguments are registers Rs and Rt and they are explicitly specified by the instruction and so they use the Register instruction addressing mode.

For the first architectural operation, the source is a memory argument. We use the **2-byte signed displacement** addressing mode for it.

For the second architectural operation, the source arguments are register Rs which is explicitly specified hence the Register addressing mode and a constant which is implied hence the Implied addressing mode.

The program is rewritten by using the new instructions :

```

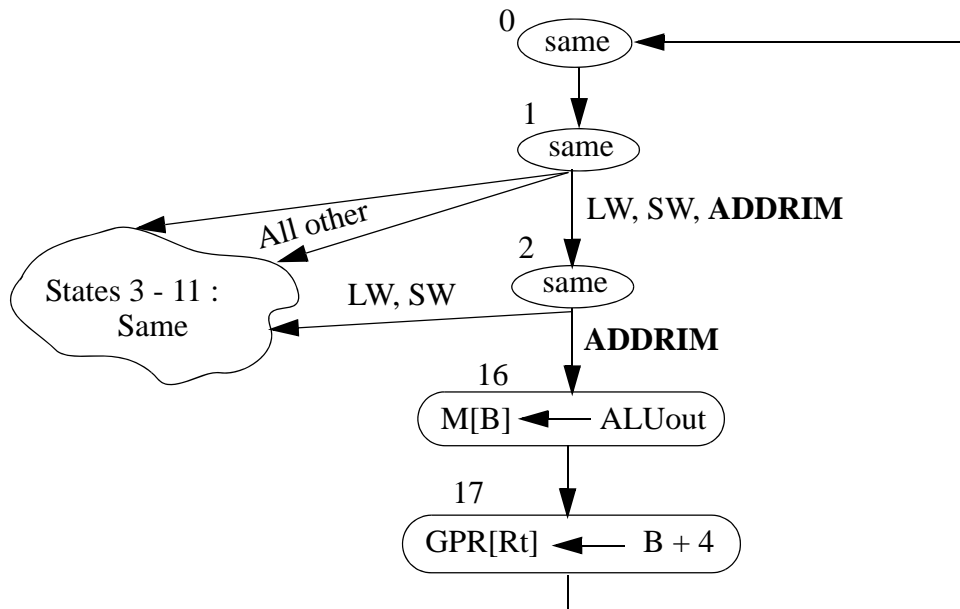
400000 LWI    R8, 0(R9)           # R9 points at array A and initially has 10000000
400004 ADDI   R8, R8, (-1)10
400008 OR     R8, R8, R10        # R10 is already initialized with a value
40000C SLL   R8, R8, 2
400010 SW    R8, (-4)10(R9)
400014 ADDI  R11, R11, (-1)10   # R11 is the loop-end counter
400018 BNE   R11, R0, (-7)10

```

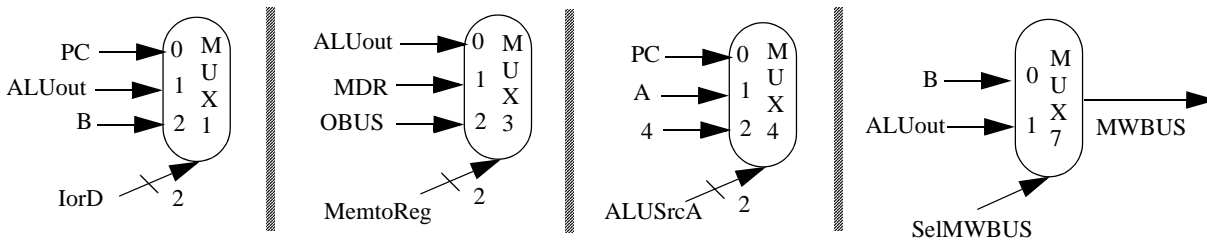
The program length is shorter. Each iteration takes 30 clock periods, not 32 as it is the case with the original program. Also, each iteration requires nine memory accesses, not 10.

However, this instruction takes 7 clock periods and pipelining it can be harder because of more stages and also a register (Rs) is read late that can cause WAR hazards.

Q4) Assume that the EMY **high-level** state diagram has been modified to be able to run a new instruction, **ADDRIM**, as shown below :



Assume that the EMY **datapath** is modified as follows :



a) Modify the EMY **low-level** state diagram accordingly.

b) Assume that the EMY CPU is **microprogrammed**.

- i) What is the **new** size of the micromemory ? **Explain**.
- ii) Show the **microcode** for micromemory locations **2, 16** and **17**.

c) This new instruction can be described as follows :

- Its syntax is : ADDRIM (Rt)++, Rs, Imm
- Its semantics is : M[Rt] Rs + Imm⁺ then Rt Rt + 4

The new instruction can be used for the following program :

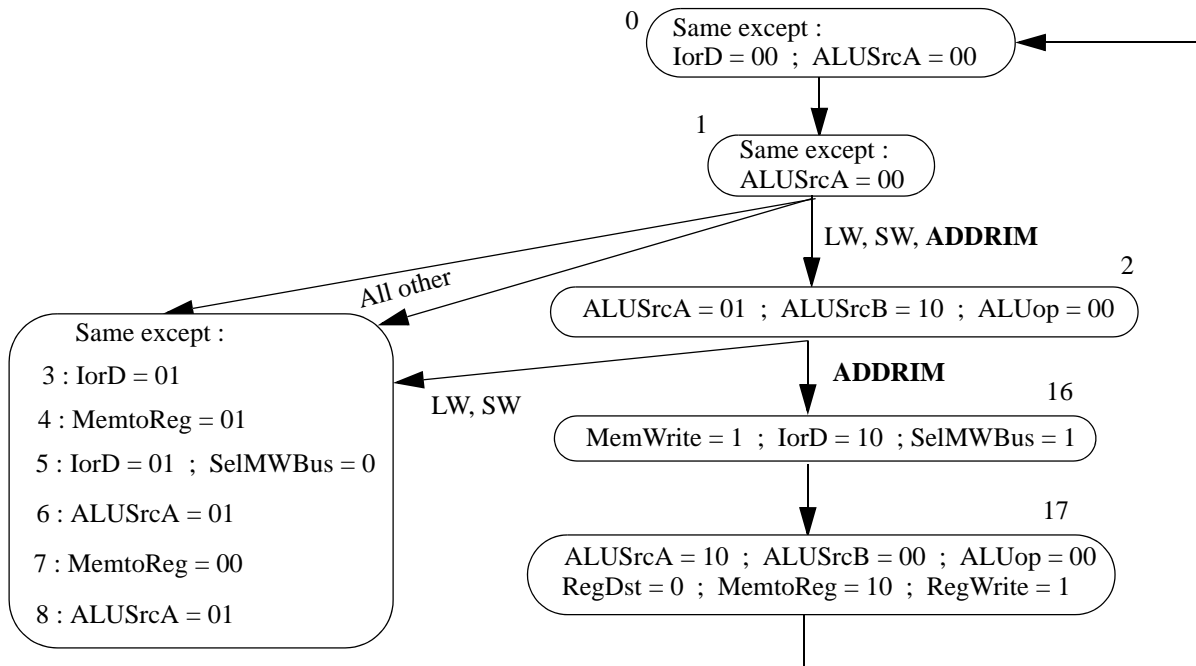
```

400C00  LW    R9, 0(R8)           # R8 points at array A and initially has 10EA0000
400C04  ADD   R9, R9, R9
400C08  ADDI  R10, R9, AC
400C0C  SW    R10, 0(R8)
400C10  ADDI  R8, R8, 4
400C14  ADDI  R11, R11, (-1)10  # R11 is the loop-end counter
400C18  BNE  R11, R0, (-7)10

```

Rewrite the program, by using the new instruction.

A4) a) The modified low-level state diagram is as follows :



b)

i) Since we have states 16, and 17, there are 32 locations in the micromemory.

We have increased the number of control signals by four. IorD, MemtoReg and ALUSrcA have one extra bit each. There is also a new control signal : SelMWBUS. Then, each micromemory location has 21 + 4 = 25 bits. Therefore, the new size is 32 x 25-bit

ii) The microcode for micromemory locations 2, 16 and 17 are as follows :

Loc	SelMWBUS	PCWrite	PCWriteCond	IorD	MemRead	MemWrite	IRWrite	MemtoReg	PCSource	ALUop	ALUSrcB	ALUSrcA	RegWrite	RegDst	AddrCtrl
2	0	0	0	00	0	0	0	00	00	00	10	01	0	0	00
16	1	0	0	10	0	1	0	00	00	00	00	00	0	0	11
17	0	0	0	00	0	0	0	10	00	00	00	10	1	0	00

d) The program is rewritten by using the new instructions :

```

400C00  LW      R9, 0(R8)           # R8 points at array A and initially has 10EA0000
400C04  ADD     R9, R9, R9
400C08  ADDRIM (R8)++, R9, AC
400C0C  ADDI   R11, R11, (-1)10      # R11 is the loop-end counter
400C10  BNE   R11, R0, (-5)10
    
```

The program length is shorter. Each iteration takes 21 clock periods, not 28 as it is the case with the original program. Also, each iteration requires seven memory accesses, not nine.

However, this instruction is a CISC instruction, accessing the memory and performing an A/L instruction. Pipelining can be harder.

Q5) Consider the following EMY subroutine :

```

400500 LW      R8, 0(R4)           # R4 is passed a value of 10002004
400504 ADD     R10, R9, R8        # R9 is passed a value
400508 SUB     R9, R11, R10       # R11 is passed a value
40050C ADDI    R4, R4, 4
400510 ADDI    R5, R5, (-1)10    # R5 is passed a value of 2
400514 BNE     R5, R0, (-6)10
400518 SW     R9, 0(R6)           # R6 is passed a value of 10002000
40051C JR      R31
  
```

The EMY CPU is pipelined as explained in class : It has forwardings, delayed branches and write-in-the-first-half, read-in-the-second-half register usage. Assume also that there is a perfect memory, with **no** stalls. Finally, assume that the ADDI takes five clock periods.

The code above is written for the **unpipelined** EMY CPU since there are **no** independent instructions after the LW and control instructions. That is why, for example, the ADD instruction is right after the LW instruction.

Rewrite the program to prevent load and branch stalls, by using as few NOPs as possible.

After rewriting the subroutine, show the execution of the modified subroutine as discussed in class until the subroutine returns. Show all **necessary** forwardings, register-reading-writing and resolved data hazard types.

A5) The reordered program and its execution timing are below. All data hazards are **RAW**. The branch delay slot contains a SUB instruction. The delay slot after the JR has a SW instruction.

Instruction	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
400500 LW R8, 0(R4)	1	2	3	4	5	7	8	9	10	11
400504 ADDI R4, R4, 4	2	3	4	5	6	8	9	10	11	12
400508 ADD R10, R9, R8	3	4	5	6	7 _H	9	10 _H	11	12	13
40050C ADDI R5, R5, (-1) ₁₀	4	5	6	7	8	10	11	12	13	14
400510 BNE R5, R0, (-6) ₁₀	5	6				11	12			
400514 SUB R9, R11, R10	6	7 _H	8	9	10 _H	12	13	14	15	16
400518 JR R31						13	14			
40051C SW R9, 0(R6)						14	15	16	17	

Q6) Consider the following piece of EMY mnemonic machine language program written for the **unpipelined** CPU:

```

400000 LW      R8, 0(R9)           # R9 points at variable k which is at 10000000
400004 LW      R10, 0(R11)        # R11 points at vector A which starts at 10004000
400008 ADD     R12, R8, R10
40000C SW     R12, 0(R13)         # R13 points at vector B which starts at 10008000
400010 SUB     R10, R0, R12
400014 SW     R10, 0(R11)
  
```

```

400018   ADDI   R11, R11, 4
40001C   ADDI   R13, R13, 4
400020   ADDI   R14, R14, (-1)10   # R14 is the loop-end counter
400024   BNE   R14, R0, (-9)10

```

The EMY CPU is **pipelined** as discussed in class : It has forwardings, delayed branches and write in-the-first-half read-in-the-second-half register usage. Assume also that there is a **perfect memory**, with no stalls.

a) **Reorder** the instructions to avoid delays. Do **not** use any NOPs. Also, do **not** increase the number of instructions.

b) Show the execution of the reordered program for **two** iterations for which you show all the **necessary** forwardings, register-reading-writing and resolved data hazard types as done **in class**.

A6) a) & b) The execution timing of the reordered program is below. All hazards are **RAW** type hazards.

Instruction	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
400000 LW R8, 0(R9))	1	2	3	4	5 H					
400004 LW R10, 0(R11)	2	3	4	5	6	11	12	13	14	15
400008 ADDI R11, R11, 4	3	4	5	6	7	12	13	14	15	16
40000C ADD R12, R8, R10	4	5 H	6	7	8	13	14	15	16	17
400010 SW R12, 4(R13)	5	6	7	8		14	15	16	17	
400014 SUB R10, R0, R12	6	7	8	9	10	15	16	17	18	19
400018 SW R10, (-4) ₁₀ (R11)	7	8	9	10		16	17	18	19	
40001C ADDI R14, R14, (-1) ₁₀	8	9	10	11	12	17	18	19	20	21
400020 BNE R14, R0, (-8) ₁₀	9	10				18	19			18
400024 ADDI R13, R13, 4	10	11	12	13	14	19	20	21	22	23

Q7) Consider the following EMY mnemonic machine language program :

```

400500   LW      R11, 0(R8)           # R8 initially has 10002200 and points at vector B
400504   LW      R12, 0(R9)           # R9 initially has 10003300 and points at vector C
400508   ADD     R13, R11, R12
40050C   ADD     R14, R14, R13        # R14 is initially 0
400510   SW      R14, 0(R10)         # R10 initially has 10004400 and points at vector A
400514   ADDI   R8, R8, 4
400518   ADDI   R9, R9, 4
40051C   ADDI   R10, R10, 4
400520   ADDI   R15, R15, (-1)10    # R15 determines the number of iterations
400524   BNE   R15, R0, (-10)10

```

The EMY CPU is **pipelined** as explained in class : It has forwardings, delayed branches and write-in-the-first-half, read-in-the-second-half register usage. Assume also that there is a perfect memory, with **no** stalls. Finally, assume that the ADDI takes five clock periods.

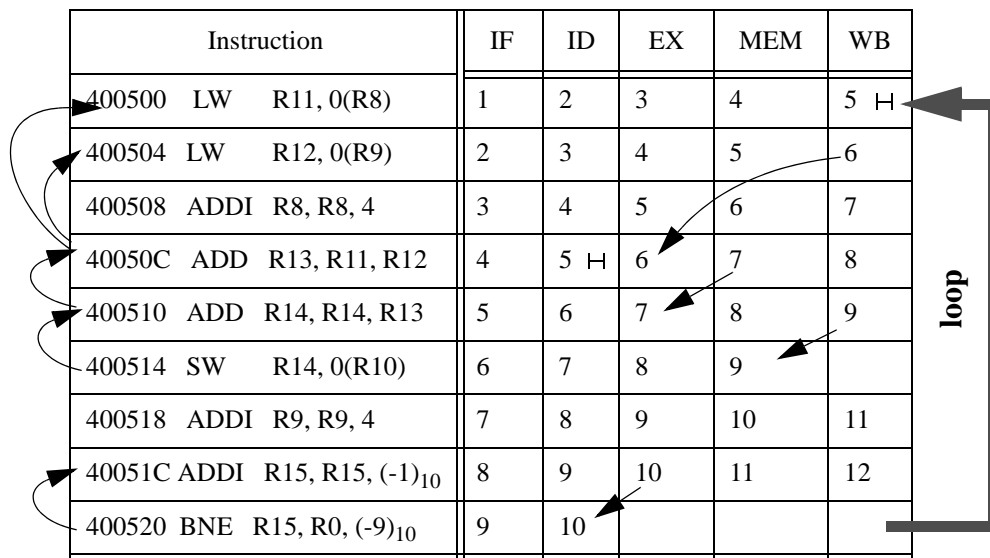
Reorder the instructions to place independent instructions after LW instructions and in the delayed branch slot. Note that the number of instructions after the reordering will be still **ten**. Then, show the execution of the loop obtained in part (a) for **one** iteration for which you show all **necessary** forwardings, register-reading-writing and resolved data hazard types as done **in class**.

A7) a) Instructions are reordered as follows :

```

400500 LW      R11, 0(R8)      # R8 initially has 10002200 and points at vector B
400504 LW      R12, 0(R9)      # R9 initially has 10003300 and points at vector C
400508 ADDI    R8, R8, 4
40050C ADD     R13, R11, R12
400510 ADD     R14, R14, R13    # R14 is 0 initially
400514 SW      R14, 0(R10)     # R10 initially has 10004400 and points at vector A
400518 ADDI    R9, R9, 4
40051C ADDI    R15, R15, (-1)10 # R15 determines the number of iterations
400520 BNE     R15, R0, (-9)10
400524 ADDI    R10, R10, 4
    
```

b) The reordered program and its execution timing are below. All data hazards are **RAW** type hazards. Note that the branch delay slot contains an ADDI instruction.



Q8) Consider the following EMY function :

```

401050 ADD     R8, R0, R0      # We clear R8
401054 ADD     R9, R4, R0      # We move R4, that is Y, to R9
401058 ADD     R8, R5, R8      # We add R5, Z, to R8 (add Z to temporary result)
40105C ADDI    R9, R9, (-1)10  # We subtract 1 from Y
401060 BNE     R9, R0, (-3)10  # Is it the end (is Y equal to zero) ? If not, go to 401058
401064 ADD     R2, R8, R0      # The end. We move the result to R2
401068 JR      R31            # We return from the subroutine
    
```

The EMY CPU is pipelined as explained in class : It has forwardings, delayed branches and write-in-the-first-half, read-in-the-second-half register usage. Assume also that there is a perfect memory, with **no** stalls. Finally, assume that the ADDI takes five clock periods.

The above subroutine is output by a compiler **without** considering delayed branches. That is why, for example, the ADD instruction is right after the BNE instruction and there is no instruction following the JR instruction.

Rewrite the program to prevent load and control instruction stalls, by using as few NOPs as possible.

After rewriting the subroutine, show the execution of the modified subroutine as discussed in class until the subroutine returns. Show all **necessary** forwardings, register-reading-writing and resolved data hazard types. Assume that the value of Y is initially 2.

A8) The reordered program and its execution timing are below. All data hazards are **RAW** type hazards. Note that the branch delay slot contains an ADD instruction. The delay slot after the JR contains a NOP instruction.

Instruction	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
401050 ADD R8, R0, R0	1	2	3	4	5					
401054 ADD R9, R4, R0	2	3	4	5	6					
401058 ADDI R9, R9, (-1) ₁₀	3	4	5	6	7 H	6	7 H	8	9	10
40105C BNE R9, R0, (-2) ₁₀	4	5				7	8			
401060 ADD R8, R5, R8	5	6	7	8	9 H	8	9 H	10	11	12
401064 JR R31						9	10			
401068 ADD R2, R8, R0						10	11	12	13	14

The diagram shows a pipeline execution table with 11 columns representing clock cycles. The instructions are: 401050 ADD R8, R0, R0; 401054 ADD R9, R4, R0; 401058 ADDI R9, R9, (-1)₁₀; 40105C BNE R9, R0, (-2)₁₀; 401060 ADD R8, R5, R8; 401064 JR R31; 401068 ADD R2, R8, R0. Arrows indicate data hazards: from 401054 to 401058 (R9), from 401058 to 401060 (R8), and from 401060 to 401068 (R8). A loop structure is shown with a vertical arrow labeled 'loop' on the right side, indicating that the sequence from instruction 401058 to 401064 repeats.

Q9) The EMY CPU is pipelined as explained in class : It has forwardings, delayed branches and write-in-the-first-half, read-in-the-second-half register usage. Assume also that there is a perfect memory, with **no** stalls. Finally, assume that the ADDI takes five clock periods.

Consider the following piece of EMY mnemonic machine language program :

```

400200 ADD R8, R9, R10      # R9 = 10000000 and R10 = 1400 initially
400204 LW R11, 0(R8)
400208 LW R13, 0(R12)      # R12 initially has 10002400
40020C LW R14, 0(R11)
400210 LW R15, 0(R16)     # R16 initially has 10003400
400214 ADD R17, R13, R14
400218 SUB R18, R17, R15
40021C ADDI R10, R10, 4
400220 ADDI R19, R19, (-1)10 #R19 initially has 2
400224 ADDI R12, R12, 4
400228 ADD R20, R18, R20  # R20 has already been initialized
40022C BNE R19, R0, (-12)10
400230 ADDI R16, R16, 4
400234 SW R20, 0(R16)

```

The above code is written for the **pipelined** EMY CPU since independent instructions are placed after the LW and branch instructions.

Show the execution of the code as discussed in class until the code completes. Show all **necessary** forwardings, register-reading-writing and resolved data hazard types.

A9) The execution timing is shown below. All resolved data hazards are RAW type hazards.

Instruction	IF	ID	EX	MEM	WB
400200 ADD R8, R9, R10	1	2	3	4	5
400204 LW R11, 0(R8)	2	3	4	5	6
400208 LW R13, 0(R12)	3	4	5	6	7 H
40020C LW R14, 0(R11)	4	5	6	7	8
400210 LW R15, 0(R16)	5	6	7	8	9
400214 ADD R17, R13, R14	6	7 H	8	9	10
400218 SUB R18, R17, R15	7	8	9	10	11
40021C ADDI R10, R10, 4	8	9	10	11	12
400220 ADDI R19, R19, (-1) ₁₀	9	10	11	12	13 H
400224 ADDI R12, R12, 4	10	11	12	3	14
400228 ADD R20, R18, R20	11	12	13	14	15
40022C BNE R19, R0, (-12) ₁₀	12	13 H			
400230 ADDI R16, R16, 4	13	14	15	16	17
<hr/>					
400200 ADD R8, R9, R10	14	15	16	17	18
400204 LW R11, 0(R8)	15	16	17	18	19
400208 LW R13, 0(R12)	16	17	18	19	20 H
40020C LW R14, 0(R11)	17	18	19	20	21
400210 LW R15, 0(R16)	18	19	20	21	22
400214 ADD R17, R13, R14	19	20 H	21	22	23
400218 SUB R18, R17, R15	20	21	22	23	24
40021C ADDI R10, R10, 4	21	22	23	24	25
400220 ADDI R19, R19, (-1) ₁₀	22	23	24	25	26 H
400224 ADDI R12, R12, 4	23	24	25	26	27
400228 ADD R20, R18, R20	24	25	26	27	28 H
40022C BNE R19, R0, (-12) ₁₀	25	26 H			
400230 ADDI R16, R16, 4	26	27	28	29	30
400234 SW R20, 0(R16)	27	28 H	29	30	

Q10) Consider the program below :

```

400000    ADDI    R8, R0, 0
400004    LW     R9, 0(R10)           # R10 points at array A and initially has 10000000
400008    ADD    R8, R8, R9
40000C    ADDI   R10, R10, 4
400010    ADDI   R11, R11, (-1)10   # R11 is the loop-end counter
400014    BNE   R11, R0, (-5)10
400018    SW    R8, 0(R10)
    
```

The EMY CPU is **pipelined** as explained in class : It has forwardings, delayed branches and write in-the-first-half read-in-the-second-half register usage. Assume also that there is a **perfect memory**, with no stalls. Finally, assume that the ADDI takes five clock periods.

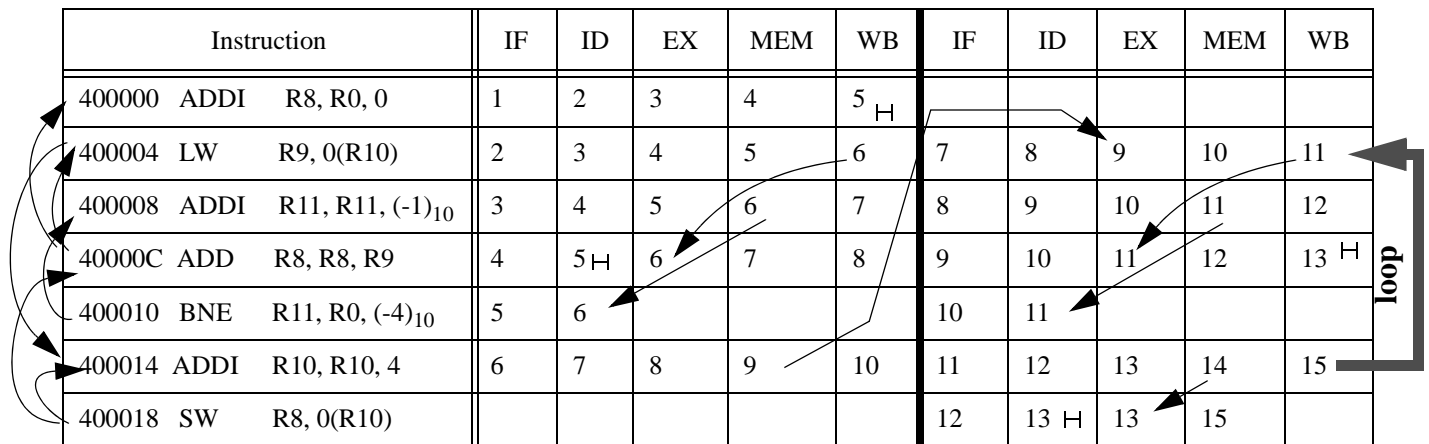
The above program is written for the **unpipelined** EMY CPU since there are **no** independent instructions after LW and Branch instructions. Assume also that R11 initially has $(1000)_{10}$.

i) Reorder the instructions to avoid delays. Do **not** use any NOPs. Also, do **not** increase the number of instructions.

ii) Show the execution of the reordered program for **two** iterations for which you show all the **necessary** forwardings, register-reading-writing and resolved data hazard types as done **in class**.

iii) How many clock periods will it take to run the reordered program if there are $(1000)_{10}$ iterations ? Explain.

A10) i) & ii) The execution timing of the reordered program is below. All hazards are **RAW** type hazards.



iii) We see that the last instruction of the loop is fetched in clock periods 6, 11, 16, 21,... or $6 + (k - 1)*5$. For 1000 iterations then it is $6 + 999*5 = 5001$. Then, the SW is fetched in clock period 5002 and completed as 5003, 5004 and 5005. Thus, it takes 5005 clock periods to run the program with pipelining.

The speedup is $T_{old}/T_{new} = 20008/5005 = 4$. The CPI_i figures are different in pipelined and unpipelined cases though.

Q11) Consider the following piece of EMY mnemonic machine language program :

```

400000    LW     R8, 0(R9)           # R9 points at array A and initially has 10000000
400004    ADDI   R8, R8, (-1)10
400008    OR    R8, R8, R10         # R10 is already initialized with a value
    
```

```

40000C    SLL    R8, R8, 2
400010    SW     R8, 0(R9)
400014    ADDI   R9, R9, 4
400018    ADDI   R11, R11, (-1)10    # R11 is the loop-end counter
40001C    BNE    R11, R0, (-8)10

```

The EMY CPU is **pipelined** as explained in class : It has forwardings, delayed branches and write in-the-first-half read-in-the-second-half register usage. Assume also that there is a **perfect memory**, with no stalls.

The program above is written for the **unpipelined** EMY CPU since there are **no** independent instructions after LW and Branch instructions.

i) Reorder the instructions to avoid delays. Do **not** use any NOPs. Also, do **not** increase the number of instructions.

ii) Show the execution of the reordered program for **two** iterations for which you show all the **necessary** forwardings, register-reading-writing and resolved data hazard types as done **in class**.

A11) i) & ii) The execution timing of the reordered program is below. All hazards are **RAW** type hazards.

Instruction	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
400000 LW R8, 0(R9)	1	2	3	4	5	9	10	11	12	13
400004 ADDI R11, R11, (-1) ₁₀	2	3	4	5	6	10	11	12	13	14
400008 ADDI R8, R8, (-1) ₁₀	3	4	5	6	7	11	12	13	14	15
40000C OR R8, R8, R10	4	5	6	7	8	12	13	14	15	16
40000C SLL R8, R8, 2	5	6	7	8	9	13	14	15	16	17
400018 SW R8, 0(R9)	6	7	8	9		14	15	16	17	18
400010 BNE R11, R0, (-7) ₁₀	7	8				15	16			
400014 ADDI R9, R9, 4	8	9	10	11	12	16	17	18	19	20

Q12) Consider the following piece of EMY mnemonic machine language program :

```

400C00    LW     R9, 0(R8)           # R8 points at array A and initially has 10EA0000
400C04    ADD    R9, R9, R9
400C08    ADDI   R10, R9, AC
400C0C    SW     R10, 0(R8)
400C10    ADDI   R8, R8, 4
400C14    ADDI   R11, R11, (-1)10    # R11 is the loop-end counter
400C18    BNE    R11, R0, (-7)10

```

The EMY CPU is **pipelined** as explained in Section 6.6 of the textbook and in class : It has forwardings, delayed branches and write in-the-first-half read-in-the-second-half register usage. Assume also that there is a **perfect memory**, with no stalls. Consider the program in Question 1 above. It is written for the **unpipelined** EMY CPU since there are **no** independent instructions after LW and Branch instructions.

i) Reorder the instructions to avoid delays. Do **not** use any NOPs. Also, do **not** increase the number of instructions.

ii) Show the execution of the reordered program for **two** iterations for which you show all the **necessary** forwardings,

register-reading-writing and resolved data hazard types as done **in class**.

A12) i) & ii) The execution timing of the reordered program is below. All hazards are **RAW** type hazards.

Instruction	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
400C00 LW R9, 0(R8)	1	2	3	4	5	8	9	10	11	12
400C04 ADDI R8, R8, 4	2	3	4	5	6	9	10	11	12	13
400C08 ADD R9, R9, R9	3	4	5	6	7	10	11	12	13	14
400C0C ADDI R10, R9, AC	4	5	6	7	8 _H	11	12	13	14	15 _H
400C10 ADDI R11, R11, (-1) ₁₀	5	6	7	8	9	12	13	14	15	16
400C14 BNE R11, R0, (-7) ₁₀	6	7				13	14			
400C18 SW R10, (-4) ₁₀ (R8)	7	8 _H	9	10		14	15 _H	16	17	

Q13) Consider the following piece of EMY mnemonic machine language program :

```

400000 LW R8, 0(R9) # R9 points at vector A which starts at 10000000
400004 LW R10, 4000(R9) # R10 is loaded from vector B
400008 SW R10, 0(R9)
40000C SW R8, 4000(R9)
400010 ADDI R9, R9, 4
400014 ADDI R11, R11, (-1)10 # R11 is the loop-end counter
400018 BNE R11, R0, (-7)10

```

a) Assume that the EMY CPU is **unpipelined**, as discussed in class and there is a **perfect memory**, with no stalls. Assume also that R11 initially has $(1000)_{10}$ and the clock frequency is 4 GHz. Assume that the instruction execution timings for the above program are as follows : $CPI_{LW} = 5$, $CPI_{SW} = 4$, $CPI_{ADDI} = 4$ and $CPI_{BNE} = 3$. Calculate the following figures :

- i) CPI_{ave}
- ii) CPUtime
- iii) $GIPS_{ave}$

b) Assume now that the EMY CPU is **pipelined** as discussed in class : It has forwardings, delayed branches and write in-the-first-half read-in-the-second-half register usage. Assume also that there is a **perfect memory**, with no stalls. Assume also that the clock frequency is the same : 4 GHz.

Consider again the program above. It is written for the **unpipelined** EMY CPU.

- i)** **Reorder** the instructions to avoid delays. Do **not** use any NOPs. Also, do **not** increase the number of instructions.
- ii)** Show the execution of the reordered program for **two** iterations for which you show all the **necessary** forwardings, register-reading-writing and resolved data hazard types as done **in class**.
- iii)** Calculate the following figures :

- aa) CPI_{ave}
- bb) CPUtime
- cc) $GIPS_{ave}$
- dd) $Speedup_{overall}$

A13)

a) The CPU is unpipelined :

i) We execute the following instructions for “**k**” iterations : $k * (LW + LW + SW + SW + ADDI + ADDI + BNE)$

“k” is given as $(1,000)_{10}$. Then, the number of clock periods for the program based on the given CPI_i figures is : $5 + 1000(5 + 5 + 4 + 4 + 4 + 4 + 3) = 29000$.

The number of instructions executed is : $1 + 1000(1 + 1 + 1 + 1 + 1 + 1 + 1) = 7000$

$$CPI_{ave} = \frac{\text{Number of clock periods for the program}}{NI} = \frac{29000}{7000} = 4.14$$

ii) The CPUtime : First, we have to obtain the clock period :

$$\text{clock period} = \frac{1}{4 \times 10^9} = 0.25 \times 10^{-9} = 0.25\text{ns}$$

$$\text{CPUtime} = NI \times CPI_{ave} \times \text{clock period duration}$$

$$= 7000 \times 4.14 \times 0.25 \times 10^{-9} \text{seconds} = 7.25 \times 10^{-6} \text{seconds} = 7.25\mu\text{seconds}$$

iii) The $GIPS_{ave}$:

$$GIPS_{ave} = \frac{NI}{\text{CPUtime} \times 10^9} = \frac{7000}{7.25 \times 10^{-6} \times 10^9} = 0.9655$$

b) The CPU is pipelined :

i) & ii) The execution timing of the reordered program is below. All data hazards are RAW type hazards.

Instruction	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
400000 LW R8, 0(R9)	1	2	3	4	5	8	9	10	11	12
400004 LW R10, 4000(R9)	2	3	4	5	6	9	10	11	12	13
400008 SW R10, 0(R9)	3	4	5	6		10	11	12	13	
40000C SW R8, 4000(R9)	4	5	6	7		11	12	13	14	
400010 ADDI R11, R11, (-1) ₁₀	5	6	7	8	9	12	13	14	15	16
400014 BNE R14, R0, (-6) ₁₀	6	7				13	14			
400018 ADDI R9, R9, 4	7	8	9	10	11	14	15	16	17	18

iii)

aa) We calculate CPI_{ave} for the pipelined case : Two iterations take 18 clock periods to run. To run 1000 iterations we spend

$$(7 * 1000) + 4 = 7004 \text{ clock periods}$$

The number of instructions run is the same as the unpipelined case : 7000. Then :

$$CPI_{ave} = \frac{\text{Number of clock periods for the program}}{NI} = \frac{7004}{7000} = 1.0006$$

bb) We calculate CPUtime for the pipelined case :

$$\text{CPUtime} = NI \times CPI_{ave} \times \text{clock period duration}$$

$$= 7000 \times 1.00 \times 0.25 \times 10^{-9} \text{seconds} = 1.75 \times 10^{-6} \text{seconds} = 1.75\mu\text{seconds}$$

cc) We calculate $GIPS_{ave}$ for the pipelined case :

$$GIPS_{ave} = \frac{NI}{CPUtime \times 10^9} = \frac{7000}{1.75 \times 10^{-6} \times 10^9} = 4$$

dd) We calculate $Speedup_{overall}$:

$$Speedup_{overall} = \frac{CPUtime_{old}}{CPUtime_{new}} = \frac{7.25}{1.75} = 4.14$$

Q14) Consider the following piece of EMY mnemonic machine language program :

```

400800    LW      R8, 0(R9)           # R9 points at array A and initially has 10000000
400804    LW      R10, 4(R9)
400808    ADD     R11, R8, R10
40080C    SW      R11, 4(R9)
400810    SUB     R8, R8, R12        # R12 is already initialized with a value
400814    ADDI   R9, R9, 4
400818    ADDI   R13, R13, (-1)10   # R13 is the loop-end counter
40081C    BNE    R13, R0, (-7)10
    
```

The EMY CPU is **pipelined** as discussed in class : It has forwardings, delayed branches and write in-the-first-half read-in-the-second-half register usage. Assume also that there is a **perfect memory**, with no stalls. The program above is written for the **unpipelined** EMY CPU since there are **no** independent instructions after LW and Branch instructions.

a) **Reorder** the instructions to avoid delays. Do **not** use any NOPs. Also, do **not** increase the number of instructions.

b) Show the execution of the reordered program for **two** iterations for which you show all the **necessary** forwardings, register-reading-writing and resolved data hazard types as done **in class**.

c) How many clock periods will it take to run the reordered program if there are $(1,000)_{10}$ iterations ? Explain.

A14) a) & b) The execution timing of the reordered program is shown below. All hazards are **RAW** type hazards.

Instruction	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
400E00 LW R8, 0(R9))	1	2	3	4	5 H					
400E04 LW R10, 4(R9)	2	3	4	5	6	9	10	11	12	13
400E08 ADDI R13, R13, (-1) ₁₀	3	4	5	6	7	10	11	12	13	14
400E0C ADD R11, R8, R10	4	5 H	6	7	8	11	12	13	14	15
400E10 SW R11, 4(R9)	5	6	7	8		12	13	14	15	
400E14 SUB R8, R8, R12	6	7	8	9	10	13	14	15	16	17
400E18 BNE R13, R0, (-6) ₁₀	7	8				14	15			18
400E1C ADDI R9, R9, 4	8	9	10	11	12	15	16	17	18	19

c) We see that the last instruction of the loop is fetched in clock periods 8, 15, 22, 29,... or $8 + (k - 1) * 7$. For $k = 1000$ iterations, it is $8 + 999*7 = 7001$. This last instruction completes by going through the stages as $7001 - 7005$.

Q15) Consider the following piece of EMY mnemonic machine language program :

```

400000    LW      R8, 0(R9)           # R9 points at vector A which starts at 10000000
400004    LW      R10, 4000(R9)      # R10 is loaded from vector B
400008    ADDI   R12, R12, (-1)10    # R12 is the loop-end counter
40000C    ADD    R11, R8, R10
400010    SW     R11, 4000(R9)
400014    BNE   R12, R0, (-6)10
400018    ADDI   R9, R9, 4
    
```

The program is written for the **pipelined** EMY CPU :

- There is an independent instruction following the second LW instruction and
- there is an independent instruction in the branch delay slot.

a) Assume the EMY CPU is **pipelined** as discussed in class : It has forwardings, delayed branches and write in-the-first-half read-in-the-second-half register usage.

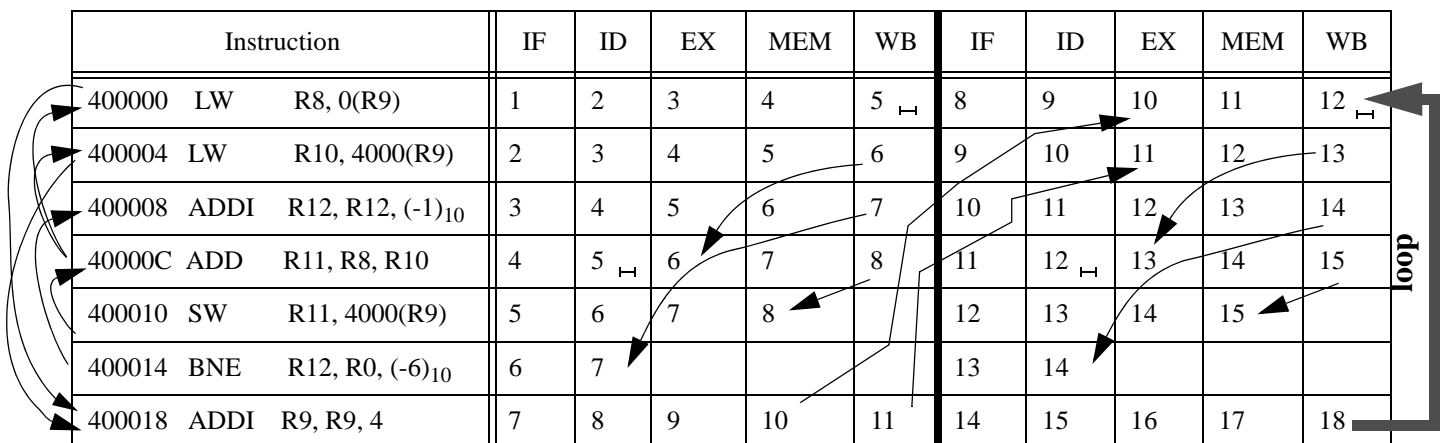
Assume also that there is a **perfect memory**, with no stalls. Show the execution of the above program for **two** iterations for which you show all the **necessary** forwardings, register-reading-writing and resolved data hazard types as done **in class**.

b) Based on your answer to part (a), calculate the following figures :

- a) CPI_{ave}
- b) CPUtime
- c) $GIPS_{ave}$

Assume that R12 initially has $(1000)_{10}$ and the clock frequency is 4 GHz.

A15) a) The execution timing of the program is below. All data hazards are **RAW** type hazards.



b) The pipelined execution of the loop for “n” iterations takes $n*7 + 4$ clock periods based on the execution for two iterations. Then, for 1000 iterations, the loop takes $1000*7 + 4 = 7004$ clock periods.

There are 7 instructions in the loop. Then, for 1000 iterations, we execute $1000 * 7 = 7000$ instructions

$$CPI_{ave} = \frac{\text{Number of clock periods for the program}}{NI} = \frac{7004}{7000} = 1$$

b) The CPUtime : First, we have to obtain the clock period :

$$\text{clock period} = \frac{1}{4 \times 10^9} = 0.25 \times 10^{-9} = 0.25\text{ns}$$

$$\text{CPUtime} = NI \times CPI_{ave} \times \text{clock period duration}$$

$$= 7000 \times 1 \times 0.25 \times 10^{-9} \text{seconds} = 1750 \times 10^{-9} \text{seconds} = 1.75\mu\text{seconds}$$

b) The GIPS_{ave} :

$$GIPS_{ave} = \frac{NI}{\text{CPUtime} \times 10^9} = \frac{7000}{1750 \times 10^{-9} \times 10^9} = 4$$

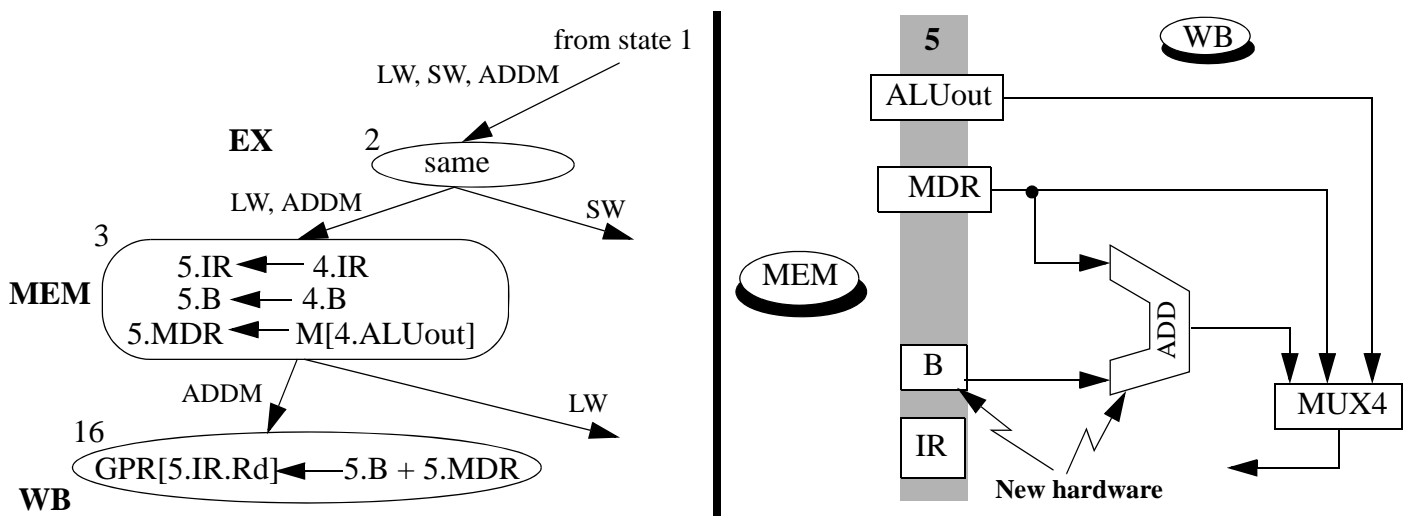
Q16) Assume that the EMY CPU is pipelined as explained in class : It has forwardings, delayed branches and write-in-the-first-half, read-in-the-second-half register usage. Assume also that there is a perfect memory, with **no** stalls.

We decide to add the following new instruction to the EMY instruction set :

ADDM Rt, Displ(Rs1) # Rt <--- Rt + M[Rs + Displ⁺]

The instruction uses the Immediate format. Modify the state diagram and the pipelined EMY CPU datapath given in class as little as possible to run the new instruction in five (5) clock periods.

A16) The modified state diagram and the datapath are shown below.



The new ADDM instruction is an A/L instruction that accesses the memory for data in state 3. Therefore, the MIPS is **not** a L/S machine anymore. Besides, a 32-bit ADDer is used in the WB stage for its execution.