

CS2214 COMPUTER ARCHITECTURE & ORGANIZATION FALL 2009

HOMEWORK III

DUE : October 27, 2009

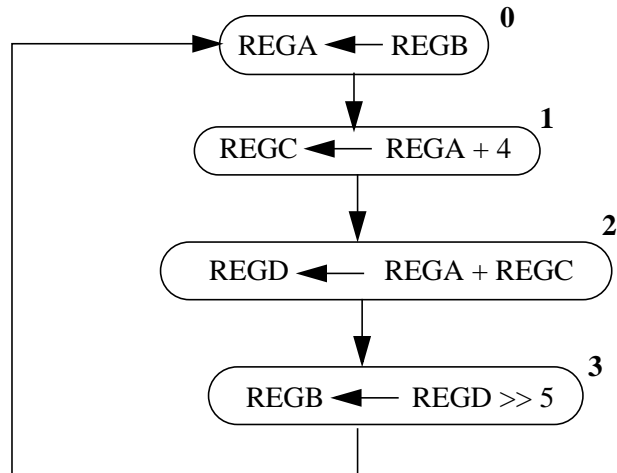
READ :

- i) Related portions of Chapter 4 (except Sections 4.4 through 4.10)
- ii) Related portions of Appendix B
- iii) Related portions of Appendix C
- iv) Related portions of Appendix D

ASSIGNMENT : There are **four** questions.

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

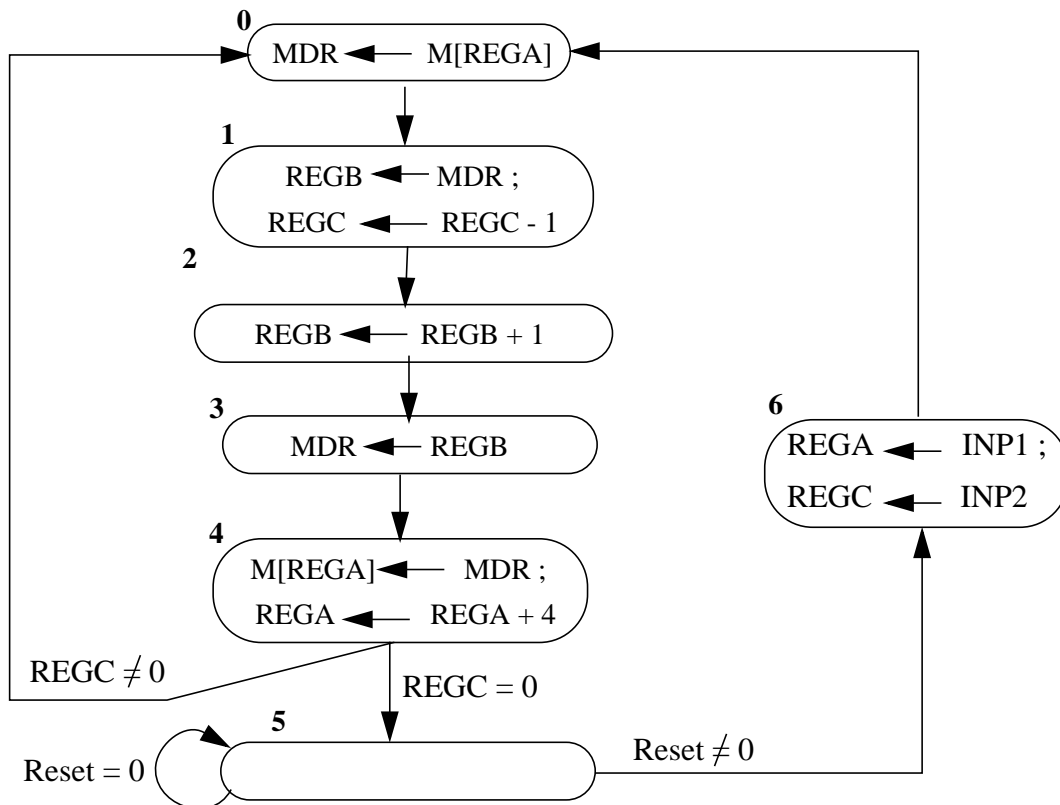
1) The registers of an imaginary digital system are manipulated as shown in the state diagram below. The registers are 32 bits wide.



Show the content of the registers in **HEX** for six (6) clock periods, by continuing the table below. **Show** how the shift operation generates its result, by working on its bits in a separate area.

Clock period	State	REGA	REGB	REGC	REGD
Initial	-----	?	20	?	?
1	0	NS	NS	NS	NS
...	Continue
6

2) Repeat Problem 1 above for the state diagram below. The registers are 32 bits wide.



Assuming that Reset is 0, show the content of the registers in **HEX** for six (6) clock periods, by continuing the table below.

Clock period	State	Reset	REGA	REGB	REGC	MDR	M[10000000]
Initial	-----	-----	10000000	?	2	?	E
1	0	0	NS	NS	NS	NS	NS
....	Continue
6	0

In a few sentences, describe what this digital system does, how it operates.

3) Consider the EMY CPU design with nine (9) integer instructions. We decide to expand the design by adding a MIPS instruction in the MIPS architecture : **JR** (Jump Register). Modify the EMY CPU, except the control unit, to run the JR instruction.

In order to solve this question, you will assume the CPU is a multicycle CPU. You will use the

EMY CPU handout, by xeroxing it and modifying the necessary pages of the xeroxed copy. If you do not want to copy and modify the handout, you can just show the changes to the datapath as done in past exam questions below.

In this question you will modify the EMY CPU so that it can execute the JR instruction. In order to do that you will follow steps III(b) and IV in the Digital System Design Basics handout. 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.

4) Repeat problem 3 above for the JAL instruction.

In order to solve this question, you will assume the CPU is a multicycle CPU. You will use the EMY CPU handout, by xeroxing it and modifying the necessary pages of the xeroxed copy. If you do not want to copy and modify the handout, you can just show the changes to the datapath as done in past exam question 3 below.

In this question you will modify the EMY CPU so that it can execute the JAL instruction. In order to do that you will follow steps III(b) and IV in the Digital System Design Basics handout. 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.

RELEVANT QUESTIONS AND ANSWERS

Q1) The architectural description of a machine language instruction has at least three components : a syntax, an instruction format and what semantics (the architectural operation) the CPU has to perform. An architectural operation is implemented by several microoperations. For example, the “LW” instruction of the EMY computer in its I-format has the following syntax and semantics:

$$\text{LW} \quad \text{Rt, Disp(Rs)} \quad \longrightarrow \quad \text{Rt} \longleftarrow \text{M[Rs + Disp]^{+}}$$

We know that this architectural operation is performed by those microoperations in states 0, 1, 2, 3 and 4.

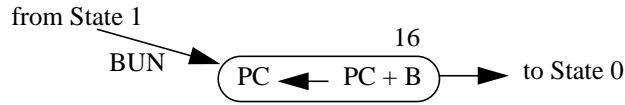
Computer architects at our company have six new instructions to add to the nine-instruction set executed by the EMY CPU of Handout 11. The architecture group assures you that the new instructions do not need a new instruction format. You, as a computer organization expert, do not want to modify the data unit for a new instruction also. They assure you about that too. They have given you the architectural operation list for these new instructions :

$$\begin{array}{ll} \text{I1 : } \text{PC} \longleftarrow \text{PC} + \text{Rt} & \text{I4 : } \text{Rt} \longleftarrow \text{Rs} + \text{Imm}^{+} \\ \text{I2 : } \text{PC} \longleftarrow \text{M[Rs + Disp]^{+}} & \text{I5 : } \text{M[Rd + Rt]} \longleftarrow \text{M[Rs + Rt]} \\ \text{I3 : } \text{M[Rs + Disp]^{+}} \longleftrightarrow \text{Rt} & \text{I6 : } \text{Rs} \longleftrightarrow \text{Rt} \end{array}$$

You want to make sure that the architects’ assurances are correct. So, for every instruction above :

- i) State the syntax and briefly what the instruction does,
- ii) If the architectural operation can be implemented by modifying the high-level state diagram, but without modifying the data unit, modify the high-level state diagram (not in terms of buses), starting at state 16,
- iii) If the architectural operation cannot be implemented without modifying the data unit, state briefly the reason.

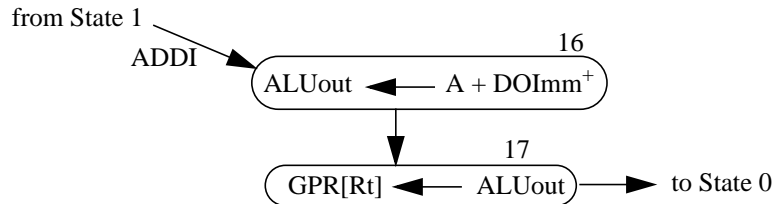
A1) II) i) Unconditional Branch by using “Rt” as an offset ==> BUN Rt
 ii) This operation can be implemented by the original EMY as follows :



I2) i) Jump via memory Indirect. Location “Rs + Displ⁺” has the effective address ==> JMPI Displ(Rs)
 iii) This cannot be implemented since the MRBUS is not connected to MUX6.

I3) i) Exchange the content of memory location “Rs + Displ⁺” and Rt ==> EXCMR Rt, Displ(Rs)
 iii) This cannot be implemented since the MRBUS that has M[Rs + Displ⁺] cannot be saved in the CPU temporarily while Rt is written to the same location : we need another temporary register besides ALUout.

I4) i) The ADD Immediate instruction of EMY from Chapter 4 ==> ADDI Rt, Rs, Imm
 ii) This can be implemented as follows :



I5) i) Copy the content of memory location “Rs + Rt” to memory location “Rd + Rt” ==> MOVMTM Rd, Rt, Rs
 iii) This cannot be implemented since we need at least one more organizational register besides ALUout to keep the second memory address.

I6) i) Exchange register “Rt” with register “Rs” ==> EXCR Rt, Rs
 iii) This cannot be implemented : We need another register besides ALUout to keep one of the architectural registers temporarily while we move the other. Also we need another input to MUX2 to select “Rs” as a Write Register and extra lines are needed to move “Rt” and “Rs” either through the ALU or around the ALU to the temporary registers.

Q2) The EMY CPU is designed as shown in class. We modify its architecture ! We add a new machine language instruction to the instruction set. Its description is as follows :

Syntax : ADDMR Rt, Rs, Offset

Architectural operation : Rt ←--- Rs + M[PC + ([DOImm⁺]<<2)]

The format : The I format.

⇒ Which addressing modes are used for the arguments of the ADDMR instruction ?

- ⇒ Show the modified high-level state diagram.
- ⇒ Show the modified portion of the data unit.

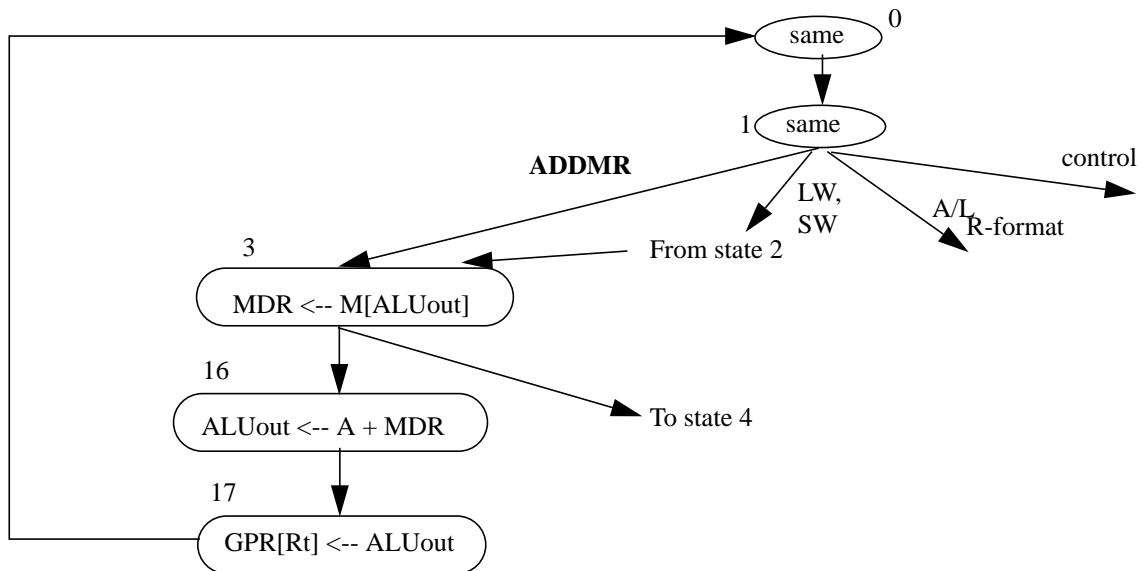
A2) The addressing modes are as follows :

$$Rt \leftarrow Rs + M[PC + ([DOImm^+] \ll 2)]$$

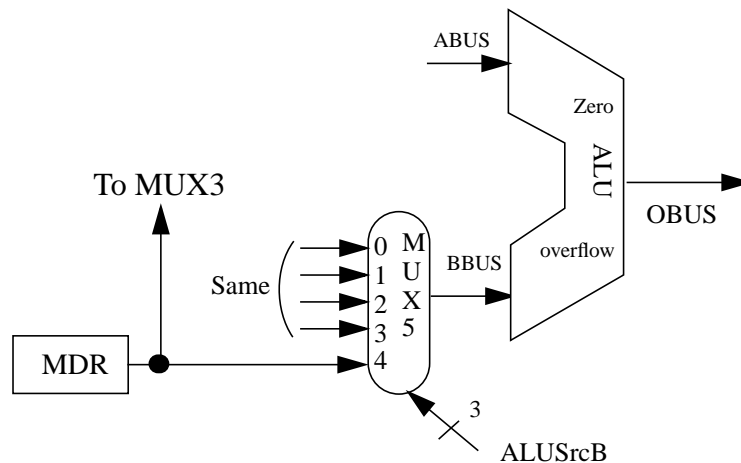
Register
Register
2-byte signed PC-relative

We use the register addressing mode for the destination register argument since it is explicitly specified in the instruction. Similarly, the first source argument is a register and is explicitly specified in the instruction and so the register addressing mode is used. The other source argument is a memory location whose address is calculated by adding PC and the 2-byte signed offset that comes with the instruction. Therefore we use the 2-byte signed PC-relative addressing mode for it.

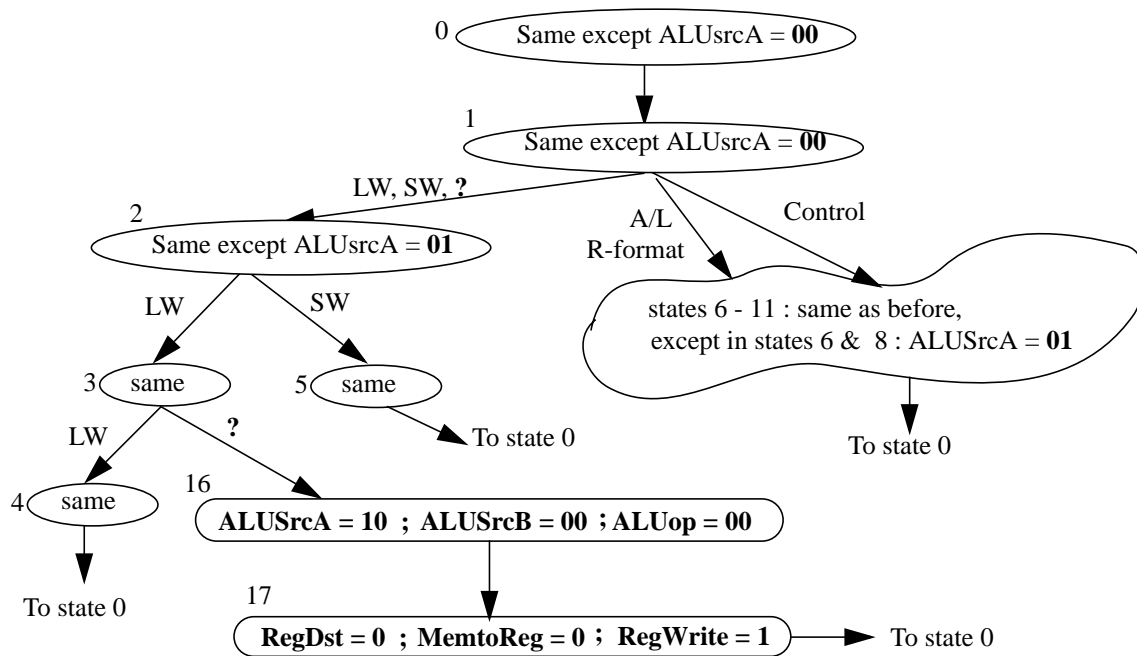
The modified high-level state diagram (not in terms of buses) and the datapath are as follows :



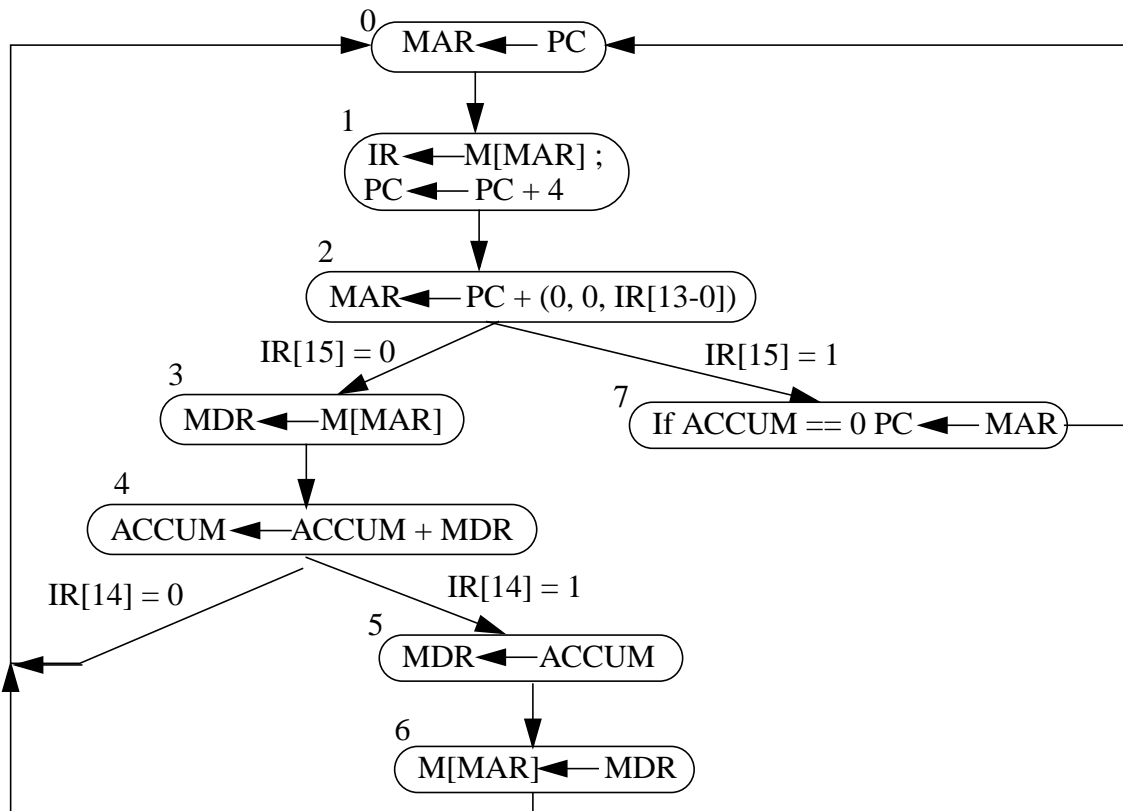
We see that the only new microoperation is in state 16 in which we add MDR and ALUout. The datapath modification is then as follows :



v) The modified EMY CPU low-level state diagram is as follows :



Q4) Consider the state diagram of a digital system shown below :



Assume that all the registers of the digital system are **16** bits long. The memory has 2^{16} bytes and 16 bits per location. By giving explanations, **show** the contents of registers and memory locations used by the above state diagram for **six** (6) clock periods. For that continue with the following table :

Clock period	State	PC	MAR	IR	MDR	ACCUM	M[200]	M[3204]
Initial	-----	200	?	?	?	1A	3000	7
1	0	NS	NS	NS	NS	NS	NS	NS
....
6

A4) The table is continued below :

Clock period	State	PC	MAR	IR	MDR	ACCUM	M[200]	M[3204]
Initial	-----	200	?	?	?	1A	3000	7
1	0	NS	NS	NS	NS	NS	NS	NS
2	1	NS	200	NS	NS	NS	NS	NS
3	2	204	NS	3000	NS	NS	NS	NS
4	3	NS	3204	NS	NS	NS	NS	NS
5	4	NS	NS	NS	7	NS	NS	NS
6	0	NS	NS	NS	NS	21	NS	NS

In state 2, MAR is transferred PC (204) plus the rightmost 14 bits of IR (11 0000 0000 0000 = 3000) catenated with two zeros (0011 0000 0000 0000 = 3000). MAR is transferred 204 + 3000 = 3204

We do state 3 after state 2 because the leftmost bit of IR (IR[15]) is zero : **0**011 0000 0000 0000.

We do state 0 after state 4 because IR[14] is also zero : **00**11 0000 0000 0000

In state 4, ACCUM is transferred ACCUM (1A) plus MDR (7). ACCUM is transferred 21.

Q5) Consider the following mnemonic machine language subroutine :

```

40050C LW    R8, 0(R9)
400510 SW    R10, 0(R9)
400514 ADD   R10, R8, R0
400518 JR    R31

```

Continue filling in the following table until the **ID** cycle of the “JR” instruction is completed :

cp	State	PC Source	PC	IR	ALU SrcB	Reg Write	A	B	R8	R9	R10	ALUout	M[10002000]
Initial	---	---	40050C	?	---	---	?	?	?	10002000	2F7	?	8A
1	0	00	NS	NS	01	0	?	?	NS	NS	NS	?	NS
2	1	00	400510	LW R8, 0(R9)	11	0	?	?	NS	NS	NS	400510	NS
...

Note that the values are shown right before the end of the clock period. Briefly describe what the subroutine does (in one sentence).

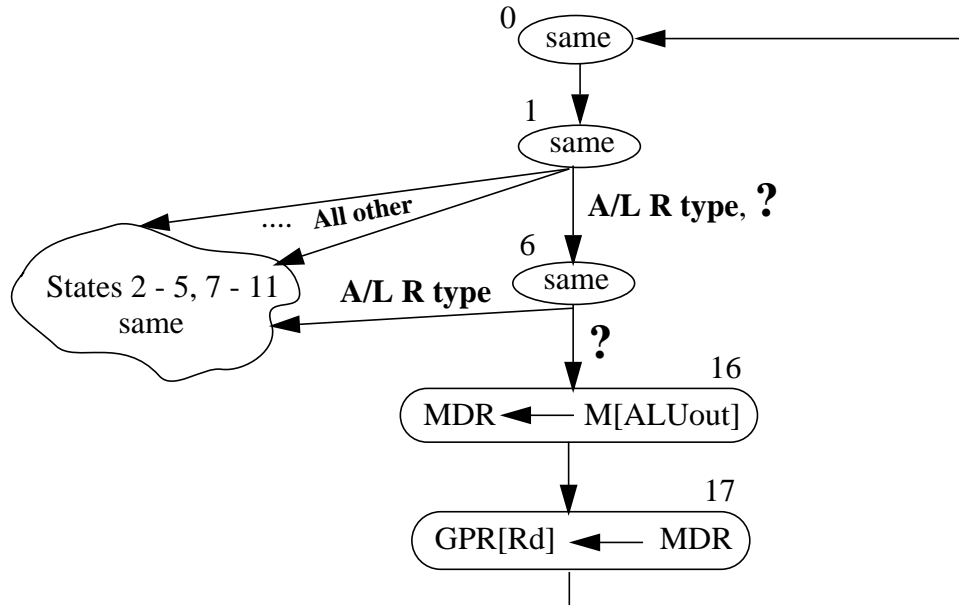
A5) The table is as follows :

cp	state	PC Source	PC	IR	ALU SrcB	Reg Write	A	B	R8	R9	R10	ALUout	M[10002000]
Initial	--	--	40050C		--	--	?	?	?	10002000	2F7	?	8A
1	0	00	NS	NS	01	0	?	?	NS	NS	NS	?	NS
2	1	00	400510	LW R8,0(R9)	11	0	?	?	NS	NS	NS	400510	NS
3	2	00	NS	NS	10	0	10002000	?	NS	NS	NS	400510	NS
4	3	00	NS	NS	00	0	10002000	?	NS	NS	NS	10002000	NS
5	4	00	NS	NS	00	1	10002000	?	NS	NS	NS	?	NS
6	0	00	NS	NS	01	0	10002000	?	8A	NS	NS	?	NS
7	1	00	400514	SW R10,0(R9)	11	0	10002000	?	NS	NS	NS	400514	NS
8	2	00	NS	NS	10	0	10002000	2F7	NS	NS	NS	400514	NS
9	5	00	NS	NS	00	0	10002000	2F7	NS	NS	NS	10002000	NS
10	0	00	NS	NS	01	0	10002000	2F7	NS	NS	NS	40080B	2F7
11	1	00	400518	ADD R10,R8,R0	11	0	8A	0	NS	NS	NS	400518	NS
12	6	00	NS	NS	00	0	8A	0	NS	NS	NS	424598	NS
13	7	00	NS	NS	00	1	8A	0	NS	NS	NS	8A	NS
14	0	00	NS	NS	01	0	8A	0	NS	NS	8A	424598	NS
15	1	00	40051C	JR R31	11	0	?	0	NS	NS	NS	40051C	NS

In clock period 15, the value of register A is the value of R31. As we know R31 contains the return address from the subroutine.

The subroutine exchanges a memory location whose address is “R9 + Displ⁺” with register R10. That is, it implements I3 of Past Exam Question 1 above.

Q6) Assume that the EMY CPU **high-level** state diagram is modified to be able to run a **new** machine language instruction as follows :



Assume that in state 6, an **addition** is performed for this **new** instruction.

- i)** What is the new instruction ? That is, determine its syntax, semantic, etc. If there is a **new** addressing mode that is **not** discussed in class, indicate so.
- ii)** Modify the EMY CPU **datapath** accordingly. How long does it take to run the new instruction ?
- iii)** Modify the EMY CPU **low-level** state diagram accordingly.

A6 i) On the high level state diagram, we see that we read a memory location and store the content on a register :

It is a Load instruction : Load Word via Register Indirect (**LWRI**)

Syntax : LWRI Rd, (Rs, Rt)

Semantics : Rd M[Rs + Rt]

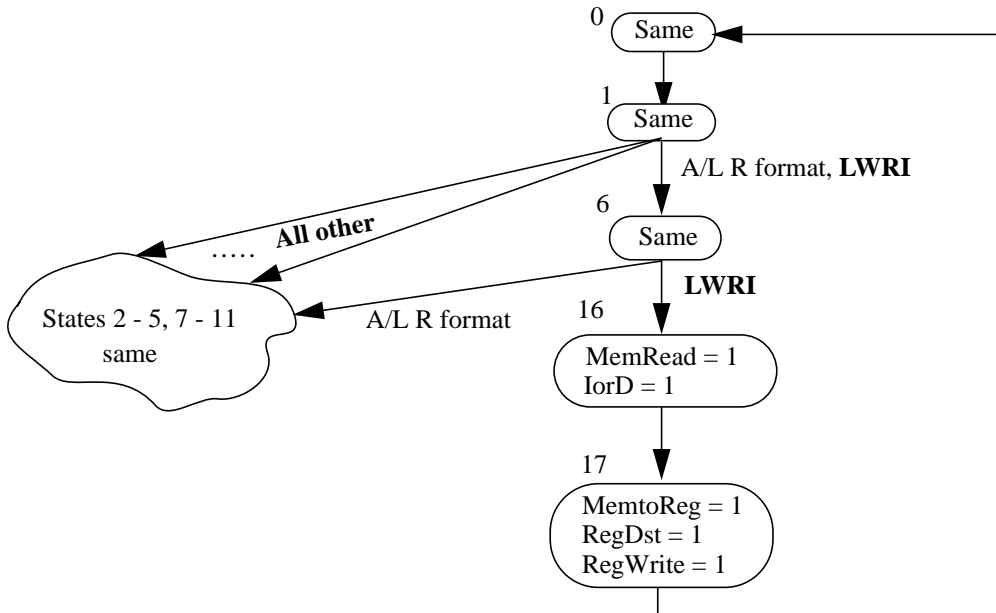
Format :

- It uses the **R** format since register Rd is specified.
 - It **has** two arguments.
 - The destination argument is a GPR register which is explicitly specified by the instructions. Therefore, the **Register** addressing mode is used.
 - The source argument is a memory location content. The address of the location is the sum of Rs and Rt. Therefore, it is the **Register Indirect Indexed** addressing mode.
 - This is a **new** addressing mode, **not** discussed in class.
 - We make **two** memory accesses for the new instruction :
 - One to fetch the instruction (state 0)
 - One to read a data element from the memory (state 16)

ii) The new instruction does **not** need any change in the datapath, given the high-level state diagram.

The new instruction takes **five** clock periods to run. That is, CPI_{LWRI} is **5** since we trace states 0, 1, 6, 16 and 17

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



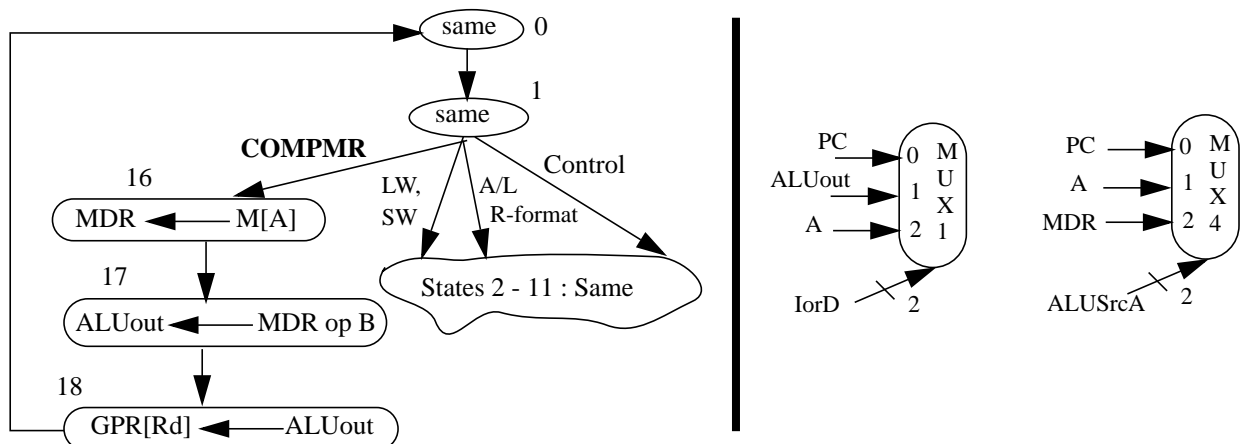
Q7) Consider the following instruction that does **not** exist in the EMY instruction set :

COMPMPR Rd, (Rs), Rt # If M[Rs] < Rt then Rd ← 1 else Rd ← 0

i) Modify the EMY CPU **high-level** state diagram and the **datapath** to be able to run the new instruction, as done in class. What is CPI_{COMPMPR} ? How many memory accesses are made for this new instruction ?

ii) The **new** datapath allows new microoperations. List at least **four (4)** new microoperations.

A7) i) The modified **high-level** state diagram and **datapath** of this high-speed implementation are shown below.

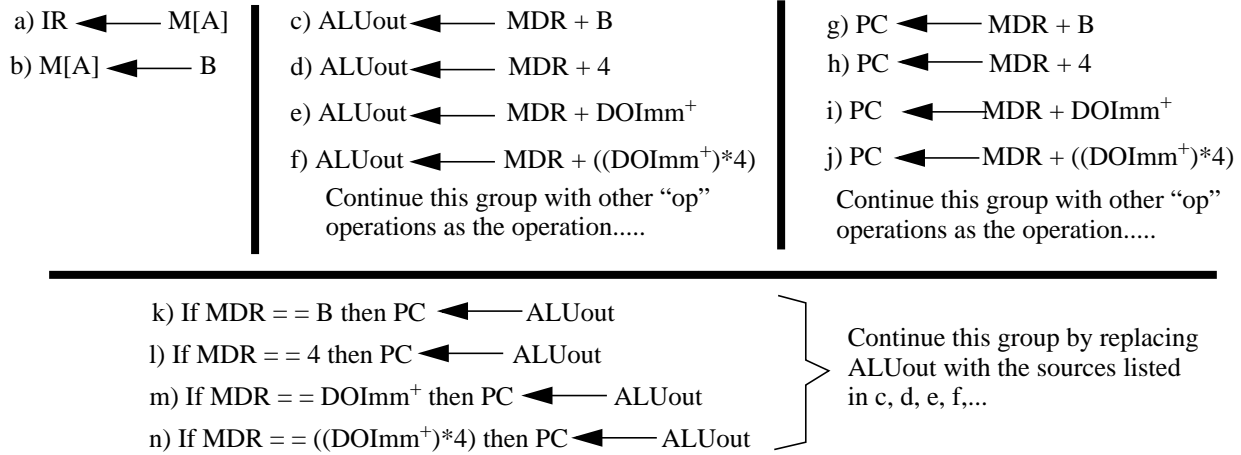


We perform an “slt” operation in state 17 for the COMPMPR instruction.

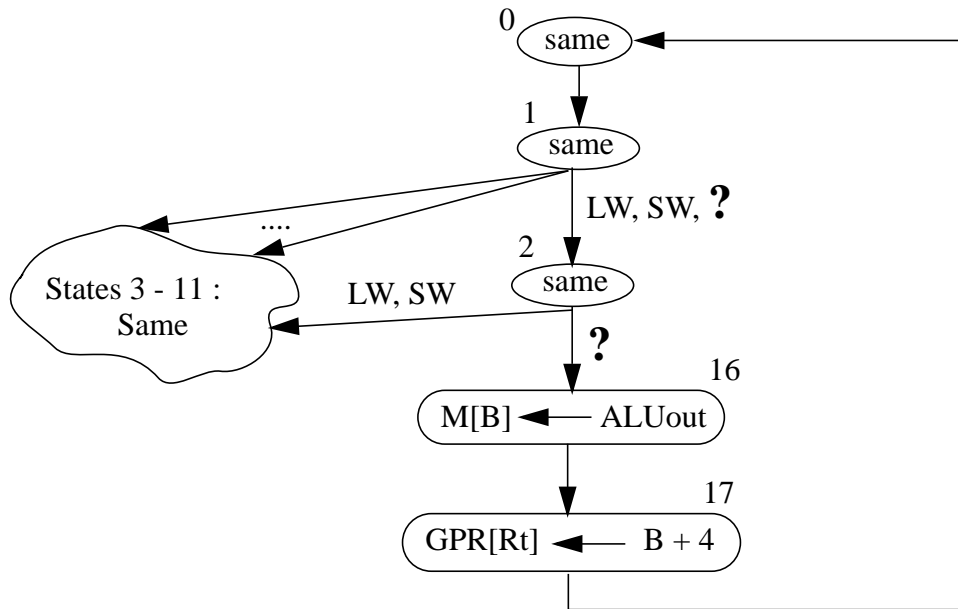
CPI_{COMPMPR} is 5 since we trace states 0, 1, 16, 17 and 18

We make **two** memory accesses for the new instruction :
 - One to fetch the instruction (state 0)
 - One to read a data element from the memory (state 16)

ii) Two new microoperations are shown in states 16 and 17 above. Below is a partial list of remaining new microinstructions :



Q8) Assume that the EMY CPU **high-level** state diagram is modified to be able to run a **new** machine language instruction as shown below :



a) What is the new instruction ? That is, determine its syntax, semantics, etc. If there is a **new** addressing mode that is **not** discussed in class, indicate so.

- b)**
- i)** Modify the EMY CPU **datapath** accordingly.
 - ii)** How long does it take to run the new instruction ? Explain.

A8) a) On the high level state diagram, we see that we add register Rs and a signed immediate element and store the result in memory pointed by register Rt and then we automatically update register Rt :

It is an Add instruction that writes to the memory (**ADDRIM**) : Add Register Immediate Memory

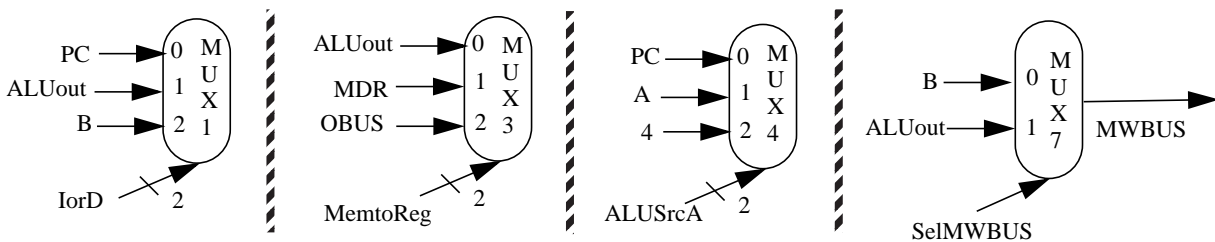
Syntax : ADDRIM (Rt)++, Rs, Imm

Semantics : M[Rt] Rs + Imm⁺ then Rt Rt + 4

Format, etc. :

- It uses the **I** format since an immediate data element is used.
- It has **six** arguments.
 - The destination argument of the first addition operation is a memory location. The address of the location is indicated by Rt. Therefore, it is the **Register Indirect** addressing mode.
 - This is a **new** addressing mode, **not** discussed in class
 - The sources of the first addition operation are a register and an immediate data element. We use the **Register** and **2-byte signed Immediate** addressing modes, respectively.
 - All the arguments of the second addition are implied and are using the **Implied** addressing mode. The destination argument is register Rt. One of the source arguments is also register Rt. The other source is a constant which is 4.
- We make **two** memory accesses for the new instruction :
 - One to fetch the instruction (state 0)
 - One to write a data element to the memory (state 16)

b) i) The new instruction requires the following changes in the datapath :



ii) The new instruction takes **five** clock periods to run.

That is, CPI_{ADDRIM} is **5** since we trace states 0, 1, 6, 16 and 17

Q9) The EMY CPU is modified to run new instructions. When the new CPU runs instructions, its control signals are observed for four clock periods. Only the following control signals are **1 (one)** in these four clock periods :

clock period	Control signals that are 1 (one)
257	MemRead, IRWrite, ALUSrcB0, PCWrite
258	ALUSrcB
259	----
260	ALUSrcA, ALUSrcB, ALUop0, PCSource0, PCWriteCond

Note that ALUSrcB has two wires, named ALUSrcB1 and ALUSrcB0, and so on....

State which state and which microoperation(s) in that state happen(s) in which clock period. Your explanation should be in the following form :

In clock period ... ALUSrcA is 1, ... therefore, we have the following microoperation(s) and the following state ...

A9) The states are as follows :

clock period	State and microoperations
257	MemRead = 1, IRWrite = 1, ALUSrcB = 01, PCWrite = 1 exactly the same as state 0 of the low-level state diagram IR \leftarrow M[PC], PC \leftarrow PC + 4
258	ALUSrcB = 11 exactly the same as state 1 of the the low-level state diagram ALUout \leftarrow PC + ((DOImm ⁺) << 2)
259	All control signals are 0, that is ALUSrcA = 0 => ABUS = PC ALUSrcB = 00 => BBUS = B ALUop = 00 => Add a new state, state 16 , of the low-level state diagram ALUout \leftarrow PC + B
260	ALUSrcA = 1 => ABUS = A ALUSrcB = 11 => BBUS = (DOImm ⁺) << 2 ALUop = 01 => Sub PCSource = 01 => PCBUS = ALUout PCWriteCond = 1 => PC is written if the subtraction results in zero = 1 a new state, state 17 , of the low-level state diagram If (A == ((DOImm ⁺) << 2)) then PC \leftarrow ALUout