Name:

Midterm Exam, CS151B/EE116C, Spring 2016

97

79

General Guidelines:

- You can use a simple calculator, the course textbook, printouts of: pp. 318-340 from chapter 5 of the 3rd edition, pp. D3-D27 from Appendix D of the textbook, and the class notes posted on the class web page. **NOTHING ELSE.**
- It is **CHEATING** to use any other notes (**including your own notes**), textbooks, or solutions to homework/example/old-exam problems.
- You must return the exam with the answers on the exam pages.
- Wherever there is space for an explanation, provide explanations and justifications for your answer and clearly state all your assumptions.
- Produce clear, well-organized answers! Points **will** be taken off for disorganized messy writing.
- Unless specified otherwise, when writing MIPS **assembly code**, you **must** use **only** the $reg-number notation (e.g., $22, $5) to specify a register. Any other notation will be considered incorrect. Furthermore, you cannot use any pseudoinstructions.
- **Your answers must fit in the space provided. You must not write on the margins or on the backs of pages. If lines are drawn, your answers must be written on the lines.**

Excellent work!

(1) Write your name at the top of this page and every other page not *stapled* to this page.

1

(2) Consider the implementation of the control unit for the multicycle MIPS shown in Figures D.4.1-D.4.5 (pages D-23 to D-26 of Appendix D) of the textbook. Due to an implementation mistake, the value in address 100011 of Dispatch ROM 1 (Figure D.4.3) is 0011 instead of 0010.

12

8/12

Explain **in full detail** what will be the consequences of this mistake when the processor executes programs — how will it change the behavior of the processor **as observed by a user/programmer** who does not know and does not care how the processor is implemented internally? Be sure to clearly identify **each and every** consequence of this fault with as much **detail** and **specificity** as possible. Explain your answer.

It will skip memory address computation for lw // This way it will take the old value of ALUOut, which would be what the control predicted for $beq. lw would load an incorrect value from memory into register -3 Not specifying correct Addr.

Name:

16 (3) This problem deals with the multicycle MIPS implementation shown in Figure 5.28, page 323, and Figure 5.28, page 338, of Chap 5, **3rd Ed** (pages 4.13 and 4.26 in the class notes). The main part of the control unit is implemented as shown on page 4.53 in the notes (Figure D.3.2, page D-10, in Appendix D of the textbook).

8/16

In the table below, each row is for a specific timing characteristic of one of the components, and each column is for one of the states. Your task is to write an **X** in every square for which there is a **possibility** that the corresponding timing characteristic can affect the minimum cycle time required for the corresponding state.

Timing Characteristic	State Number									
	0	1	2	3	4	5	6	7	8	9
A reg setup		X								
B reg clk-to-Q										
PC setup	X									
PC clk-to-Q	X									
Sign extend latency										
IorD MUX latency	X			X						
ALU Control latency	X	X	X				X			

15 (4) Consider the following C function:

15/15

```
int doit(int aa, int bb, int cc)  {
   if  (aa < bb)
      return(funa(aa,cc)+funb(bb,cc)) ;
   else
      return(bb+cc) ;
}
int funa(int x, int y)  {
   return(x - y - y) ;
}
int funb(int x, int y)  {
   return(x + x + y) ;
}
```

On the next page is a MIPS translation of the program above. In some instructions, a subset (but not all) of the instruction fields are missing (indicated by underlines). Fill in all these blank instruction fields with the correct values. In some lines in the listing on the next page, the "instruction" consists entirely of underlines. These lines should either be left alone, indicating that there is no need for an instruction there, or an assembly instruction is missing and you must provide it. You can only "fill in" underlined fields — you cannot add new instructions anywhere you want!

The instructions that include at least one non-blank field are all needed. You cannot convert any of them to "do nothing" (NOP) instructions.

The code **must** follow the standard MIPS calling conventions. The code should be as efficient as possible — minimize the number of instructions and the number of memory accesses.

Assume that when `doit` is called, `aa` in `$a0`, `bb` in `$a1`, and `cc` in `$a2`.

For this problem, **you must use the symbolic names for registers**: `$t1`, `$a2`, etc. You must use only assembly instructions — do not use pseudoinstructions.

Name:

```
doit:   slt   $t0,$a0,$a1
        bne   $t0,$zero,foo
        ___   _____,_____,_____
        add   $v0,$a1,$a2
        jr    $ra
foo:    addi  $sp,$sp,-12
        sw    $ra,8($sp)
        sw    $a1,0($sp)
        or    $a1,$a2,$zero
        sw    $a2,4($sp)
        jal   funa
        lw    $a0,0($sp)
        lw    $a1,4($sp)
        sw    $v0,4($sp)
        jal   funb
        lw    $t0,4($sp)
        add   $v0,$v0,$t0
        lw    $ra,8($sp)
        addi  $sp,$sp,12
        jr    $ra
funa:   ___   _____,_____,_____
        sub   $v0,$a0,$a1
        sub   $v0,$v0,$a1
        jr    $ra
funb:   ___   _____,_____,_____
        add   $v0,$a0,$a0
        add   $v0,$v0,$a1
        jr    $ra
```

15/15.

Good!

Name:

(14) (5) Consider the multicycle MIPS implementation. Assume that it is possible to change this implementation so that reliable operation can be maintained at a higher clock rate. However, as a result, the execution times of the `lw` instruction and the `beq` instruction will have to be increased by one cycle.

14/14

In the original implementation, the maximum clock rate for reliable operation is 1.9GHz. The workload of the system is characterized by the following table:

instruction type	dynamic frequency	CPI	NewCPI
R-type	48%	4	4
lw	22%	5	6
sw	10%	4	4
beq	15%	3	4
j	5%	3	3

Your task is to determine whether this design change would be worthwhile. Specifically, you need to determine what is the minimum clock rate for the new design that would result in higher system performance.

You must show and explain every step of the computation.

$Avg\,CPI_{old} = \frac{48}{100}\times 4 + \frac{22}{100}\times 5 + \frac{10}{100}\times 4 + \frac{15}{100}\times 3 + \frac{5}{100}\times 3 = 4.02$

$Avg\,CPI_{new} = \frac{48}{100}\times 4 + \frac{22}{100}\times 6 + \frac{10}{100}\times 4 + \frac{15}{100}\times 4 + \frac{5}{100}\times 3 = 4.39$

For same no of instructions, We want

$$CPI_{new}\times\frac{1}{CR_{new}} < CPI_{old}\times\frac{1}{CR_{old}}$$

$$4.39\times\frac{1}{CR_{new}} < 4.02\times\frac{1}{1.9\times10^9}$$

$$CR_{new} > \frac{4.39}{4.02}\times 1.9\times10^9$$

$$CR_{new} > 2.07\times10^9\ Hz = 2.07\ GHz$$

As we want to reduce execution time, CR_{new} would have to be bigger than 2.07 GHz

Name:

13

9/13

(6) Consider the single cycle MIPS implementation shown in Figure 4.24 (page 271) and described in Section 4.4 in the book. The ALU implementation is shown in Figure B.5.12 (page B-36).

In the implementation shown in Figure B.5.12, the Bnegate control signal is connected to the Binvert inputs of all the ALU bit slices and to the Carryin of slice `ALU0`.

Due to an implementation mistake, instead of connecting the Bnegate signal to the Carryin of slice `ALU0`, the **Ainvert** signal is connected to the Carryin of slice `ALU0` (Ainvert is also connected to the Ainvert inputs of all the ALU bit slices, as shown in the figure).

A) List all the instructions that will be affected by this implementation mistake.

R-type (subtraction) & slt, beq

B) Explain your answer to Part A.

subtraction affected as now it will compute a - b - 1. b will get inverted but not negated. slt will be affected as slt uses a - b.

Do not write anything in this space (or below)

Name:

29

(7) Consider the multicycle implementation shown in Figure 5.28, page 323, and Figure 5.37, page 338, of Chap 5, **3rd Ed**.

24/29

Your task is to modify this implementation so that it supports a new instruction, `clw` (conditional load word). The instruction consists of **two** words, in the following format:

101100	Rs	Rt	Rd	00000000000
address				

The `clw` instruction compares the value in the **Rs** register to the value in the **Rt** register. If the two values are equal, the value in memory, at the address contained in the second word of the instruction, is copied to register **Rd**. Note that the second word of the instruction is thus **a pointer** to the value to be loaded. If the values in the Rs and Rt registers are not equal, the instruction must not modify register Rd.

Your modifications must meet the following requirements:

1) All the existing functionality must be maintained.
2) You must not increase the execution time of any of the other instructions.
3) You cannot speed up any of the building blocks used to construct the implementation in Figure 5.28.
4) You cannot add another ALU of any kind (this means that you also cannot add a new adder or comparator).
5) You cannot add any new registers.
6) You cannot add a new memory.

Within the above constraints, your first priority is to minimize the implementation complexity. Your second priority is to minimize the execution time of the new instruction.

If you need to add any new building blocks to the datapath or the control, or replace an existing module with a new module that has some additional capabilities, it is **your responsibility** to make sure that it is completely clear **exactly** how each wire is connected to the new building block. If the new building block is not a copy of an already existing building block, you **must** draw the implementation of the new building block. This means that, for example, you do **not** have to show the implementation of standard MUXes or copies of existing registers. Make sure that your drawings are not messy.

A) Explain the basic idea of your modifications in 2-4 clear sentences. You **must** specify how many cycles it will now take to execute the `clw` instruction.

B) Show the necessary modifications to the datapath. For your convenience, a copy of the datapath is on page 8 and you must use it (together with other drawings, as you wish) to show your modifications.

C) Are any new control signals required? If so, list them with an explanation and identify them on the datapath diagram.

D) On page 9, there is the HDL description of the original implementation. Modify it to reflect your new implementation.
Be sure that it is clear exactly where any new code is placed.

E) On page 10, there is the original state diagram of the control unit. Modify it to reflect your new implementation.

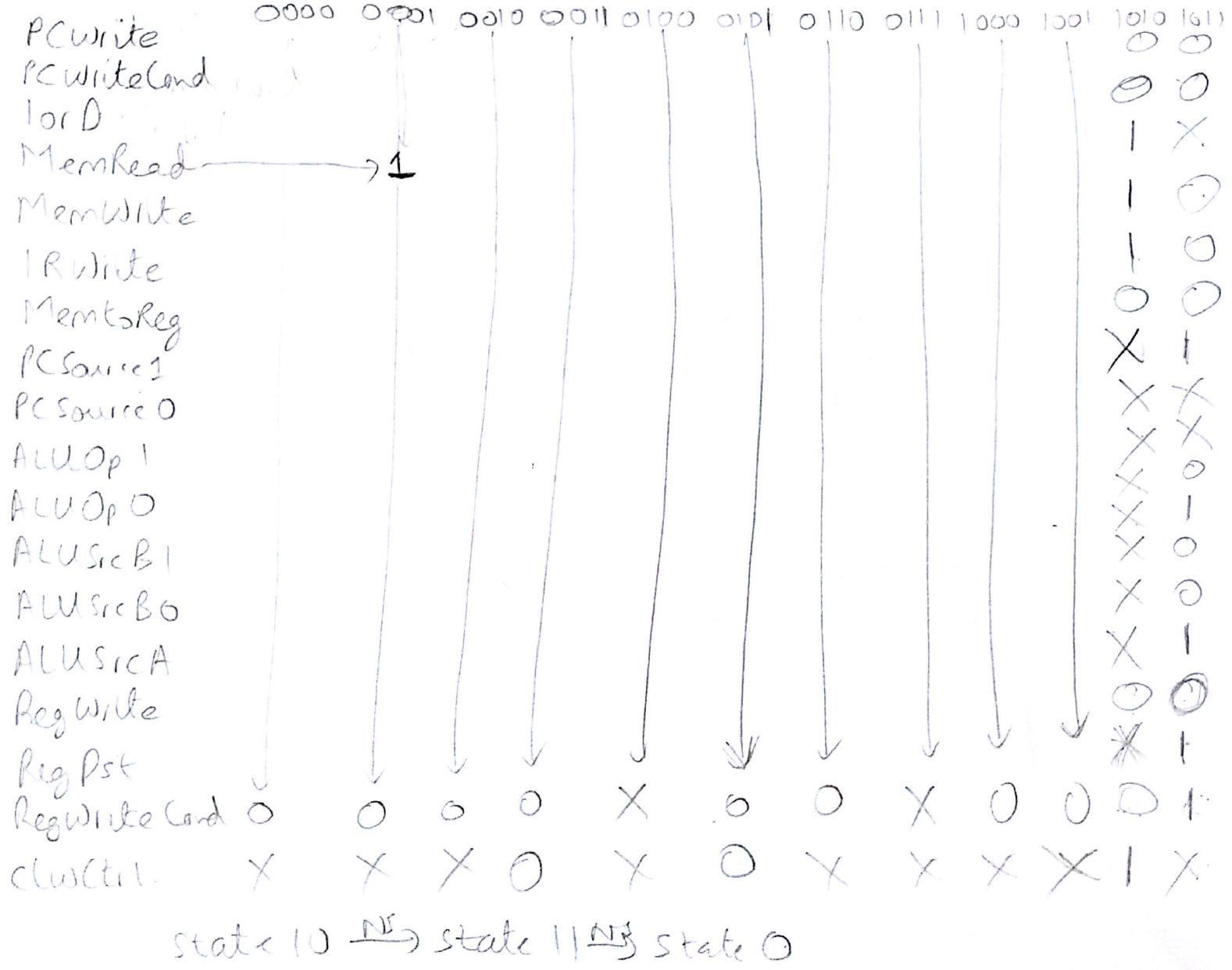
Name:

The answer to Part A must be written here.

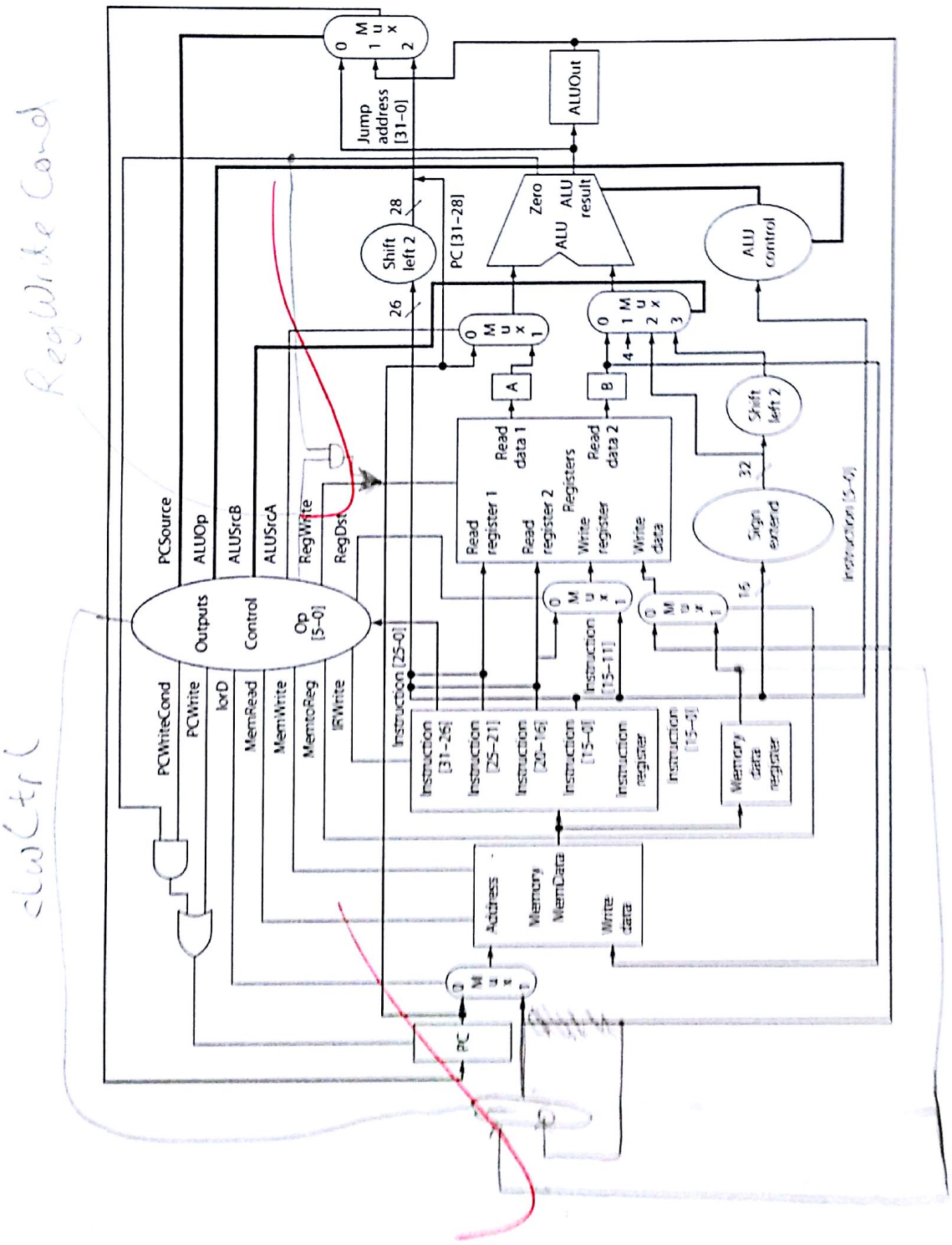
We read the second word into the MDR in state 1.
We add an AND gate and a OR gate near RegWrite & add a control signal RegWriteCond
We will also take a wire from MDR back into memory, in order to access value to be loaded.

With this implementation, instruction `clw` executes in 4 cycles.

The space below is for part of your answer to Part C (the list of control signals) as well as for showing the implementation of any additional circuitry you need (part of your answer to Part B).



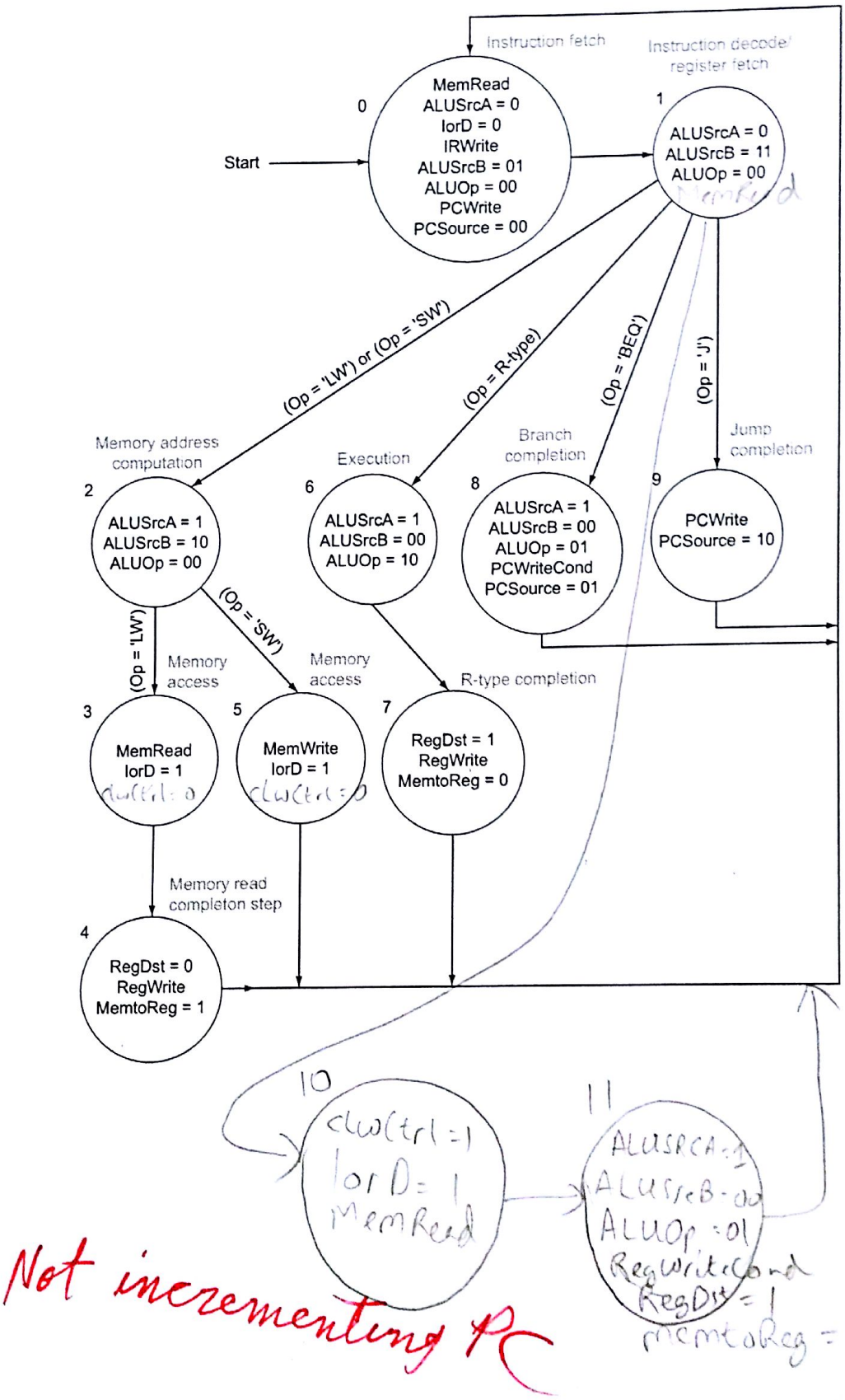
Name:



Name:

```
IR ← M[PC];  PC ← PC + 4;  Next;

A ← R[IR25..21];  B ← R[IR20..16];

ALUout ← PC + SignExt(IR15..0 || 00);  Next;
MDR ← M[PC] Next,
switch(IR31..26) {
    case 0:  /* R-format */
        switch(IR5..0) {
            case 32:  /* add */
                ALUout ← A + B;  Break;
            case 34:  /* sub */
                ...

        }

        Next ;
        R[IR15..11] ← ALUout;  Break;


    case 35:  /* lw */

        ALUout ← A + SignExt(IR15..0);  Next;
        MDR ← M[ALUout];  Next;
        R[IR20..16] ← MDR;  Break;


    case 43:  /* sw */
        ...

case44: /* clw */
MDR ← M[MDR]; Next;
ALUOut ← A - B;
if(A==B) R[IR20..16] ← MDR; Break;

PC ← PC + 4

}

Next ;
```

Name:



Not incrementing PC