UNIVERSITY OF CALIFORNIA, LOS ANGELES UCLA

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO SANTA BARBARA • SANTA CRUZ

CS M151B / EE M116C
Midterm Exam

Before you start, make sure you have all 11 pages attached to this cover sheet.

Please put your name at the top of each page.

All work and answers should be written directly on these pages to help in assigning partial credit, use the backs of pages if needed. Be as neat and clear as possible.

This is an open book, open notes quiz – but you cannot share books, notes, or calculators.

NAME: _____

ID: _____

Please do not write anything in the area below on this page:

Problem 1: _____ (15)

Problem 2: _____ (12)

Problem 3: _____ (6)

Problem 4: _____ (20)

Problem 5: _____ (20)

Problem 6: _____ (20)

Total: _____

Fall 2004 NAME _____

1. ***Execution Time (15 points):*** You are asked to choose between two approaches to reducing the impact of loads on processor performance on the **multicycle datapath**. For this problem, assume that R-type instructions (*add*, *sub*, *and*, *or*, and *slt*) account for 45% of all executed instructions, stores account for 15%, loads account for 20%, and branches account for 20%. Further assume that the application we are considering executes one million instructions and that the processor uses a 4 GHz clock.
   a. What is the execution time for the application we are considering on the machine described above?

ET_a is _____1.0e-3 s_____

CPI = .20 * 5 + (.45+.15) * 4 + .20 * 3 = 4
Cycle time = 0.25e-9
IC = 1.0e6

ET = CPI x cycle time x IC = 1.0e-3

   b. One architect proposes a technique that increases load execution to 6 cycles (from 5 in the original datapath). However, this technique increases the clock rate of the processor by 15%. Calculate the execution time for the given workload with this change.

ET_b is ____0.91e-3 s_____

CPI = .20 * 6 + (.45+.15) * 4 + .20 * 3 = 4.2
Clock rate = 4GHz * 1.15 = 4.6 GHz
Cycle time = 0.217e-9
IC = 1.0e6

ET = 0.91e-3

c. Another architect observes that many load instructions do not use their offset field (i.e. the offset is zero) and are closely followed by an add instruction that increments the result of the load by some other register. For example:

lw r5, 0 (r8)
add r5, r5, r7

This architect proposes to replace such instances with a single new instruction, *law*, that will perform the load and add together:

law r5, r8, r7

This instruction will have the following functionality:

R[r5] = M[R[r8]] + R[r7]

The *law* instruction takes 5 cycles, and has no impact on the cycle time of the processor. Assume that 50% of loads can make use of this optimization. Calculate the execution time when using this approach.

ET_c is _____.9e-3 s_____

50% of loads use this optimization – so 50% of the 20% of instructions that are loads means that 10% of all instructions will be law.

Each law instruction replaces an lw and an add. A law takes the same number of cycles as a lw – so the 5 cycle instructions will stay the same. But the 4 cycle instructions will be reduced.

$$CPI = \frac{.20 * 5 + .20 * 3 + .50 * 4}{.9} = 4$$

you could also have intuitively seen that this optimization only removes 4 cycle instructions (the law and lw effectively contribute the same amount to CPI) – and no matter how many 4 cycle instructions are removed, the CPI will still remain 4 unless the balance of 3 and 5 cycle instructions change.

IC = .9e6 <- IC does drop by 10%

Cycle time = .25e-9

ET = .9e-3

2. ***Impacting Performance (12 points):*** For the following, circle how the proposed change will impact CPI, # of instructions executed, and cycle time (i.e. an answer might be that CPI will increase, # of instructions executed will remain the same, and cycle time MAY decrease). Consider the **multicycle datapath**. Provide a brief justification of each answer and make sure you address each of the three components by clearly circling **ONE** option from each group.

   Loads can only use the base addressing mode – the base+offset addressing mode is completely removed from the ISA. Any load that requires base+offset addressing will need to use an add instruction before the load to perform the addition.

   CPI (will) / may increase / (decrease) / stay the same
   Every load will require 4 cycles instead of 5. (We'll get rid of the stage/step/cycle #2 for loads)*

   # instructions executed (will) / may (increase) / decrease / stay the same
   Every load that require base+offset calculation must be "augmented" by an add.**

   cycle time (will) / may increase / decrease / (stay the same)

   Cycle time was determined as a maximum of memory access time, ALU time and register file access time. Even if ALU were the greatest of those three, cycle time wouldn't change, since we'll need ALU for other instructions (R-type, beq, sw)

   *Assuming we have loads in our program. Otherwise CPI is unchanged.
   **Assuming we have Loads that need base + offset. Otherwise # instructions is unchanged.

3. ***Cycle Counting (6 points):*** For the following code:

```
add $s1, $s2, $s3
add $s4, $s5, $s6
lw  $s7, 0 ($t8)
```

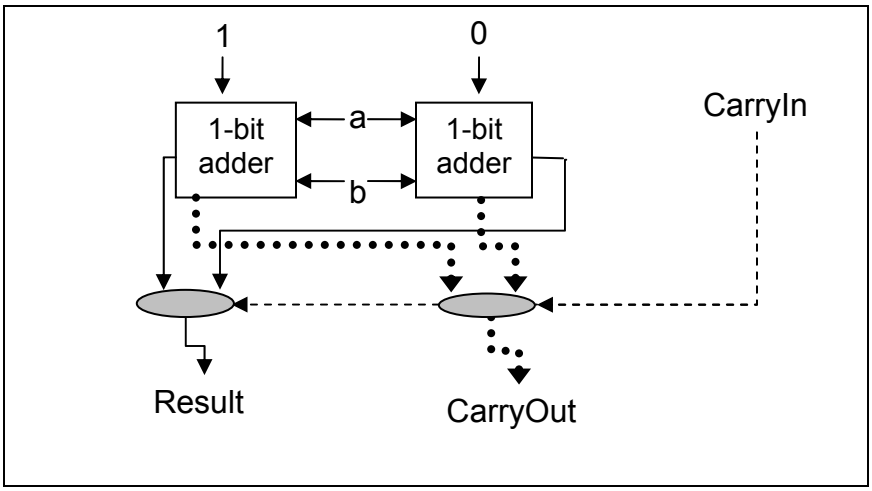
   How many cycles will this take to execute on:

   a) the single cycle datapath we explored in class
      3

   b) the multicycle datapath we explored in class
      13

   c) the pipelined datapath we explored in class (assume NO hazards of any kind)
      7

4. ***Carry Select Adder (20 points):*** A 1-bit *carry select adder* is shown below – a 1-bit adder is replicated to compute both possibilities for CarryIn (1 and 0). The CarryIn signal is then used to choose which of the Result and CarryOut signals to output for that bit of the computation.



We will evaluate the use of carry selection with carry lookahead – the hope is that our carry lookahead hardware will accelerate the selection process of the carry select adder. The following page shows two figures that should be familiar to you, a 4-bit carry lookahead adder and a 16-bit carry lookahead adder that is made up of four 4-bit carry lookahead adders. We have already examined these adders in class. For this problem, assume that each 1-bit adder of the 16-bit carry lookahead adder is a 1-bit carry select adder. Further assume that inverters have a delay of **T** and logic gates AND, OR, XOR, NAND, NOR, and XNOR have delays of **2T** associated with them (where **T** is measured in seconds). Treat the multiplexor as a special case that has a delay of **M** (where **M** is also measured in seconds).
*** NOTE THAT THESE DELAYS ARE DIFFERENT FROM WHAT WE DID IN CLASS! ***

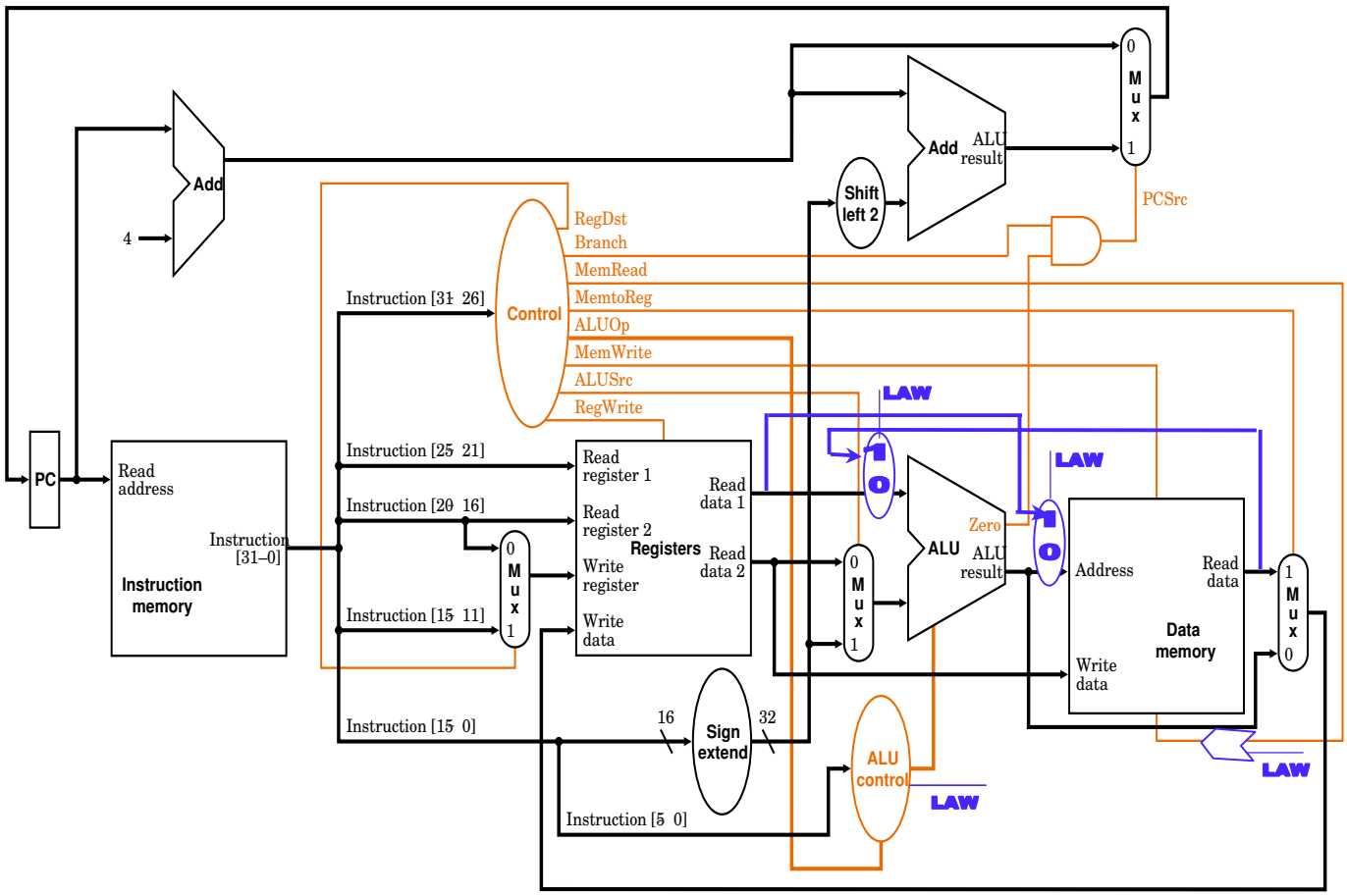
a. What is the estimated delay for the 16-bit carry lookahead adder made up of carry select adders (should be in terms of **T** and **M**)? So each 1-bit adder will be a carry select adder. You will want to measure the maximum delay to compute the output bits (16 Result bits, and 1 CarryOut bit). Show your work on the diagram on the next page by labeling wires with the time when a given value will be ready (in terms of **T** and **M**).

Maximum delay of all output bits = ____14T + M____

b. Carry select requires two adders for each bit – to justify this extra hardware, how large can M be without the loss of any possible performance advantage? Express in terms of **T**.

M < ____2T_____

Assuming sum bit implemented with a single XOR gate

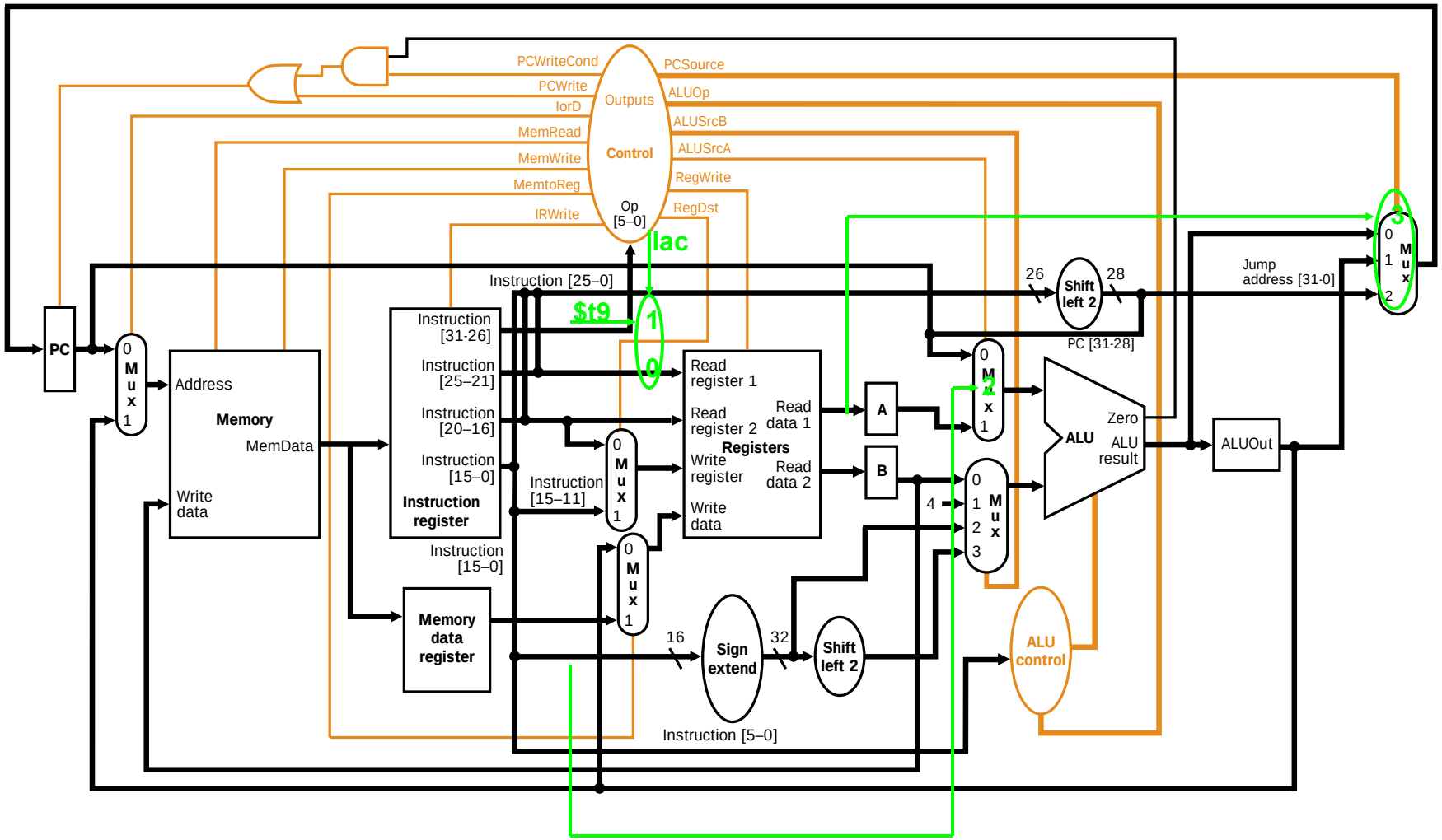

Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
ALU result
PCSrc
LAW
ALU
Zero
ALU result
Address
Read data
Write data
Data memory
16
Sign extend
32
ALU control
M u x

Main Controller

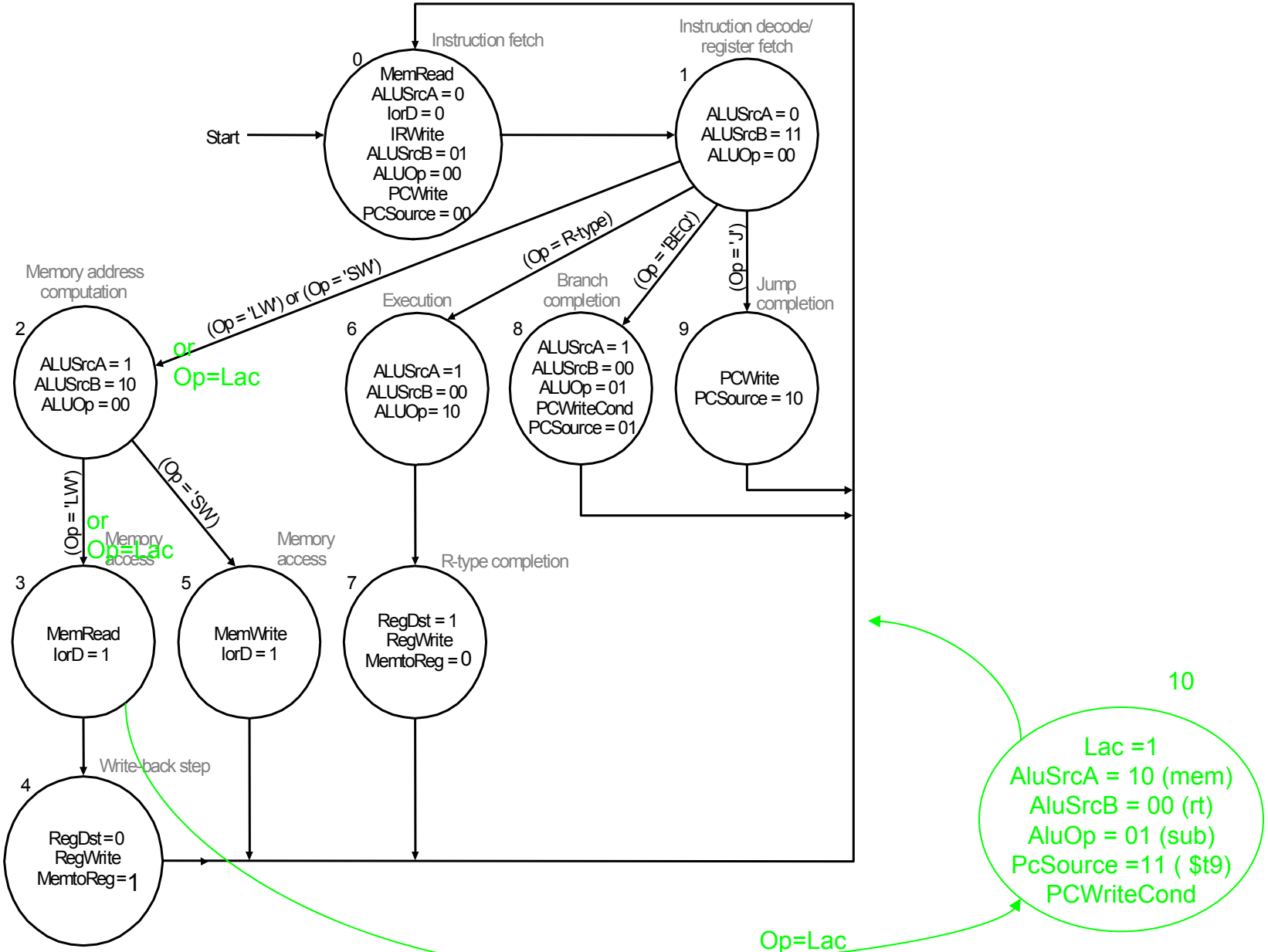
Input or Output	Signal Name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

ALU Controller

Inputs		**ALU Action**	**Output**	
ALUOp	**function**		**ALUCtrl**	LAW
00	XXXXXX	Add	010	0
00	XXXXXX	Add	010	0
01	XXXXXX	Subtract	110	0
10	100000	Add	010	0
10	100010	Subtract	110	0
10	100100	AND	000	0
10	100101	OR	001	0
10	101010	SLT	111	0
10	110000	Add	010	1


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
lac
$t9
Instruction [25–0]
26
Shift left 2
28
Jump address [31-0]
PC [31-28]
PC
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–0]
Instruction register
Instruction [15–11]
Instruction [15–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
ALU
Zero
ALU result
ALUOut
Memory data register
16
Sign extend
32
Shift left 2
ALU control
Instruction [5–0]
Mux

$t9 = 25


Start
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Instruction fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
Instruction decode/
register fetch
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Memory address
computation
or
Op=Lac
6
ALUSrcA =1
ALUSrcB = 00
ALUOp= 10
Execution
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Branch
completion
9
PCWrite
PCSource = 10
Jump
completion
(Op = 'LW')
or
Op=Lac
(Op = 'SW')
3
MemRead
IorD = 1
Memory
access
5
MemWrite
IorD = 1
Memory
access
7
RegDst = 1
RegWrite
MemtoReg = 0
R-type completion
4
RegDst = 0
RegWrite
MemtoReg = 1
Write-back step
10
Lac =1
AluSrcA = 10 (mem)
AluSrcB = 00 (rt)
AluOp = 01 (sub)
PcSource =11 ($t9)
PCWriteCond
Op=Lac