

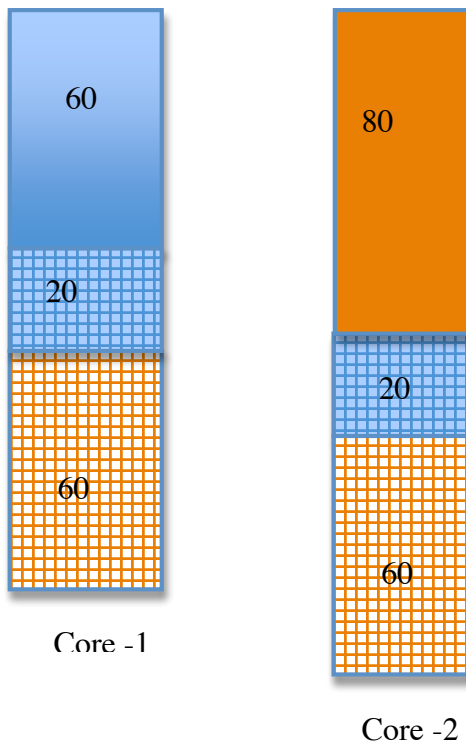
CSCE 4610/5610: Exam 1 Solutions (Krishna Kavi)

1. (15%) Consider that you have purchased a dual core Pentium system. Your computer only runs 2 applications A and B. Application A takes 100 seconds to run on a single core while application B takes 200 seconds to run on a single core.

- a). Suppose you run each application in isolation, and 40% of application A and 60% of application B can be parallelized. What is the speed up achieved when compared to running the applications on a single core.
- b). Suppose you want to run the applications concurrently (that is Application A on one core and Application B on the second core), what speed up is achieved (compared to running the applications on a single core)?
- c). Can you think of any other combinations of running the applications that gives better speed up than the options above?

Key a). For application A, the execution time on 2 cores = $60 + 40/2 = 80$ seconds
 For application B, the execution time on 2 cores = $80 + 120/2 = 140$ seconds
 Total execution time on 2 cores = $80 + 140 = 220$ (instead of 300 seconds on one core)
 Speed up = $300/220 = 1.364$ or 36.4% speedup

b). If we assign application A to one core and application B to the second core, the total execution time when both applications complete is 200. So the speed up = $300/200 = 1.5$ or 50% improvement.



c). The best possible performance (if the applications permit) is given by the time chart shown. The blue portions are for application A while the orange areas are for application B. The solid areas reflect the serial portions of the applications and the hashed areas reflect the concurrent portions.

So the two application complete in 160, and the speed up is $300/160 = 1.875$. Note the maximum possible speed up is 2 with two processors. Of course this is rarely achieved.

2. (35%) In some older architectures an indirect memory address mode is permitted. Consider the following instruction: `LWI Rd, disp`

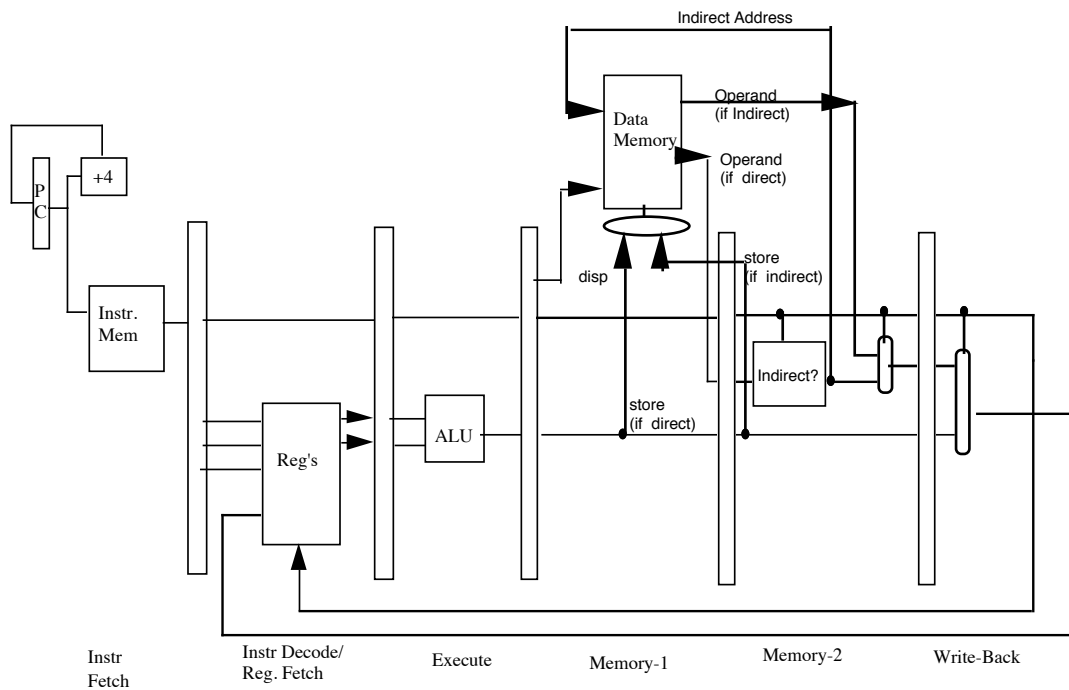
This instruction uses disp as an indirect address. That is, use disp as a memory address, read the contents of memory at that address, and use the value just read as the address of the real operand (that is read memory again and store the value in Rd). Consider the following example: LWI R2, 100

Let us assume that at memory address “100” we have a value of 1500. The instruction will use 100 as an indirect address, obtains the value 1500 stored at memory address 100. The actual address of the operand is 1500. If we have a value of -10 in memory location 1500, then the instruction will load -10 into R2. Note that such indirect address is applicable to both Load and Store.

Describe how we can modify our pipeline design to implement the indirect address mode. Show the pipeline stages and data-paths to indicate which hardware units are accessed in each of the pipeline stage. Also describe in English the functionality of each stage. Indicate the number of read and write ports needed to the data memory to avoid structural hazards.

Key

Since indirect address mode requires that we access the memory twice, we need to design a pipeline with two Memory Access stages. Consider the following diagram.



To simplify, I have eliminated some of the data paths that are needed to handle branch instructions, to use immediate operands in instructions (i.e., sign extension hardware, testing for zero for branch instructions, etc).

The main change is the introduction of two separate memory stages. In Memory-1, we use the displacement from the instruction to fetch a data value. We check to see if the instruction is an

indirect instruction, if so we use the data fetched from memory as an address, and fetch memory again in Memory-2. The Mux in Memory-2 forwards either the data fetched in Memory-1 or in Memory-2 to Write-back. Note that the decision is based on the opcode which is obtained from the pipeline latches.

For store, we may also have direct or indirect addresses. If the opcode is SW (direct), then we store in Memory-1 stage; if the opcode is SWI, we do the store in Memory-2 stage. I hope the diagram is clear with these data paths.

As can be seen we need two read ports and two write ports to the data memory to avoid stalls, since it is possible to have either two LWI in a sequence or a SWI followed by SW instructions in sequence. Consider the following examples

.....
LWI R1, disp1	SWI disp1, R1
LW R2, disp2	SW disp2, R2

The example on the left hand side shows why we need two read ports to data memory (assuming instructions are in separate memory that will be accessed by IF—otherwise we need 3 read ports). The example on the right hand side shows why we need two write ports since the second Store stores in the first MEM –1 stage while SWI stores in the second MEM-2 stage.

Although I did not ask for dependencies, here a bit of discussion on data dependencies.

If we have the following sequence of instructions,

```
LWI R1, disp
ADD R3, R1, R2
```

The ADD will incur two stalls since the operand (R1) for ADD will not be available until LWI passes Memory-2 stage (we can forward the data from MEM2/WB to ID/EX).

Likewise if we have

```
LWI R1, disp
Some Instruction with no dependency on R1
ADD R3, R1, R2
```

The ADD will incur one stall (and we need to forward the data from MEM2/WB to ID/EX)

Note that we could have re-arranged the pipestages (say, IF, ID, Mem-1, Mem-2, EX, WB) to reduce or eliminate stalls due to LW or LWI to an ALU instruction. However, this can cause additional stalls in Branch since value of a register tested by branch may will not be available until EX.

3. (20%). Consider the following code segment

1: ADD	R3, R0, R7
2: LW	R8, 0(R3)
3: ADDI	R3, R3, #4
4: LW	R9, 0(R3)
5: SUB	R1, R8, R9
6: BNEG	R1, Exit

Note R0 is hardwired to Zero.

- List all dependencies (i.e., RAW, WAR, WAW) among these instructions.
- Assuming that we have a delayed branch, is there an instruction that you can move to the delay slot?

Key.

RAW on R3 from 1 to 2.
 RAW on R3 from 1 to 3.
 RAW on R8 from 2 to 5.
 RAW on R3 from 3 to 4.
 RAW on R9 from 4 to 5.
 RAW on R1 from 5 to 6.

WAW on R3 from 1 to 3.
 WAR on R3 from 2 to 3

- We can move #3 ADDI R2, R3, into the delay slot but we need to adjust #4 to LW R9, #4(R3)

The new code will be

1: ADD	R3, R0, R7
2: LW	R8, 0(R3)
3: LW	R9, #4(R3)
4: SUB	R1, R8, R9
5: BNEG	R1, Exit
5: ADDI	R3, R3, #4

4. (30%) Examine the following code segment. Assume following latencies: LD = 2; MULDD = 5; ADDD = 3; Integer operations = 1. We will need one stall on BEQ since this is a taken branch.

Loop: LD	R3, 0(R5)
ADDD	R7, R7, R3
LD	R4, 4(R5)
MULD	R8, R8, R4
ADDD	R10, R7, R8
SD	R10, 0(R5)
BEQ	R10, R11, Loop

- a). Show how many cycles are needed to execute an iteration of the loop
- b). Can you re-order the instructions to improve the number of cycles needed. Show the reordered code.

Key. There are two ways of interpreting the latency. For example we can say that the value loaded by LD R3, 0(R5) is not available for 3 cycles after LD starts executing; this means 3 stalls between LD R3, 0(R5) and ADDD R7, R7, R3. Or we can say that the value is available in the 3rd cycle after LD starts executing; in this case we need two stalls between LD R3, 0(R5) and ADDD R7, R7, R3. I will show both cases here

Cycle	Case 1	Cycle	Case 2
1	LD R3, 0(R5)	1	LD R3, 0(R5)
2	Stall	2	Stall
3	Stall	3	ADDD R7, R7, R3
4	ADDD R7, R7, R3	4	LD R4, 4(R5)
5	LD R4, 4(R5)	6	Stall
6	Stall	7	MULD R8, R8, R4
7	Stall	8	Stall
8	MULD R8, R8, R4	9	Stall
9	Stall	10	Stall
10	Stall	11	Stall
11	Stall	12	ADDD R10, R7, R8
12	Stall	13	Stall
13	Stall	14	Stall
14	ADDD R10, R7, R8	15	SD R10, 0(R5)
15	Stall	16	BEQ R10, R11, Loop
16	Stall	17	Stall
17	Stall		
18	SD R10, 0(R5)		
19	BEQ R10, R11, Loop		
20	Stall		

Thus, one iteration takes either 20 cycles or 17 cycles. If we use branch prediction or delayed branch, we can eliminate the stall after BEQ instruction.

(b). Consider the following reordering of instructions

```

Loop: LD      R4, 4(R5)
      LD      R3, 0(R5)
      MULD   R8, R8, R4
      ADDD   R7, R7, R3
      ADDD   R10, R7, R8
      SD     R10, 0(R5)
      BEQ    R10, R11, Loop

```

Now let us find out how many cycles are needed

Cycle	Case 1	Cycle	Case 2
1	LD R4, 4(R5)	1	LD R4, 4(R5)
2	LD R3, 0(R5)	2	LD R3, 0(R5)
3	Stall	3	MULD R8, R8, R4
4	MULD R8, R8, R4	4	ADDD R7, R7, R3
5	ADDD R7, R7, R3	6	Stall
6	Stall	7	Stall
7	Stall	8	Stall
8	Stall	9	ADDDR10, R7, R8
9	Stall	10	Stall
10	ADDDR10, R7, R8	11	Stall
11	Stall	12	SD R10, 0(R5)
12	Stall	13	BEQ R10, R11, Loop
13	Stall	14	Stall
14	SD R10, 0(R5)		
15	BEQ R10, R11, Loop		
16	Stall		

We saved 4 cycles by reordering the instructions.