CS 2200 Spring 2010 Test 1

Name: Kishore Prism ID:_____

GTID#: 9_____

Problem	Points	Lost	Gained	Running Total	TA
1	1				
2	9				
3	10				
4	10				
5	10				
6	20				
7	15				
8	10				
9	15				
Total	100				

- **You may ask for clarification but you are ultimately responsible for the answer you write on the paper.**
- **Illegible answers are wrong answers.**
- **Please look through the entire test before starting. WE MEAN IT!!!**

Good luck!

1. (1 point, 1 min) (select one)

Slam dunk refers to

- a pagan ritual practiced in Polynesian islands _____
- the act of pouring Gatorade on a coach after winning Superbowl _____
- a weird contest in which grown men leap into the air and try to put a basketball into a hoop __X__
- a metaphor to describe how easy this test is going to be _____
- a metaphor to describe how hard this test is going to be _____

CS 2200 Spring 2010 Test 1

Prism ID:_____

Name:_____ GTID#: 9_____

Processor design

2. (9 points, 10 mins)
(a) (3 points)
Given the following load instruction

```
      LW    Rx, Ry, OFFSET    ;     Rx <- MEM[Ry + OFFSET]
```

Show how to realize a new addressing mode, called *indirect*, for use with the load instruction that is represented in assembly language as:

```
      LW    Rx, @(Ry)   ;  Rx <- MEM[MEM[Ry]]
```

The semantics of this instruction is that the contents of register Ry is the address of a pointer to the memory operand that must be loaded into Rx. You should ensure that there are no other side-effects (i.e., only Rx should be affected by this instruction).

LW Rx, Ry, Ø ; Rx ← MEM[Ry]

LW Rx, Rx, Ø ; Rx ← MEM[Rx]

(b) (6 points)
Given the following instructions:

```
      ADD   Rx, Ry, Rz  ;     Rx <- Ry + Rz
      ADDI  Rx, Ry, Imm ;     Rx <- Ry + Immediate value
      NAND  Rx, Ry, Rz  ;     Rx <-  NOT (Ry AND Rz)
```

Show how you can use the above instructions to achieve the effect of the following instruction:

```
      CLR   Rx          ;     Rx <- 0
```

You can use an extra register to realize this instruction the contents of which can be trashed.

NAND Rz, Rx, Rx ; Rz ← 1's Complement of Rx

ADDI Rz, Rz, 1 ; Now Rz Contains −Rx

ADD Rx, Rx, Rz ; Rx ends up with Ø

CS 2200 Spring 2010 Test 1

Prism ID:_____

Name:_____ GTID#: 9_____

3. (10 points, 10 mins)
Given the software convention for registers:

```
      a0-a2:  parameter passing
      s0-s2:  callee saves if need be
      t0-t2:  caller saves if need be
      v0:     return value
      ra:     return address
      at:     target address
      sp:     stack pointer
```

Recall that JAL instruction of LC-2200 has the following semantics:

```
      JAL at, ra;       ra <- PC_incremented (return address)
                ;       PC <- at (entry point of procedure)
```

The state of the stack is as shown below. To help you out, we have put down the action corresponding to saving **t** registers on the stack. Fill out the actions similarly for the other entries on the stack (who is responsible for the action caller/callee, and what is the action).

Your answer:

Stack Pointer→	Local Variables	Callee allocates any needed space
	Saved **s** Registers	Callee saves if he is going to use them
	Prev Return Address	Caller stores current $ra
	Add'l Return Values	Caller allocates space on stack
	Add'l parameters	Caller places them on stack
	Saved **t** registers	**Caller** saves any t registers whose values it needs upon return

CS 2200 Spring 2010 Test 1

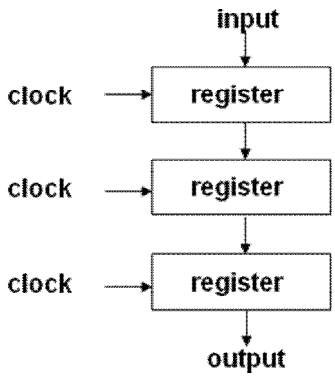
Prism ID:_____

Name:_____ GTID#: 9_____

Datapath and control

4. (10 points, 5 mins) (select one)

(a)



How many clock cycles are needed to get the value at the input to the output in the circuit shown:

_____ 0 cycles

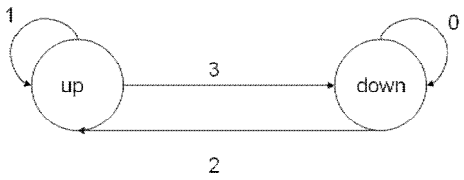
_____ 1 cycles

_____ 2 cycles

__X__ 3 cycles

_____ 4 cycles

(b)



The number of rows in the state transition table for the above FSM is:

_____ 2

_____ 3

__X__ 4

_____ 5

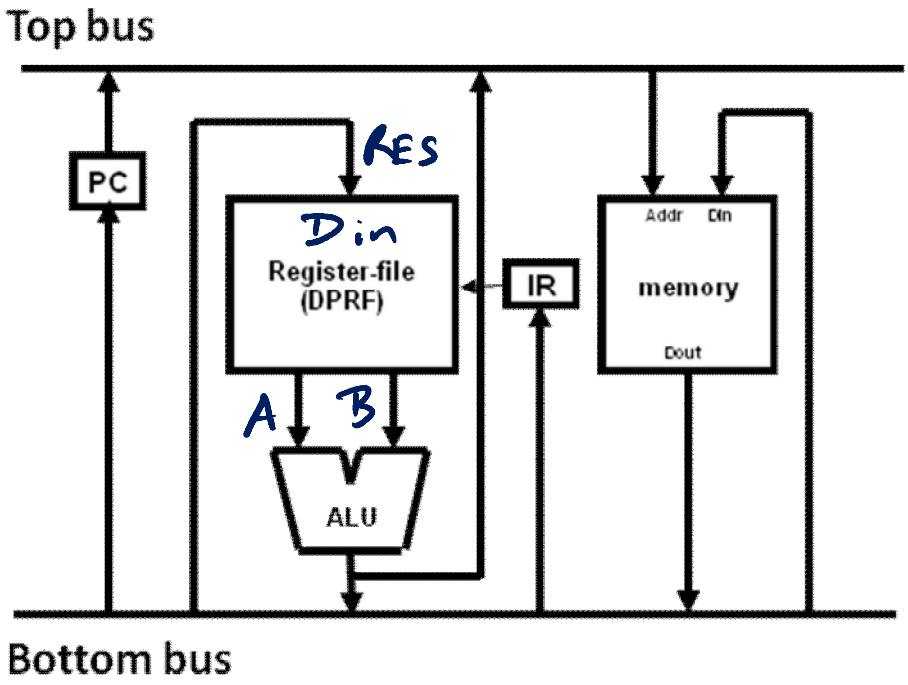
_____ 6

CS 2200 Spring 2010 Test 1

Prism ID:_____

Name:_____ GTID#: 9_____

5. (10 points, 10 min)



Using the above datapath, we wish to perform the following operations:

- IR <- memory[PC]; get contents of memory addressed by PC into IR
- Reg-file[IR 27-24] <- Reg-file[IR 23-20] + Reg-file[IR 3-0]; add values from two registers in the register file and store the result in a third register

Cycle by cycle, show the datapath actions that will accomplish the above operations. Use register transfer format (e.g., ALU -> Bottom bus; Bottom bus -> PC, etc.) to show your work in each cycle.

From IR all register specifiers (2 source addresses and 1 destination address are directly wired into DPRF)

Prism ID:_____

Name:_____ GTID#: 9_____

(additional workspace for 5)

Cycle 1:

PC → Top Bus → Addr input of Memory

Memory [Addr] → Dout → Bottom Bus

Bottom Bus → IR

Cycle 2: (See Figure above)

IR [23-20] → Reg-file address for A

IR [3-0] → Reg-file address for B

A → ALU ; B → ALU

ALU result → Reg file Din

IR [27-24] → Reg-file address for destination register

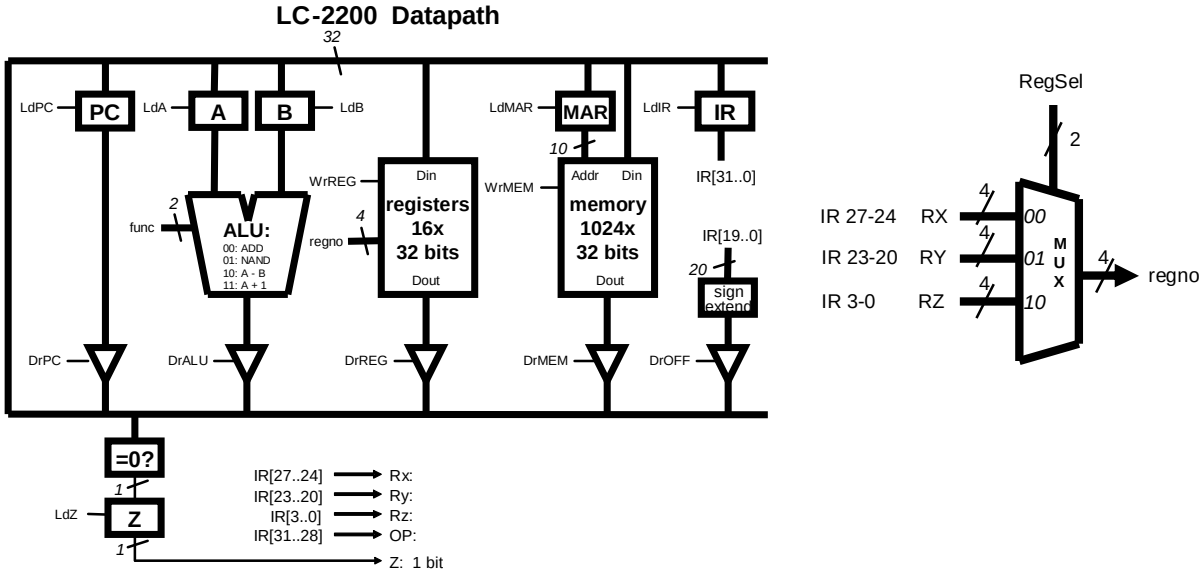
Reg-file Din → Reg-file destination register

CS 2200 Spring 2010 Test 1

Prism ID:_____

Name:_____ GTID#: 9_____

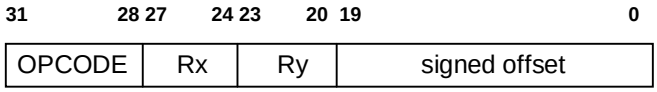
6. (20 points, 15 min)
You are given below a datapath similar to what we have discussed in class.



Given SW instruction as follows:

SW Rx, offset(Ry); MEM[Ry + signed offset] <- Rx;

The instruction format is as shown below:



Write the sequence for implementing the SW (**you don't need to write the fetch sequence for the instruction**). For each microstate, show the datapath action (in register transfer format such as A <- Rx) along with the control signals you need to enable for the datapath action (such as DrREG).

Write your answer on the next page.

Prism ID:_____

Name:_____ GTID#: 9_____

(additional workspace for 6)

SW 1:

Ry → A

control signals needed:

RegSel = 01

Dr REG

Ld A

SW 2:

Sign-extended offset → B

control signals needed:

Dr OFF

Ld B

SW 3:

A+B → MAR

control signals needed:

func = 10

Dr ALU

Ld MAR

SW 4:

Rx → Memory

Control signals needed:

RegSel = 00; Dr REG

Wr Mem = 0

CS 2200 Spring 2010 Test 1

Prism ID:_____

Name:_____ GTID#: 9_____

Interrupts

7. (15 points, 10 mins)
(a)

Save *ko* on stack
Enable interrupt
XXXX
Device code
Restore state
Disable interrupt
Restore *ko* from stack
Return from interrupt

In the above interrupt handler code xxxx is

_____ Get interrupt vector from device

__X__ Save state

_____ Save vector received from device on stack

_____ Acknowledge interrupt

(b) (10 points, 10 mins)
In the following sentence, use the terms “program counter” , “handler address”, “interrupt vector”, “operating system”, “CPU”, “interrupt vector table”, “data bus”, “program discontinuities”, “interrupt acknowledge”, “low memory” to fill in the blanks.

Each device has a unique interrupt vector **assigned by the** operating System**. At boot time, the operating system builds the** interrupt vector table **in** low memory **with the handler addresses for all the known** Program discontinuities**. Upon receiving an** interrupt acknowledge **from the** CPU**, the device puts out its interrupt vector on the** data bus**. The CPU uses the interrupt vector as an index into the interrupt vector table to look up the** handler address **to load into the** Program Counter **for handling the interrupt.**

CS 2200 Spring 2010 Test 1

Prism ID:_____

Name:_____ GTID#: 9_____

Performance

8. (10 points, 10 min)

Machine **Ma**:

- All LC 2200 instructions take on an average 5 clock cycles
- Floating point addition emulation in software takes 30 LC-2200 instructions (i.e., a total of 150 clock cycles)

Machine **Mb**:

- Floating point addition implemented in hardware takes 10 clock cycles
- Clock cycle time of processor goes up by 10% compared to **Ma**

What fraction of the instructions ought to be floating point additions to guarantee at least a 25% improvement in execution time of **Mb** over **Ma**?

Use the following equations in doing this problem:

Let f be fraction of flt. pt. instructions

Execution time = n * CPI_{Avg} * clock cycle time

$$\text{Improvement in execution time} = \frac{\text{old execution time} - \text{new execution time}}{\text{old execution time}}$$

Execution time of Ma

$= [n(1-f) * 5 + nf * 30 * 5] * 1$

$= n(5 + 145f)$

Execution time of Mb

$= [n(1-f) * 5 + nf * 10] * 1.1$

$= n(5.5 + 5.5f)$

plugging into above equation

$$0.25 = \frac{(5 + 145f) - (5.5 + 5.5f)}{5 + 145f}$$

Simplifying

$$f = \frac{7}{413} = 0.01694915$$

CS 2200 Spring 2010 Test 1

Prism ID:_____

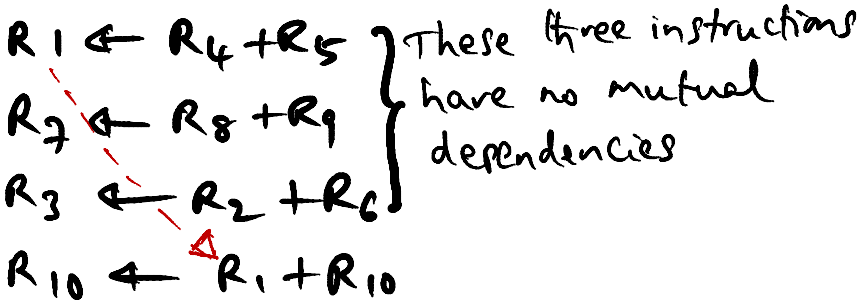
Name:_____ GTID#: 9_____

Pipelining

9. (15 points, 10 mins)(**SHORT ANSWERS WE MEAN IT!!!!**)
(a) Define *Instruction Level Parallelism*, showing an example code to explain your definition.

ILP is the situation wherein, in a sequential program, a set of adjacent instructions in textual order are independent of one another.

Here is an example code where the ILP is 3:



(b) What is the most important consideration in designing the stages of a pipelined processor?

Ensuring that the amount of work in each stage of the pipe is roughly the same so that no single stage will become a bottleneck to performance.

(c) Explain the difference between *latency* and *throughput* in the context of a pipelined processor implementation. What are the metrics used for each?

Latency: The amount of time it takes to execute a single instruction usually expressed as CPI
Throughput: The number of instructions executed per unit time by the processor usually expressed as IPC.

(d) What gives the independence between the stages in the sandwich assembly line that we discussed in class? Identifies the corresponding entities in the instruction pipeline that give the same independence.

Three things:

1. **No resource contention among the stages (each worker has all he needs to do the work for his stage)**
2. **Order form fully specifies the work that a given worker has to do for that specific sandwich**
3. **The partially assembled sandwich on which he has to do additional work**

Corresponding entities in the instruction pipeline

1. **Dedicated hardware resources for each stage of the pipeline**
2. **The pipeline buffer serves the role of the order form and the partially assembled sandwich**

(e) Explain the process by which the size and content of the pipeline registers are decided in the instruction pipeline.

1. **For each instruction, work out the details of the execution in each stage of the pipeline**
2. **For each instruction, work out what partial state has to be passed from one stage to the next**
3. **The union of the partial states thus compiled for each instruction at a given stage helps us to decide the size and content of the pipeline buffer sitting at its output**