# CS 2200 Fall 2012 Test 1

Prism ID:____________

Name:___________________________ GTID#: 9____________

| Problem | Points | Lost | Gained | Running Total | TA |
|---|---|---|---|---|---|
| 1 | 1 (0 min) | | | | |
| 2 | 10 (5 min) | | | | |
| 3 | 10 (5 min) | | | | |
| 4 | 18 (10 min) | | | | |
| 5 | 12 (5 min) | | | | |
| 6 | 7 (5 min) | | | | |
| 7 | 10 (5 min) | | | | |
| 8 | 10 (5 min) | | | | |
| 9 | 22 (10 min) | | | | |
| Total | 100 (50 min) | | | | |
| 10. Bonus Question | 4 | | | | |

- **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, 0 min) (select one, this is a freebie, you get 1 point always)

**NFL Replacement Refs are the reason for**

- Yellow Jacket losing to Miami Hurricanes on Saturday
- Falcons being three and 0 after week 3
- Why I cannot sleep at night
- Why this course is so hard
- Natural disasters
- World unrest
- What is NFL?
- Ask me about soccer

# CS 2200 Fall 2012 Test 1

Prism ID:___________
Name:_________________________ GTID#: 9___________

**Processor design**

2. (10 points, 5 mins)
Given the following instructions

```
      SW    Rx, Ry, OFFSET    ;     MEM[Ry + OFFSET] <- Rx
      ADD   Rx, Ry, Rz        ;     Rx <- Ry + Rz
      ADDI  Rx, Ry, Imm       ;     Rx <- Ry + Immediate value
```

Show how you can realize (i.e., simulate) using the above instructions, a new addressing mode called autodecrement for use with the store instruction that has the following semantics:

```
      SW    Rx, -(Ry)         ;     Ry <- Ry - 1;
                              ;     MEM[Ry] <- Rx;
                              ;
```

Your answer below should show how the SW instruction using autodecrement will be realized with the given instructions.

3. (10 points, 5 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 (for Q3):

| | | |
|---|---|---|
| **Stack Pointer→** | Local Variables | ____________________________ |
| | Saved **s** Registers | ____________________________ |
| | Prev Return Address | ____________________________ |
| | Add'l Return Values | ____________________________ |
| | Add'l parameters | ____________________________ |
| | Saved **t** registers | **Caller** saves any t registers whose values it needs upon return |

# CS 2200 Fall 2012 Test 1

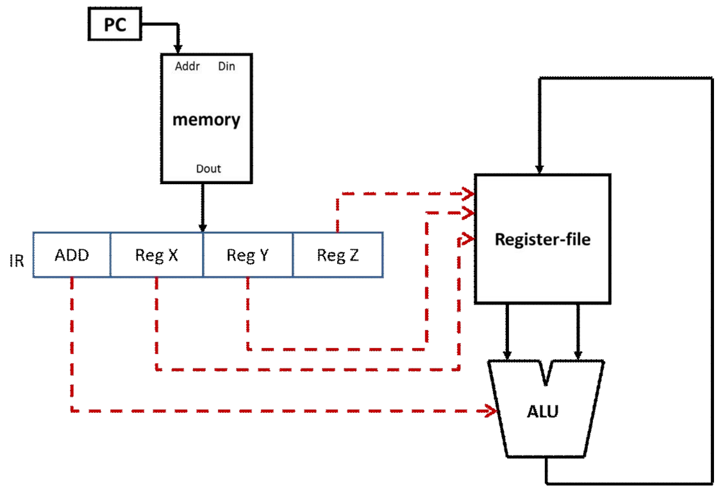
Prism ID:____________

Name:____________________________ GTID#: 9____________

## Datapath and control

4. (10 min)

a) (12 points) In the circuit shown below, both the IR and the Register-file (which is dual-ported) are positive edge-triggered, i.e., data is stored only on the rising edge of the clock. As should be obvious solid lines are for data and dashed lines are for control.



With respect to the above circuit enumerate the number of clock cycles and the action in each clock cycle to perform the following operations: (i) use PC to address memory, (ii) pull the instruction from the memory into IR, (iii) use the register specifiers in IR (Reg Y, and Reg Z) to read two source registers from the Register-file, (iv) supply the register contents to the ALU as source operands, (v) perform addition using the ALU, (vi) use the register specifier in IR (Reg X) as the destination register, and (vi) write the result of the ALU operation into the register file.

b) (6 points) Comparing the Flat ROM based control unit design and the Microsequencer based control unit design:

   (i) which is more space efficient and why?

   (ii) which is more time efficient and why?

**Interrupts**

5. (12 points, 5 mins)

a) (circle the correct choice) An interrupt handler saves and restores the entire register-file since

   (i) it does not know what registers may have been in use at the time of the interrupt in the original program
   (ii) it is a good software engineering practice
   (iii) interrupts could be nested
   (iv) we may never return to the original program from the handler

b) (circle the correct choice) The interrupt vector table holds the starting addresses of the interrupt handlers for all the known sources of program discontinuities during program execution. This table is set up

   (i) by each application when it is started up on your computer
   (ii) by the OS every time a program makes a system call
   (iii) by the OS at boot time (i.e., every time we re-start the OS)
   (iv) preset by the hardware at manufacturing time of the computer

c) (circle the correct choice) We need a “return from interrupt” (RTI) instruction since

   (i) the handler does not know where to return to in the original program
   (ii) it is more intuitive as to what we want to accomplish than a simple jump instruction
   (iii) we need to enable interrupts in addition to returning to the original program
   (iv) we need to store the handler address back in the interrupt vector table in addition to returning to the original program

# CS 2200 Fall 2012 Test 1

Prism ID:____________

Name:______________________________ GTID#: 9____________

d) (circle the correct choice) Upon executing an RTI instruction the processor always goes back to
   (i) user mode
   (ii) system mode
   (iii) mode prior to entering this interrupt handler

e) (circle the correct choice) The difference between an external device interrupt and an internal trap or exception is that
   (i) there is none
   (ii) the vector number for a trap/exception is internally generated by the processor
   (iii) the vector number for a trap/exception is provided by the user in one of the general-purpose registers
   (iv) the vector number that is put out by a given device, changes each time it interrupts

f) (circle the correct choice) We need a “disable interrupt” instruction in the ISA since

   (i) each individual instruction is not atomic
   (ii) the processor often needs to execute a set of instructions atomically
   (iii) the handler needs to execute that instruction before saving the return address in **$k0** on the stack
   (iv) interrupts should be disabled when we are already in an interrupt handler

6. (5 mins)

a) (3 points) Let’s assume that multiple devices simultaneously interrupt the processor. Assume that all the devices are connected to a single interrupt line. Explain (concise bullets please) how the processor selects one of the interrupting devices from among them.

b) (4 points) Give two good reasons why the interrupt handler should use a “system” stack that is distinct from the “user” stack.

**Performance**

7. (10 points, 5 mins)

Given CPI of instruction classes

R-type = 2
I-type = 10
J-type = 3
S-type = 4

Code 1 has:
3 of R-type
3 of I-type
5 of J-type
2 of S-type

Code 2 has:
10 of R-type
1 of I-type
2 of J-type
3 of S-type

For the same program, a compiler generates two possible code sequences as shown above. Which code sequence is better and why? You have to show your work to get ANY credit.

8. (10 points, 5 min)

An architect determines that she can cut the execution time in **half** for **25%** of the instructions by introducing an additional resource in the datapath without affecting the clock cycle time or the execution time of the other instructions. What is the expected **speedup** of the new machine compared to the original?

**Pipelining**

**9.** (10 mins) (**SHORT ANSWERS WE MEAN IT!!!!**)

(a) (3 points) Explain the difference between *latency* and *throughput* in the context of a pipelined processor implementation.

(b) (2 points) (Answer True/False with justification) The primary reason for implementing a pipelined processor is to have a constant latency for each instruction.

(c) (3 points) What is the most important consideration in designing the stages of a pipelined processor?

(d) (2 points) What are the metrics used for latency and throughput as they apply to a pipelined processor?

(e) (2 points) What gives the independence between the stages in the sandwich assembly line that we discussed in class?

(f) (2 points) Identify the corresponding entities (to your answer in (e)) in the instruction pipeline that give the same independence.

(g) (3 points) The FBUF which is at the output of the IF stage of the pipeline contains the instruction that has been fetched from the memory and the PC value corresponding to that instruction. Why is it necessary to have the PC value in the FBUF?

(h) (5 points)

**BEQ Rx, Ry, offset**

has the following semantics:

```
If (Rx == Ry) then PC <- PC + offset
```

Considering only the BEQ instruction, what are the contents of the **DBUF** which is at the output of the ID/RR stage of the pipeline?

**10.** (4 points) **Bonus Question**

a) Structural hazard is due to ____________________ limitation.

b) Code sequence

```
R1 <- R2 + R3
R4 <- R1 + R5
```

Represents a ______________________________________ data hazard

c) Code sequence

```
R2 <- R1 + R3
R1 <- R4 + R5
```

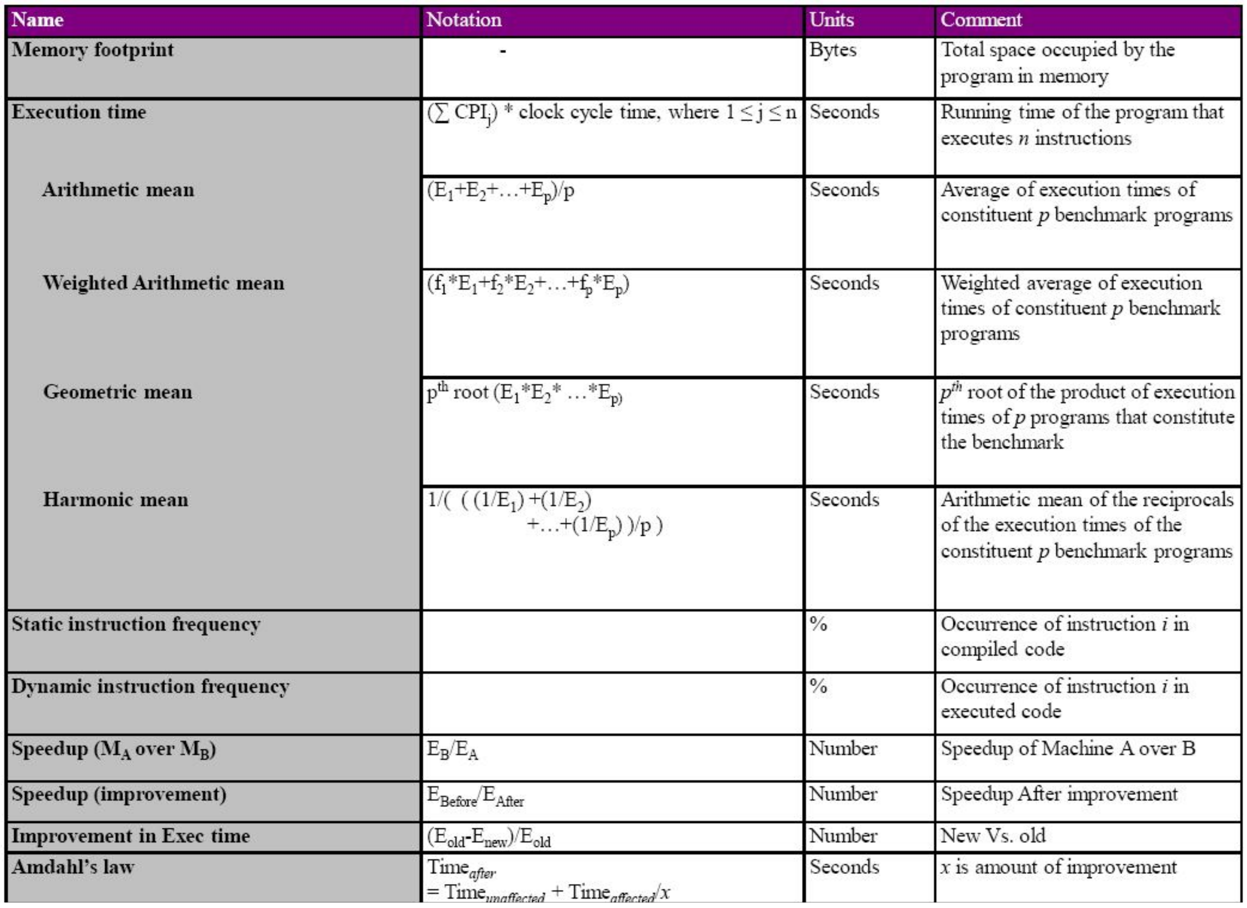
Represents a ______________________________________ data hazard

d) Code sequence

```
R2 <- R1 + R3
R2 <- R4 + R5
```

Represents a ______________________________________ data hazard

Metrics cheat sheet:



| Name | Notation | Units | Comment |
|---|---|---|---|
| **Memory footprint** | - | Bytes | Total space occupied by the program in memory |
| **Execution time** | $(\sum CPI_j)$ * clock cycle time, where $1 \le j \le n$ | Seconds | Running time of the program that executes *n* instructions |
| **Arithmetic mean** | $(E_1+E_2+\ldots+E_p)/p$ | Seconds | Average of execution times of constituent *p* benchmark programs |
| **Weighted Arithmetic mean** | $(f_1{*}E_1+f_2{*}E_2+\ldots+f_p{*}E_p)$ | Seconds | Weighted average of execution times of constituent *p* benchmark programs |
| **Geometric mean** | $p^{th}$ root $(E_1{*}E_2{*}\ldots{*}E_p)$ | Seconds | $p^{th}$ root of the product of execution times of *p* programs that constitute the benchmark |
| **Harmonic mean** | $1/(\ ((1/E_1)+(1/E_2)+\ldots+(1/E_p))/p\ )$ | Seconds | Arithmetic mean of the reciprocals of the execution times of the constituent *p* benchmark programs |
| **Static instruction frequency** | | % | Occurrence of instruction *i* in compiled code |
| **Dynamic instruction frequency** | | % | Occurrence of instruction *i* in executed code |
| **Speedup ($M_A$ over $M_B$)** | $E_B/E_A$ | Number | Speedup of Machine A over B |
| **Speedup (improvement)** | $E_{Before}/E_{After}$ | Number | Speedup After improvement |
| **Improvement in Exec time** | $(E_{old}-E_{new})/E_{old}$ | Number | New Vs. old |
| **Amdahl's law** | $Time_{after} = Time_{unaffected} + Time_{affected}/x$ | Seconds | *x* is amount of improvement |