CS 250 Spring 2017
Purdue University
Practice Midterm Exam 02 SOLUTION

1. The register file is used in two stages of the 5-stage pipelined version of Figure 6.9.
   A **True** Used in both ID for reading and WB for writing.
   B False

2. Which instruction is the most likely to cause a structural hazard in the EX stage of a traditional 5-stage pipeline?
   A load
   B branch
   C sign-magnitude integer compare
   D **floating point divide** A structural hazard in EX can be caused by a long-running operation. This operation is the longest running by far of the options in responses A through E.
   E BCD addition

3. Pipelining is a form of hardware parallelism.
   A **True** Parallelism means doing multiple activities simultaneously. Pipelining overlaps the execution of multiple machine instructions.
   B False

4. **Two versions of this question.** Assume that a five-stage instruction execution pipeline (IF, ID, EX, MEM, WB stages) is capable of hazard detection, but it is not capable of forwarding, so it stalls instead. The register file for this pipeline **can/can not** be written to and then read from during one clock cycle. How many clock cycles will it take to execute the following two instruction assembly language sequence?

```
          sub $5, $7, $8          ; $6 ← $7 – $8
          add $5, $6, $7          ; $5 ← $6 + $7
```

Clock	1	2	3	4	5	6	7	8
sub	IF	ID	EX	MEM	WB			
add		IF	ID	stall	stall	EX	MEM	WB

   A 5
   B 6
   C 7
   D **8 "Can" answer.** See the table above for the timing of each instruction in the pipeline.
   E **None of the above "Cannot" answer.** Without early write and late read of the register file, the ID stage cannot read the result of the sub instruction until after sub completes the WB stage. This places the add ID stage completion in clock cycle 6 above, thus add will finish in clock cycle 9, which is not one of the

choices given.

5. Which of the following lists presents its terms in order from least abstract to most abstract?
   A register, full adder, Boolean logic, S'R' latch. S'R' latch is more concrete, more basic a circuit than a register, which may be built from such latches.
   B "bare metal" computer, silicon wafer, silicon chip, wire. A "bare metal" computer is a more abstract term than a "wire".
   C **transistor, gate, function unit, machine instruction execution.** A transistor can be used to build a gate. Gates can be used to build a function unit. A collection of function units can be used to execute a machine language instruction.
   D multiplexer, decoder, data path, address. A decoder is a component within a multiplexer, and hence less abstract.
   E None of the above is in order from least abstract to most abstract

6. What is the **most/least** complex of the combinatorial circuits in a pipelined processor?
   A The multiplexer in the WB stage. A mux consists of a full set of SOP terms (k inverters and 2^k AND gates to select from 2^k inputs) connected to 2^k switches; simpler than 2^k full adders alone.
   B The largest of the pipeline stage interface registers. Not a combinatorial circuit.
   C **The ALU. Most version.** Contains a circuit for each operation in the instruction set unless sharing happens to be possible; some of these are enormous circuits: shifter, multiplier.
   D **The sign extender. Least version.** Only connects one sign wire to multiple higher order bit positions. Just a wire that fans out to multiple locations. Probably needs a signal boost to provide adequate signal strength, say one AND gates with both inputs tide together to power 5 destination bits from the AND output.
   E No one of the above is always the most complex

7. An 8-byte data item is stored in the byte-addressed memory of a 32-bit computer at address 0x004488cc. To load this item into registers R1 and R2 so that R1 contains the most significant bytes and R2 the least significant bytes, the processor should
   A **Execute two load word instructions, one at address 0x004488cc and one at 0x004488d0.** The 8-byte data item is located at byte addresses 0x004488cc, 0x004488cd, 0x004488ce, … , 0x004488d2, and 0x004488d3. A load word 0x004488cc instruction on this 32-bit (4-byte) computer will load the 4 bytes at addresses 0x004488cc through 0x004488cf. The load word at 0x004488d0 will obtain the other 4 bytes. Which of these two loads it destined for R1 and which for R2 will depend on whether memory is big or little endian, but these two loads are the ones to perform.
   B Execute two load word instructions, one at address 0x004488c8 followed by one at 0x004488cc. Accesses memory locations that, in part, do not contain the 8-byte data item.

- **C** Execute eight load word instructions, one each from addresses 0x004488cc through 0x004488d3. Comprises 8 overlapping loads of 4 bytes each for a total of 32 bytes.
- **D** It is not possible to load this data item into two registers as described
- **E** None of the above is correct

8. Designers are working on a new, pipelined version of the processor with the four instructions shown in text Figure 6.2 and the circuit of Figure 6.9. The have decided to save money and instead of having a dedicated sign extension unit in the ID stage, the ALU in the EX stage will be used to perform sign extension as a dedicated operation. These designers have
   - **A** Increased the speed of the ID stage. Sign extension is really fast, so it is unlikely that it was the limiting factor for ID stage speed.
   - **B** **Created a structural hazard for memory access.** Now the ALU is busy doing sign extend when the load and store memory access instructions need the ALU to be adding the already sign-extended offset to the base address value coming from a register.
   - **C** Forgotten that there is a limit on the number of different circuits (add, subtract, etc.) that may be placed in the ALU. There is no such limit.
   - **D** Created a data hazard when reading offset values. Reading does not create data hazards.
   - **E** None of the above is correct

9. What happens to instruction(s) in a pipeline that were fetched earlier than an instruction that is stalled?
   - **A** Those instruction(s) stall, also. If so, then the entire pipeline is stalled. Computer operation ceases. Not desirable.
   - **B** **Those instruction(s) do not stall.** Fetched earlier means that these are the instructions that are further along in the pipeline than the stalled instruction. These instructions must be allowed to complete because the action that can release the stalled instruction to continue in the pipeline will be performed by one of these instructions.
   - **C** Those instruction(s) do not stall if they have no dependence with the stalled instruction. These instructions may well have a dependence with the stalled instruction(s), but their completion is what respects the dependence.
   - **D** Those instruction(s) stall if they have a true data dependence with the stalled instruction. Same reason as answer C.
   - **E** Those instruction(s) are replaced with NOP instruction(s). Doing so would remove a portion of the program from the execution carried out by the pipeline rendering the pipeline results incorrect.

10. Consider the following three-instruction assembly code sequence.

```
store r5, r7, 4       ; memory[r7 + 4] ← r5
add r7, r7, r8        ; r7 ← r7 + r8
load r3, r6, –16      ; r3 ← memory[r6 – 16]
```

In addition the instruction schedule shown above, *how many more* valid schedules exist for this instruction sequence?

A 0
B **1** The other schedule is store, load, add.
C 2
D 3! = 3 x 2 x 1
E None of the above

The dependences in the given code are the following:

```
store r5, r7, 4        ; anti-dependence via r7 from store to add
add r7, r7, r8         ;
load r3, r6, –16       ; possible true data dependence between store memory
                       ; location r7+4 and load memory location r6-16
                       ; we cannot rule out that r7+4 = r6-16
```

So the scheduling constraints are that the load must remain after the store and the add must remain after the store. So we can validly schedule the order store, load, add but no other. Thus, there is one additional valid schedule.

11. Consider this code.

```
for (i=1000; i>=1; i=i-1) {
        x[i] = x[i+1] + s;
}
```

Can this loop be unrolled 4 times and, after performing or not performing the loop unrolling, how many basic blocks will the code have?

A Cannot be unrolled 4 times; will have 1 basic block
B Cannot be unrolled 4 times; will have 1000 basic blocks
C **Can be unrolled 4 times; will have 1 basic block.** Any fixed iteration loop can be unrolled. In the limit you can write your code out in full and eliminate the need for a loop. Loop bodies of this type have no exit from within the loop body, so the loop body is a single basic block. Unrolling only extends the length of the basic block to match the length of the unrolled loop body. So, the unrolled loop still have one basic block in the loop body.
D Can be unrolled 4 times; will have 4 basic blocks
E Can be unrolled 4 times; will have 250 basic blocks

12. Which instruction causes a control hazard in a pipelined processor?

A load R1, R2, 16
B add R1, R1, R3
C store R1, R2, 32
D subi R3, R3, 8
E **bne R2, R3, target** Control hazards are caused by control instructions. bne is the only control instruction among those presented in the answer options.

13. Loop unrolling

A Increases the number of instructions in instruction memory

B Reduces the number of instructions fetched
C Increases basic block size
D **All of the above** I hoped that this question might give something of an AHA moment as you considered items A, B, and C and perhaps saw that each of them is true, which at first, seems contradictory.
E None of the above

14. Refer to the following code the answer the next three questions.

```
        loadi  r5, 16         ; r5 ← 16  (load immediate value into register)
loop:   load   r1, r5, 40     ; r1 ← memory[r5 + 40]
        add    r1, r1, r2     ; r1 ← r1 + r2
        addi   r5, r5, -4     ; r5 ← r5 – 4
        store  r1, r5, 40     ; memory[r5 + 40] ← r1
        bne    r5, 0, loop    ; branch to loop if r5 != 0  (r5 not equal to zero)
```

If the loop is unrolled 2 times, the number of instructions in the resulting loop body
A must be 10
B **need only be 8** Loop overhead instructions, those instructions that exist only to manage decision making about whether to perform another loop iteration or not, can and should be eliminated from all but the final iteration in the unrolled loop body. For this loop there are five instructions in the loop body (load, add, addi, store, and bne) of which two are overhead instructions (addi and bne). Eliminating one pair of overhead reduces the number of instructions in the 2x unrolled loop body from 10 to 8.
C need only be 6
D Cannot be determined until the instructions are scheduled
E None of the above

15. As the code above executes, there is a true data dependence from the store to the load.
A **True** The store to address r5,40 at the end of one loop iteration is followed by a load from address r5,40 in the following iteration. So information is flowing from the store instruction to the load instruction: a true data dependence. This true data dependence also suggests that we re-write the code to eliminate the load.
B False

16. Which of the following choices for predicting bne in the above code is better than the others?
A **Taken** Predict taken will be correct on every loop iteration except the final one.
B Not taken. Will be wrong on every loop iteration except the final one.
C From the 1-bit branching history of bne. Will not predict on the first loop iteration because there will be no history yet. Will predict correctly on every subsequent iteration except the final one.
D **No one of the above choices is better than the other two.** Because option A and C are so close, option D was also accepted.

E Unfortunately, prediction cannot be used with this branch instruction. Prediction can always be used. Whether the payoff is positive or negative depends on the nature of the code and the data being processed.

Part II: Short Answer Questions

Legibly store your answers to these questions on this exam document. Keep your answers short and to the point. Value for each question shown in [x].

1. A 64-bit RISC computer has byte-addressed memory. At what address should we expect to find the default next instruction? __PC + 8__________________

2. How many different bit strings correspond to machine language **ADD/LOAD** instructions
   for the format of text Figure 6.2? _2^{27}__. A figure 6.2 the add and load instructions have a fixed 5-bit opcode, and after that all bits are free to be specified. These bits are are regA, regB, dst reg, and unused offset, for a total of 4+4+4+15 = 27 bits that may be chosen, so there are 2^{27} different load instruction bit strings.

3. Branch prediction has a positive __payoff__ when correct, and likely a negative one when incorrect.

4. Register renaming can remove __name dependence__ from assembly language code.

5. A zero-address computer executes a subtract instruction:
   Difference = Subtrahend – Minuend
   The Minuend value is sourced from the __top or next to top of stack_________ and

   the Difference destination is __top of stack_________ . (Be specific for full credit.)

6. Computer Y is a 3-address architecture for which operands and results reside in memory. Computer Z is a 3-address computer for which operands and results reside in the register file. What is the most significant difference between the ADD instructions for computers Y and Z?
   Answer: This question was insufficiently constrained to produce any focus to the answers. All responses earned 3 points. Left blank earned zero points.
   My intent was to focus on the size of the instruction words for computer Y versus Z.

7. Two assembly language instructions are defined as follows.
   ```
   jump offset(register)   ; PC ← offset + contents of register
   brr regA, regB, offset  ; if contents of regA = contents of regB, then
                           ;    PC ← PC + offset else  PC ← PC + 4
   ```
   Which instruction, jump or brr, is easier to use when writing program loops and why?
   Answer: brr because target = PC + offset so the offset is just based on the instruction count in the assembly listing from brr to its target instruction.

8. What, if any, constrains are there on the execution order of the following assembly language instructions?

label01:	add R1, R3, R7	# R1 = R3 + R7
label02:	sub R4, R4, R6	
label03:	add R9, R1, R2	
label04:	shift R12, 5	

Answer: There is a data dependence from the first add instruction to the second add instruction, via register R1, and no other dependences.

9. Consider the processor circuit below and the instruction lw defined as follows.

lw r4, r3 ; r4 ← memory[r3]

Show how to modify Figure 6.9 to perform the lw ~~instruction~~ memory access during the EX stage. Clearly draw in any necessary new data path(s) and clearly mark with an "x" any data paths that must be removed.

Answer: See the solution to Homework 07, Question 6.

10. Consider pipelining the circuit of text Figure 6.9 into five stages: IF, ID, EX, MEM, and WB. On this figure, use a thin line, rather than a thin rectangle, to show the location of the ID/EX pipeline register. Your line must cut through all the signal paths shown in Figure 6.9 that connect to this register. You may ignore any signals not shown in 6.9 that would be part of an actual ID/EX register.

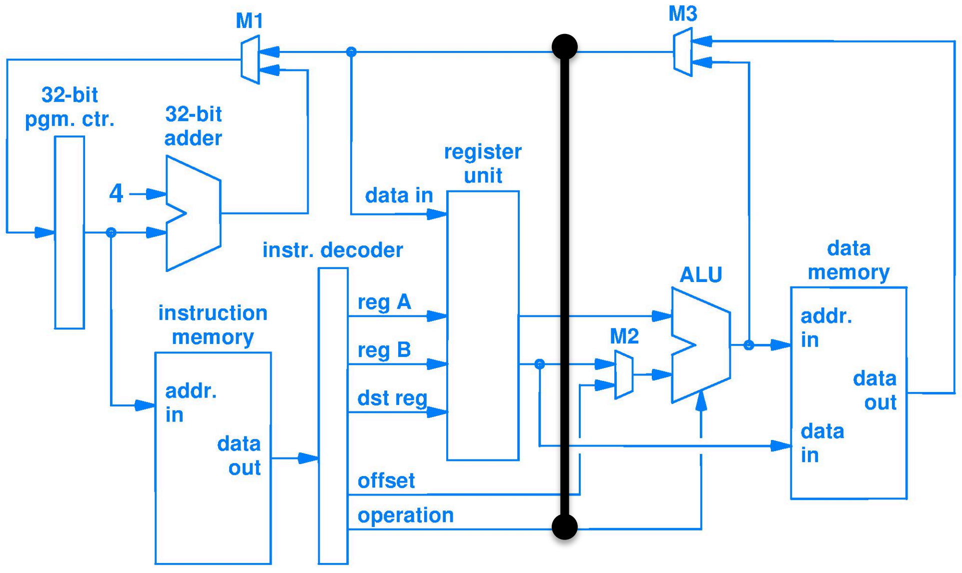

Figure 6.9 Illustration of data paths including data memory.

17. Refer to Figure 6.9 for the following set of questions (a) through (g). If asked to specify parts of Figure 6.9 please use *exactly* the same names as shown in the figure. For example, write **32-bit pgm. ctr.** or **M2** if one of these is a component you wish to name; write **dst reg** if that is a signal path you wish to name.

A With respect to execution of the four instructions shown in Figures 6.1 and 6.2, list all the components in Figure 6.9 that output pointers only.
Answer: 32-bit adder, M1

B With respect to execution of the four instructions shown in Figures 6.1 and 6.2, list all the components in Figure 6.9 that sometimes output a pointer and sometimes output a value.
Answer: ALU, M3.

C After an **add** instruction the value of **32-bit pgm. ctr.** will be updated to what new value?
Answer: New 32-bit pgm.ctr. value will be the location in memory of the **add** instruction plus 4.

D The instruction format of Figure 6.2 can use addressing mode 3 in Figure 7.6 with a memory up to what maximum size?
Answer: 2^{15} locations because the offset field is 15 bits.