
CS230: Practice Problem Set 1 (Autumn 2024)

Date posted: August 13, 2024

These are ungraded practice questions. You are strongly encouraged to solve these independently to ensure you understand the content taught in class.

- Given $X = 0001001111110000$ and $Y = 0000110001011111$. Calculate $Z = X - Y$ using 2's complement subtraction.
 - For example, $Z = 0000110011001100$, the positions of set bits (LSB is position 0) in Z will be the minterms. So, in this case $Z = \sum m(2, 3, 6, 7, 10, 11)$. Minimize the computed Z from (a) using a K-map.
- Implement the below expression using a single 8 x 1 MUX.
 $F(A, B, C, D) = \sum m(1, 3, 4, 11, 12, 13, 14, 15)$.
- By now, you must have figured out that NAND and NOR gates are universal gates (any Boolean logic circuit can be constructed using only NAND gates or NOR gates). Find out the minimum number of 2-input NAND gates or NOR gates (find both the counts) required to implement:
 - Half Adder
 - Full Adder
 - Full Subtractor
- Let m and n be the number of input lines and output lines, respectively, for a decoder that is used to uniquely address a byte-addressable 1 GB RAM, then what is the minimum value of $m+n$?
- Design a four-input priority encoder with the help of the table given below. D0 has the **highest** priority, and D3 has the **lowest** priority.

Inputs				Outputs		
D3	D2	D1	D0	A	B	C
0	0	0	0	X	X	0
X	X	X	1	0	0	1
X	X	1	0	0	1	1
X	1	0	0	1	0	1
1	0	0	0	1	1	1

Table 1: Truth Table for Priority Encoder

- How many 3-to-8 line decoders with an enable input are needed to construct a 6-to-64 line decoder without using any other logic gates?

7. Assume a full adder has a worst-case delay of 15 ns for *sum* output and 12 ns for *carry* output. If such adders are used to design a 16-bit ripple carry adder, then compute the worst-case delay of the circuit.
8. If instead of a ripple carry adder design for a 16-bit adder, we use a cascade of four 4-bit carry-lookahead adders, each of which has a worst-case delay of 25 ns for its *carry* output and worst-case delay of 30 ns for its *sum* outputs, then find the worst-case delay of such a 16-bit adder.
9. Let $f(w, x, y, z) = (w'x'y'z' + w'x'y'z + w'xyz' + w'xyz + wx'y'z' + wx'yz)$ be the given function of 4 variables w, x, y, z .
Suppose only a single 4 x 1 MUX and a single 2-input XOR gate is available. Although inverters are available in plenty. Implement the above logic function using signals w, x to drive the select inputs s_1, s_0 of the multiplexer and single XOR gate and a **minimum** number of inverters.
Express your solution uniquely by describing the logic expressions for the 4 data inputs d_0, d_1, d_2, d_3 of the multiplexer. Note that these expressions should adhere to the restriction of using a **single XOR** gate and a **minimum number of inverters**.
10. Using a 4 x 1 MUX with select inputs S_1, S_0 as A, B , implement the following function: $F(A, B, C, D) = \sum m(3, 4, 6, 9, 10, 11, 13, 14)$.
Draw the circuit. Assume complements of inputs are available. In addition, you may use NOT, OR, AND or XOR gates. An **inefficient** implementation (unnecessary use of gates) will **not** receive full credits.
11. What is the value of xyz if the following expression is solved?
 $(11)_2 + (22)_3 + (33)_4 + (44)_5 = (xyz)_6$. a_b denotes a represented with base b .
12. You have to design a two-bit less than comparator logic circuit ($A < B$). The following incomplete circuit is provided to you. Name the appropriate logic gate for each box to complete the circuit.

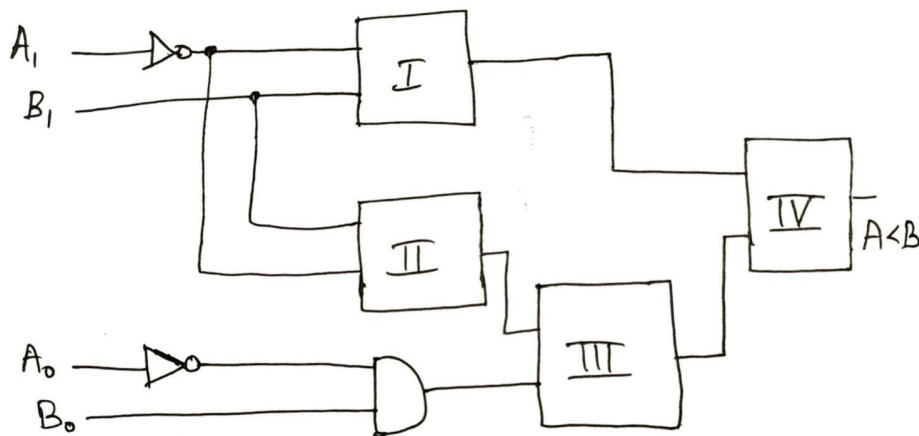


Figure 1: Incomplete circuit for Q.12

13. Minimize and realise the function $f = \overline{\overline{\overline{A + [B + \overline{C} \cdot (\overline{A \cdot B + A \cdot \overline{C}})]}}}$ using minimum number of 2-input NOR gates.
 14. Find dual and complement of the function $F = \overline{X \cdot Y \cdot Z + \overline{X} \cdot \overline{Y}} + Y \cdot Z$
 15. What is the minimum number of 2-input NOR gates required to implement a 4-variable function represented in the form of minterms as, $f = \sum m(0, 2, 5, 7, 8, 10, 13, 15)$? Draw the circuit.
 16. Give the map of an irreducible four-variable function whose sum-of-products represents 2^3 minterms. Prove that there exists a function of n variables whose minimal sum-of-products form consists of 2^{n-1} minterms and that no function, when expressed in sum-of-products form, requires more than 2^{n-1} product terms. Derive a bound on the number of literals needed to express any n -variable function.
 17. For the three functions shown below, obtain a multi-output minimized two-level implementation using an augmented prime implicant chart. Assume that minimizing the total number of gates is the sole objective.
 - (a) $f1 = \sum(2, 3)$
 - (b) $f2 = \sum(2, 3, 4, 5, 6, 7)$
 - (c) $f3 = \sum(1, 3, 5, 7)$
-

Best wishes!

CS230: Practice Problem Set 2 (Autumn 2024)

Date posted: August 14, 2024

These are ungraded practice questions. You are strongly encouraged to solve these independently to ensure you understand the content taught in class.

1. Consider a sequential digital circuit consisting of T flip-flops and D flip-flops as shown in the figure. CLKIN is the clock input to the circuit. At the beginning, Q_1 , Q_2 , and Q_3 have values 0, 1, and 1, respectively.

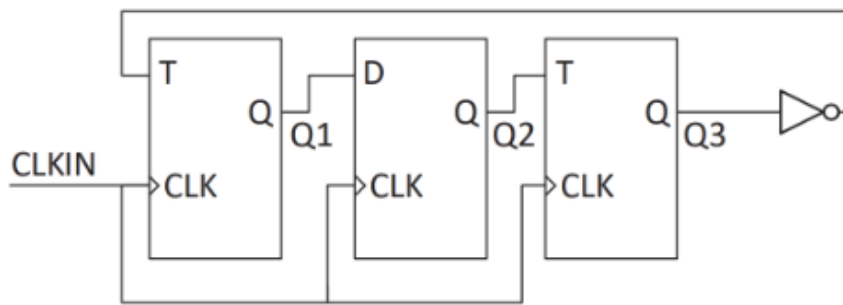


Figure 1: Sequential circuit for Q.18

Analyze the circuit and determine the possible sequences of states for Q_1 , Q_2 , and Q_3 . Determine the sequence of state of Q_1 , Q_2 , and Q_3 that can never occur.

2. Consider the circuit given below with initial state $Q_0 = 1$, $Q_1 = Q_2 = 0$. The state of the circuit is given by the value $4Q_2 + 2Q_1 + Q_0$. Note that this represents the decimal value corresponding to a state. For example, $Q_2 = 1$, $Q_1 = Q_0 = 0$ represents decimal value 4. Find the correct state sequence of the circuit.

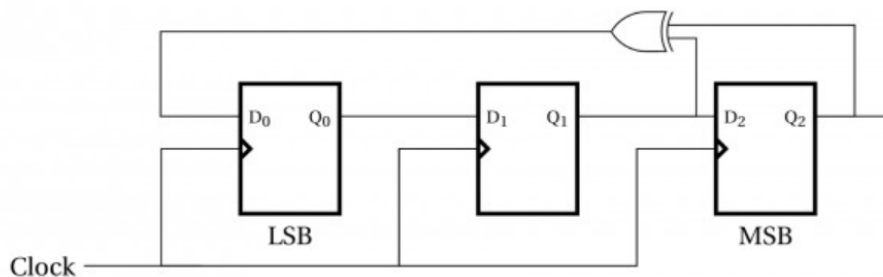


Figure 2: Sequential circuit for Q.2

3. Design a Mealy state machine that detects a sequence '110' in a stream of input bits. The machine should output '1' on the clock cycle following the detection of the sequence and '0' otherwise. Your design should minimize the number of states and transitions.
 - (a) Draw the state diagram.
 - (b) Derive the state transition table.
 - (c) Implement the circuit using D flip-flops and logic gates.
4. A sequential circuit with two D flip-flops A and B , two inputs x and y , and one output z is specified by the following next-state and output equations:

$$A(t+1) = x'y + xB$$

$$B(t+1) = x'A + xB$$

$$z = A$$
 - (a) Draw the logic diagram of the circuit.
 - (b) List the state table for the sequential circuit.
 - (c) Draw the corresponding state diagram.
5. Design a sequential circuit with two JK flip-flops A and B and two inputs E and F . The circuit should operate according to the following rules:
 - (a) If $E = 0$, the circuit remains in the same state regardless of the value of F .
 - (b) When $E = 1$ and $F = 1$, the circuit should go through the state transitions $00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ and repeat.
 - (c) When $E = 1$ and $F = 0$, the circuit should go through the state transitions $00 \rightarrow 11 \rightarrow 10 \rightarrow 01 \rightarrow 00$ and repeat.
6. A Door opens if it ever sees the password 101 in a transmission. More formally, this FSM takes a bitstring consisting of 0's and 1's as its input and continually outputs 0's until it sees the substring 101, after which it outputs 1's continuously. Example execution of the FSM Input: 0001000101000101 Output:0000000001111111.

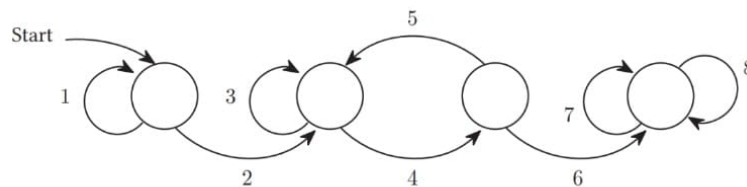


Figure 3: FSM for Q.6

- (a) Mark all the arrows numbered from 1 to 8 with appropriate transitions.

- (b) Modify the above FSM to open the door when it detects the string "0110" in the transition.
 - (c) Extend your FSM to recognize two different patterns: "100" and "010". the FSM should output a continuous 1's after detecting either pattern.
7. Design a sequence detector that identifies the pattern 1101 in any given input, such as 011010110100011 and also detects the pattern 1010. The detector should output a high signal (1) for one clock cycle when either of the sequences is detected. The detector should be able to detect overlapping sequences and sequences that start in the middle of the input stream.
- (a) Draw a state diagram for the sequence detector that detects both patterns 1101 and 1010.
 - (b) Decide whether your designed FSM should be a Mealy and/or Moore machine and provide justification for your design choice.
 - (c) Draw the state diagram of your chosen machine (Mealy/Moore).
 - (d) Draw the state table for your sequence detector. (also check for input stream '0110101101000110001010101101')
8. Design a synchronous counter that counts from 0 to 255 and then wraps around to 0, using only 8 flip-flops and a minimal number of gates.
- (a) Use only 8 flip-flops (D-type or T-type)
 - (b) Use a minimal number of gates (AND, OR, NOT, etc.)
 - (c) The counter should be synchronous, meaning that all flip-flops are clocked simultaneously
 - (d) The counter should count from 0 to 255 and then wrap around to 0
 - (e) The counter should have a single clock input and a single reset input

Hint: You may need to combine binary and gray code to achieve the desired count sequence.

9. Consider the following finite state machine (FSM) table with six states. Minimize the number of states in the FSM using state equivalence while preserving the machine's original behaviour.

Table 1: Table for Q.9

Current State	Input	Next State	Output
A	0	B	0
A	1	C	1
B	0	A	0
B	1	D	1
C	0	E	0
C	1	F	1
D	0	B	0
D	1	C	1
E	0	A	0
E	1	D	1
F	0	C	0
F	1	E	1

- (a) The minimized FSM should have the same input alphabet and output alphabet as the original FSM.
- (b) The minimized FSM should produce the same output sequence for any given input sequence as the original FSM

Hint: You may need to use the concept of state equivalence

Best wishes!

CS230: Digital Logic Design and Computer Architecture

Tutorial 01 [Mon 19 Aug, Tue 20 Aug, Thu 22 Aug]

Concepts tested: Introduction to Computer Architecture, Instruction Set Architecture

1. What is a combinational circuit? What is a combinatorial circuit? What is a sequential circuit?
2. Name a C compiler you have used. In this case, which program was the assembler?
3. The language hierarchy needs some enhancement in the case of Java. Draw the enhanced language hierarchy.
4. Name an obsolete storage device.
5. What is the input device for the “Cell Phone Motor Starter” product described here:
<https://www.indiamart.com/mobitech-wireless/agriculture-automations.html>
6. How much faster would an octa-core smart-phone be compared to a dual-core smart-phone?
7. Consider the following definition of a C structure.

```
struct props {
    int x; // size 32 bits
    long int a; // size 64 bits
    char *y; // size 32 bits
};
```

Give this structure definition, translate the following C code into MIPS assembly code.

```
int len = 100;
struct props M[100];
int i;
struct props *ptr;
for(i = 0; i < len; i++) {
    ptr = &M[i];
    ptr->x = 65539;
    ptr->a = 281487861612544L; // 65539 * 65536 * 65536
    ptr->y = i+3;
} // End for()
```

8. [Based on Q2.14 from the text] Translate the following C code segment to MIPS assembly language:

```
while (save[i] == k) { i = i + 1; }
```

- (a) First, using a conditional branch at the top of the loop and one unconditional branch at the bottom of the loop.
 - (b) Next, an equivalent code using only one conditional branch per loop execution.
 - (c) What is the static code size in each case?
 - (d) How many instructions are executed in each case, if the number of iterations of the loop is 10 (i.e., $\text{save}[i + 10 * j]$ does not equal k but $\text{save}[i], \dots, \text{save}[i + 9 * j]$ equal k)?
9. What does the acronym MIPS (the processor) stand for?
 10. In which classes of computing platforms is MIPS popular today?

CS305: Computer Architecture

Tutorial 03, [Thu 27 Jul, Mon 31 Jul, Tue 01 Aug]

Concepts tested: Instruction Encoding, Function Call Support, HLL Code to Process

1. An assembly program has three functions as outlined below.

main:	F:	G:
...
# read \$a0, \$a1	# read \$a0, \$a1	# set \$s3
...	# set \$s0, \$s1	...
# set \$s0, \$a0, \$a1	# set \$t0, \$t1	# set \$t1
# set \$s1, \$t0	...	# set \$ra (unusual)
...	jal G	...
jal F	...	# read \$s3, \$t1
# read \$v0, \$s0, \$a0	# read \$t1	...
...	# read \$s0, \$s1, \$a0	jr \$ra
jr \$ra	...	
	jr \$ra	

- (a) Which registers does main have to save as caller? As callee?
 - (b) Which registers does F have to save as caller? As callee?
 - (c) Which registers does G have to save as caller? As callee?
2. If the number of registers in MIPS is increased to 64, what implication does it have on the instruction encoding?
 3. What is the maximum array index which can be supported as a constant in a single load instruction? Assume that the array is of 32-bit integers.
 4. Suppose that program P is written in 2 files $p1.s$ & $p2.s$. It has no other external library. And program Q is written in 2 files $q1.s$ and $q2.s$. Q has to be linked with an external library $lib1.o$ before being executed. Answer the following questions.
 - (a) While generating the object files $p1.o$ & $p2.o$ for P , can the assembler exchange every instance of $\$s0$ with $\$s1$ (i.e. use $\$s1$ wherever $\$s0$ appears, and vice versa)? What about while generating the object files $q1.o$ & $q2.o$? You can assume (for both P and Q) that no unresolved instruction (i.e. in the relocation table) uses the two registers in question. Explain your answer briefly.
 - (b) Answer the above question for the case when the two registers being exchanged are $\$s0$ and $\$t0$. Explain your answer briefly.

CS230: Digital Logic Design and Computer Architecture

Tutorial 03 [Mon 02 Sep, Tue 03 Sep, Thu 05 Sep]

Concepts tested: Computer Arithmetic in MIPS, Computer Performance Quantification, Single Cycle Implementation

1. Consider the following Java code fragment:

```
for(i = k; i < N; i++) { a[i] = b[i] + c; }
```

Assume that 'a' and 'b' are arrays of words. Also assume that the base address of 'a' and 'b' are in \$a0 and \$a1 respectively. Register \$t0 is associated with 'i', \$t1 with 'k', \$t2 with 'N', and register \$s0 with the value of 'c'. Assume that the (maximum allocated) lengths of arrays 'a' and 'b' are in \$a2 and \$a3 respectively.

- (a) Translate the above code into MIPS assembly code. Comment your code appropriately, and use intuitive label names. You have to check for memory out-of-bounds exception before each array reference. If there is any out-of-bounds exception, you should break out of the loop by jumping to a label called 'OutOfBounds'. Write the code without any optimization taking advantage of MIPS's 2's complement representation.
- (b) What optimization can you make in the above code to take advantage of 2's complement representation, while doing the out-of-bounds check?
- (c) Suppose you want to further optimize the above code by extending the MIPS ISA. To the ISA, we want to add instructions `lw_inc` and `sw_inc`, which increment something in addition to loading/storing. What is this something which must be incremented by the instruction? And what should be the default increment value?
- (d) Which of `sw_inc` and/or `lw_inc` can you use in the above code to optimize it? How?

2. MIPS code performance analysis

For this question, take the code segment from the previous question. And assume that the 2's complement-based optimization in part (b) of the above question always applies.

- (a) How many instructions does the optimized code from (b) execute, assuming that there are no out-of-bounds exception? Give your answer in terms of appropriate symbols, as necessary.
- (b) What is the new instruction execution count after using `sw_inc` and/or `lw_inc` in (d)?
- (c) Suppose that the addition of `sw_inc`/`lw_inc` to the ISA results in an factor of 'x' increase in the clock period. For what values of 'x' does the use of `sw_inc`/`lw_inc` improve the performance of the above code segment? For what values of 'x' does it degrade the performance?

3. **[Based on Q4.11 from the text]** Consider program P which runs on a 1 Ghz machine M in 10 sec. An optimization is made to P, replacing all instances of multiplying a value by 4 (`mult X, X, 4`) with two instructions that set X to X+X twice (`add X, X, X; add X, X, X`). Call this optimized program P1. The CPI of a multiply instruction is 4, and the CPI of an add is 1. After recompiling, the program now runs in 9 sec on M.

- (a) How many executed multiplies were replaced by the compiler?
- (b) What is the minimum possible increase in static code size when going from P to P1?

4. The following is a C++ implementation of a method for computing the square root of a given number (assumed positive).

```
float sqrt(float x) {  
    float est = 1;  
    while ((est*est) != x) { est = 0.5*(est + x/est); }  
    return est;  
}
```

- (a) What is wrong with the above code and how can you correct it?
- (b) What is the name (in Mathematics) of the algorithm logically implemented by the given algorithm? (Up to 3HP for the first three who identify the name and tell the instructor during the tutorial).

5. Draw the hardware diagram for the single cycle implementation of the MIPS instruction subset `add`, `sub`, `and`, `or`, `slt`, `lw`, `sw`, `beq`.

CS230: Digital Logic Design and Computer Architecture

Tutorial 04 [Mon 09 Sep, Tue 10 Sep]

Concepts tested: Single Cycle Implementation, Extension

1. Draw the hardware diagram for the single cycle implementation of the MIPS instruction subset **add**, **sub**, **and**, **or**, **slt**, **lw**, **sw**, **beq**.
2. Identify the control lines in the above implementation and draw the truth table for the (main) control unit.
3. Extend the above datapath to support **{lb, lbu}**; that is, the signed and unsigned versions of the load-byte instruction. Show *only the changes* to the original datapath.
4. Show the modified truth table to generate the controls; mention just the changes/additions from the earlier table.
5. Extend the original datapath to support a new instruction **ldpc**, which loads the value of **PC+4** onto a given register (no need to include support for **lb**, **lbu** here).
6. Show the modified truth table to generate the controls; mention just the changes/additions from the earlier table.

CS230: Digital Logic Design and Computer Architecture

Tutorial 05, [Mon 23 Sep, Tue 24 Sep, Thu 26 Sep]

Concepts tested: Introduction to Pipelining, Structural Hazards, Pipelined Datapath, Data Hazards

1. Suppose we have the `add3` instruction in MIPS, which uses the `MEM` cycle to do the extra addition, using the same ALU. Give an example of how this would cause a structural hazard. Give the corresponding pipeline timing diagram, with the necessary stalls.
2. Given the above stall(s), does `add3` in the instruction set have any performance advantage? Explain.
3. Draw the pipeline timing diagram for the following two-instruction sequence, without any data forwarding, but assuming that the register file can be written in the first half of a cycle and read in the second half. Be sure to mark any repeated stages with a bubble.

```
lw $t0, 4($sp)
sw $t1, 8($t0)
```

4. Exhaustively list all the possible data forwarding required in the MIPS pipeline, to minimize data hazard related stalls. For each forwarding, mention the latch from which forwarding is done, and the stage to which forwarding is done.
5. **Zero offset memory access**
 - (a) Many memory access instructions use zero as offset, with the usual base register. There is a possible advantage in having a special instruction for such cases. Say, we have a special instruction `lw0`, of the form:
`lw0 $t0, ($sp) # equivalent to: lw $t0, 0($sp)`
Show a sequence of two instructions where, if a `lw0` is used instead of the usual `lw`, a pipeline stall can be saved. You may assume that `lw0` uses appropriate data forwarding in the pipelined MIPS implementation.
 - (b) For your two-instruction sequence above, show the pipeline timing diagram for both cases: i.e. using `lw`, using `lw0` instead of `lw`. Mark any stalls and data forwarding appropriately.
 - (c) Show the datapath *change* required to implement data forwarding for the `lw0` instruction above. Show *only the change*, not the entire pipelined datapath.

CS230: Digital Logic Design and Computer Architecture

Tutorial 06, [Mon 30 Sep, Tue 01 Oct, Thu 02 Oct]

Concepts tested: Control Hazards

1. **Compatibility of ideas:** Among the following ideas for dealing with control hazards, which pairs are NOT compatible with one another?

- 3-stage branch completion
- 2-stage branch completion
- Assume branch not taken
- Branch prediction
- Branch target buffer
- Delayed branches

2. **Filling the branch delay slot:**

Consider the following (incomplete) code. It is supposed to execute on a MIPS machine with a 5-stage pipeline implementation (as discussed in the lectures). It has a 2-stage branch completion scheme, with a branch delay slot of exactly one instruction.

L.no:	
L1	_____ # Answer below, not here
L2	beq \$s1, \$zero, NO_IF
L3	<u>nop</u> # delay slot is initially empty
L4	addi \$s2, \$s2, 1
L5	add \$a0, \$s3, \$zero
L6	jal FUNC
L6.5	nop
L7	NO_IF: _____ # Answer below, not here

In the following sub-questions, we will consider various possibilities to schedule the delay slot (i.e. fill a useful instruction in the delay slot). Note that these possibilities are *exclusive* of one another. Also, briefly explain your answer in each case, in a sentence or two.

- (a) Give an example non-branch instruction in line L1, which the compiler *can safely* schedule in the delay slot.
- (b) Give an example non-branch instruction in line L1, which the compiler *cannot* schedule in the delay slot.
- (c) Fill L7 with a non-branch instruction such that the compiler *can safely* schedule the instruction in L7 in the delay slot.
- (d) Fill L7 with a non-branch instruction such that the compiler *cannot* schedule the instruction in L7 in the delay slot.
- (e) Fill L7 with a non-branch instruction such that the compiler *can safely* schedule the instruction in L4 in the delay slot.
- (f) Fill L7 with a non-branch instruction such that the compiler *cannot* schedule the instruction in L4 in the delay slot.
- (g) Fill L1 & L7 with non-branch instructions such that the instruction in L1 can be safely scheduled in the delay slot, but does not end up enhancing the performance.

CS230: Digital Logic Design and Computer Architecture

Tutorial 07, [Mon 07 Oct, Tue 08 Oct, Thu 10 Oct]

Concepts tested: Pipeline Control, Exceptions in the Pipeline

1. Write the data forwarding control logic to the EX stage, including forwarding from lw.
2. Write the stalling control logic for a dependent **reg-reg** following a lw. Can this code be combined with the earlier data forwarding control logic code?
3. Considering a 2-stage **beq**, give the modified datapath to enable the necessary data forwarding (ignore any other data hazards).
4. As per the above datapath modification, write the data forwarding control logic for a dependent 2-stage **beq**, dependent on any earlier **reg-reg** instruction.
5. Write the stalling logic resulting from a control hazard due to a 2-stage **beq**.
6. Suppose that the data memory address misaligned exception is detected early in the MEM stage of LW. Give an example sequence of instructions in which the first instruction of the exception handler is fetched but is flushed due to another exception.

CS230: Digital Logic Design and Computer Architecture

Tutorial 08, [Mon 14 Oct, Tue 15 Oct, Thu 17 Oct]

Concepts tested: Memory Hierarchy, Cache Design

1. What is the difference between registers acting as a cache for L1-cache, versus L1-cache acting as a cache for main memory?
2. What are the three reasons why a cache is faster than main memory?
3. Consider a main memory of 256 words, and a direct mapped cache with 8 blocks of 2 words each. The cache is initially empty. The following sequence of word addresses are accessed by the processor.
2, 3, 11, 16, 21, 13, 64, 48
19, 11, 3, 22, 4, 27, 6, 11
Mark each reference as a hit or a miss, and show the final cache contents. Identify instances of temporal locality and spatial locality.
4. Re-work the above problem assuming that the main memory size is 1024 words.
5. Consider a system with main memory size 32MB, cache size 512KB, 16-word blocks. For the following cases, (i) show the memory address fields as interpreted by the cache controller, (ii) mention the number of comparators required, (iii) indicate the number of bits that must be compared in each comparator, (iv) compute the number of meta-data bits required to implement the cache.
 - (a) Direct mapped
 - (b) 2-way set associative
 - (c) 4-way set associative
 - (d) 8-way set associative
 - (e) Fully associative

CS230: Digital Logic Design and Computer Architecture

Tutorial 09, [Mon 21 Oct, Tue 22 Oct, Thu 24 Oct]

1. (Based on Q7.15 from the textbook). Suppose a processor with a 16-word block size has an effective miss rate per instruction of 0.25%. Assume that the CPI without cache misses is 2.5. The DRAM (main memory) latencies are: 2 cycles for communicating the address, 20 cycles for the access latency, and 2 cycles for communicating the data read. Compute the effective CPI under the following scenarios:
 - (a) 1-word wide memory, with and without interleaving
 - (b) 2-word wide memory, with and without interleaving
 - (c) 4-word wide memory, with and without interleaving
 - (d) 8-word wide memory, with and without interleaving
2. Cache performance tradeoffs (based on Q7.32)

Three processors P1, P2, and P3 are the same, except for their cache configurations.

P1 has a direct-mapped cache with 1-word blocks

P2 has a direct-mapped cache with 4-word blocks

P3 has a 2-way set associative cache with 4-word blocks

The measured miss-rates for the 3 processors, for a particular benchmark program, are:

P1: instruction miss rate = 4%, data miss rate = 6%

P2: instruction miss rate = 2%, data miss rate = 4%

P3: instruction miss rate = 2%, data miss rate = 3%

The benchmark program is such that half the executed instructions have a data memory access. Assume that the cache miss penalty is 6 + block size in words. P1's CPI is measured as 2.0.

 - (a) Compute the ideal CPI, i.e. assuming a perfect cache with 0% miss rate.
 - (b) Compute the CPI of P2 and P3, and determine which of the 3 processors is the fastest.

3. Unusual cache size (based on Q7.30)

Consider a cache of size 3K words of data (note: $1K = 2^{10}$ here). Answer the following, with appropriate explanation.

- (a) Is it possible to organize it as a fully associative cache? As a 2-way set associative cache? As a direct-mapped cache? Assume that the cache controller has to work with bit extractions alone, on the given memory address, and no other complex computations.
- (b) For each of the above cases in (a) which is possible, indicate the maximum possible block size.
- (c) For each of the above cases in (a) which is possible, and for the largest possible block size as computed in (b), show the various memory address fields, when the memory addresses are 32-bit long.

CS230: Digital Logic Design and Computer Architecture

Tutorial 10, [Mon 28 Oct, Tue 29 Oct]

1. Consider a virtual memory system with virtual byte address = 40-bits, physical byte address = 36-bits, and page size = 16 KB.
 - (a) Draw a diagram to show the virtual-to-physical address mapping. Show the various fields of the virtual and physical addresses.
 - (b) In the default page table scheme, what is the total size of the page table for each process? Assume that a page-table entry has a 32-bit disk address, in addition to 4 meta-data bits (the valid, write protection, dirty, and use bits), apart from the fields necessary for the virtual address to physical address mapping.
2. **Multi-level page tables:** A computer has a 32-bit virtual address space, and a 24-bit physical address space. It has 8KB pages. Suppose a 2-level page table arrangement is used. It is convenient to arrange the second level page tables to be of the same size as a page. With this arrangement, and assuming that each page table entry is of length 4 bytes, answer the following. Show brief calculations/steps/explanations.
 - (a) How many memory accesses will a jal instruction produce?
 - (b) How many memory accesses will a lw instruction produce?
 - (c) What is the size of the per-process first-level page table?
 - (d) Draw a diagram to show how a virtual address is translated to a physical address in this arrangement.

CS230: Digital Logic Design and Computer Architecture

Tutorial 11, [Mon 04 Nov, Tue 05 Nov, Thu 07 Nov]

1. (Based on 7.37) Rank each of the following event combinations according to how frequently they are likely to occur. Explain your answer briefly.
 - (a) TLB miss followed by page fault followed by cache miss
 - (b) TLB miss followed by page hit followed by cache miss
 - (c) TLB hit followed by page hit followed by cache miss
 - (d) TLB miss followed by page hit followed by cache hit
2. A computer has a 32-bit virtual address space, and a 24-bit physical address space. It has 8KB pages. The TLB is shared between instructions and data. It has 256 entries and is 8-way set associative. The TLB block size is one entry.
 - (a) Draw a diagram showing the various TLB fields, and the mapping from virtual memory to physical memory, on a TLB hit.
 - (b) How will the TLB fields change if the TLB were fully associative? If it were direct-mapped?
3. Describe the aliasing problem with virtually addressed caches.
4. Mr. Buddhimaan, after having done the computer architecture course, proposes to solve the aliasing problem with virtually addressed caches as follows. On each process context switch, flush the cache. Will this solve the aliasing problem? Why or why not?
5. What is done to prevent the requirement to flush the TLB on each context switch? Explain briefly.
6. **Hit-miss possibilities:** Consider a computer with a TLB, L1 cache, L2 cache, and virtual memory pages. Now, TLB is a cache for page-table entries, L1 is a cache for L2 blocks, L2 is a cache for main memory blocks, and main memory pages are a cache for pages on disk. During program execution, a

particular memory access potentially involves the above four caches, and each of caches can result in a hit or a miss. So overall there are $2^4 = 16$ combinations. List these combinations, and comment on each. Also indicate which combinations are impossible and why.

7. **Stuck at TLB:** Suppose a TLB implements a most-recently-used eviction policy for its entries. Construct a situation where the processor *cannot* execute an instruction. State the necessary conditions under which this will happen.
8. **Non-restartable instruction:** In the context of page-faults, a restartable instruction is one which can be restarted after handling a page-fault which was caused during the instruction's execution. In the MIPS subset we have seen, all instructions are restartable. The Intel IA32 architecture has a single instruction which can perform string copy (strictly speaking, it is memcpy). Prove that this instruction is non-restartable. *Hint: think of some special cases in the specification of the origin and destination strings.*