Computer Systems Organization
CSCI-UA.0201-001 Fall 2019
SAMPLE Final Exam

ANSWERS

1. True/False. Please circle the correct response.

   a. F In the C and assembly calling convention that we used for this class, all arguments to a function call are passed in registers (no matter how many).

   Only the first 6 parameters (in Unix) or the first 4 parameters (on Windows) are passed in registers.

   b. F In C, for the expression `(x | THE_MASK)`, where `THE_MASK` has at least one bit that is not zero, the result will be zero if all the bits of `x` are zero.

   Even if `x` is all zeros, the result will have a 1 anywhere `THE_MASK` has a 1.

   c. T In x86-64 assembly, the instruction `movq 16(%rcx),%rax` will add 16 to the value contained in the %rcx register in order to compute a memory address.

   d. T In x86-64 assembly, the instruction `popq %rbx` adds 8 to the value of the %rsp register.

   e. F When processor with a direct-mapped cache issues a request to load data from memory, if the tag in the specified address matches the tag in the corresponding cache entry, then a cache hit results – no matter what the status of any other bit in the cache entry is.
   The valid bit has to be 1 as well.

   f. T Any circuit constructed using only AND, OR, or NOT gates can also be constructed using only NAND gates (which perform an AND and then a NOT).

   This was stated in class.

   g. F The use of a write-through cache is slower than a write-back cache because, using a write-through cache, the CPU must wait until the data is written to main memory before proceeding with the next instruction.

   The CPU does not need to wait for a write to finish in either case.

   h. T A multiplexer uses N select lines to select from 2^N input lines.

   i. T A 32-bit adder can be used for subtraction if each bit of the second operand is first sent through a NOT gate and the carry-in to the adder is set to 1.

   j. T In an unclocked latch, setting both the S and the R inputs to 0 will cause the output Q to retain its current value.

2.

   a. Write the number AB31 hex in binary: 1010 1011 0011 0001.

   b. Write the number 73 decimal in hex: 49 and in binary: 1001001.

   c. log 16G = 34 .

   d. In order to access all bytes in a 64GB memory, an address must have at least 36 bits.

e. If an integer variable is represented by 25 bits, how many different numbers could that variable take on? 32M

3. Write a C function, `int foo(int x)`, that returns the index of the most significant bit of `x` whose value is 1. For example, if bit 15 of `x` is the most significant bit of `x` whose value is 1, then `foo` should return 15. If no bits are 1, then `foo` should return -1.

   Answer: There are many ways to do this, either shifting `x` or shifting a mask.

```
int foo(int x)
{
  unsigned int mask = (1 << 31);
  int index = 31;
  while ((index >= 0) && !(x & mask)) {
    mask >>= 1;
    index--;
  }
  return index;
}
```

4. Given the following definition of a `CELL` type used for the elements of linked lists,

```
typedef struct cell {
   long x;
   struct cell *next;
} CELL;
```

   a. Write a C function, `long sumList(CELL *head)`, that returns the sum of the `x` fields in the cells of a linked list whose first element is pointed to by `head`.

      Answer:

```
long sumList(CELL *head)
{
  CELL *p = head;
  long sum = 0;
  while (p != NULL) {
    sum += p->x;
    p = p->next;
  }
  return sum;
}
```

   b. Translate your C function into x86-64 assembly code for either Unix (e.g. MacOS or Linux) or Cygwin/Windows. You can assume that the two fields of a `CELL` are next to each other, with `x` coming first. The calling conventions for Unix and Windows are provided at the end of this exam.

      Answer: Note that if %rcx points to a CELL, then the x field is at 0(%rcx) and the next field is at %8(%rcx). Below is the Unix version.

```
_sumList:
      pushq %rbp
      movq  %rsp,%rbp
      movq  %rdi,%rdx     // p = head
      movq  $0,%rax       // sum = 0
LOOP:
      cmpq  $0,%rdx       // if (p == NULL)
      je    DONE          //   jump out of loop
      addq  0(%rdx),%rax  // sum += p->x
      movq  8(%rdx),%rdx  // p = p->next
      jmp   LOOP
DONE:                      // sum is already in %rax
      popq  %rbp
      ret
```

The Windows version would be identical, except the function name would just be "`sumlist`" and the parameter, `head`, would be in `%rcx` rather than `%rdi`.

5.

a. On modern processors, a C program whose main loop reads the elements of a huge array in random order will run much slower than an otherwise identical C program that reads the array elements in consecutive order. Why is that?

Answer: On modern processors, caches have multi-word cache lines and therefore exploit spatial locality. A program that accesses the elements of an array in consecutive order will have a cache miss the first time it accesses an array element that is not already in the cache, but accessing successive elements that are in the same cache line will result in cache hits. A program that accesses array elements randomly will not take advantage of multi-word cache lines and will result in many more cache misses.

b. Suppose that, on a computer with only one cache, the following holds:

- the cache has a 90% hit rate for both data and instructions,
- if an instruction does not cause a cache miss, executing the instruction takes one cycle,
- the cache miss penalty is 200 cycles

What is the overall execution time (in cycles) of a program that executes N instructions, where 25% of the instructions access data in memory?

Answer: The total number of cycles will be the sum of the following:

- The N cycles it takes to execute the N instructions, once the instructions and data have already been fetched.
- The cycles it takes to fetch the 10% of instruction fetches that result in cache misses and pay a 200 cycle penalty, so $N \times 0.1 \times 200$.
- The cycles it takes to access the data from memory that 25% of instructions need, where, of those data accesses, 10% miss the cache, so $N \times 0.25 \times 0.1 \times 200$.

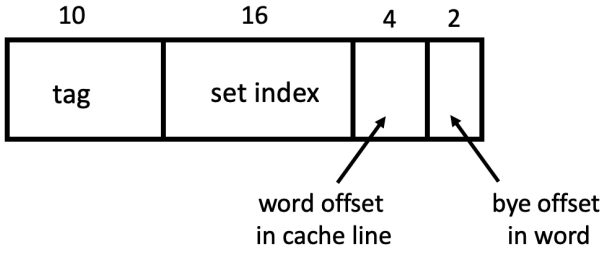
Thus, the total number of cycles is $N + (N \times 0.1 \times 200) + (N \times 0.25 \times 0.1 \times 200) = N + 20N + 5N = 26N$. Note that, in this case, instructions and data have the same cache miss rate, but that is not necessarily the case.

c. On a machine with 32-bit words, suppose there is an 8MB, 2-way set associative cache with 16 words per cache line. Further, suppose that an instruction issues a memory address for loading a single word from memory into a register. Indicate how the various bits of the address are used by the cache hardware to determine whether a cache hit has occurred and, if so, to return the requested word.

Answer:

- Since there are 8M bytes in the cache and 4 bytes per word, the number of words in the cache is $8M/4 = 2M = 2^{21}$.
- Since there are 16 words per cache line, the number of cache lines in the cache is $2^{21}/16 = 2^{21}/2^4 = 2^{17}$.
- Since there are two cache lines per set, the number of sets in the cache is $2^{17}/2 = 2^{16}$.

Therefore, starting from the least significant bits, the address contains 2 bits for the byte offset within a word, $\log 16 = 4$ bits for the word offset within a cache line, $\log 2^{16} = 16$ bits to select the set within the cache, and the remaining 10 bits for the tag:



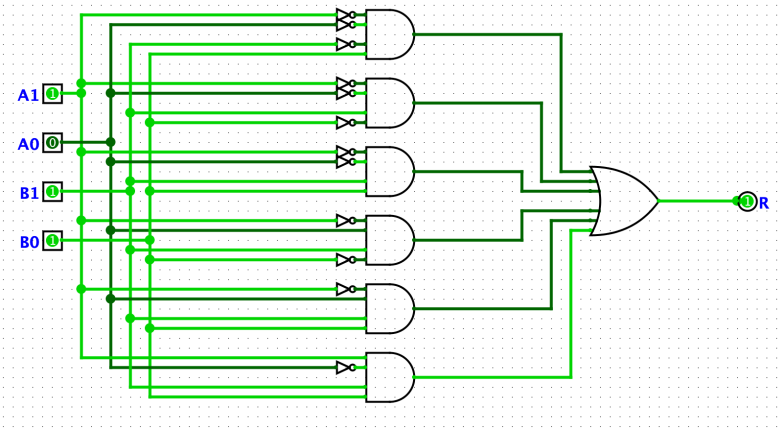
6.

a. Build, from AND, OR, and NOT gates, a circuit that represents the two-bit "less-than" function. That is, it has two two-bit inputs, A and B, and a single one-bit output, R, such that R is true when A< B. [Hint: enumerate the possible inputs for which the output is true.]

Answer: The brute force method (which is fine) is to create a truth table that just lists the rows in which R is true, rather than all 16 possible combinations of A and B:

A1	A0	B1	B0	R
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	1	0	1
0	1	1	1	1
1	0	1	1	1

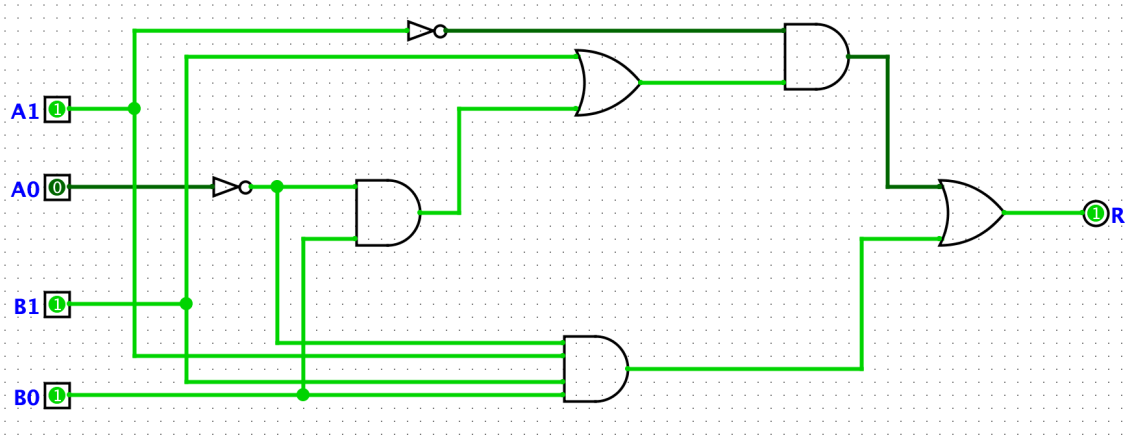
This would result in the following circuit:



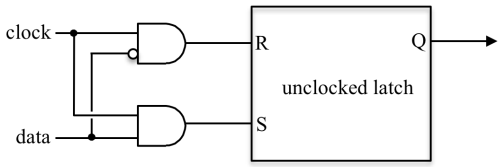
Alternatively, you could simplify this by realizing that:

- If A1 = 0, then R is true if B1 = 1 or if A0 = 0 and B0 = 1.
- IF A1 = 1, the R is true if A0 = 0 and both B0 and B1 = 1.

The circuit would then be:



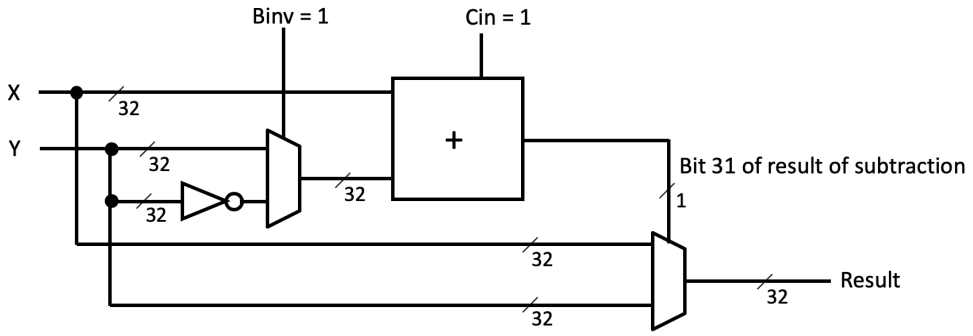
b. As you saw in class, a clocked latch is built from an unclocked latch as shown below. Why are flip-flops used for storing bits in a CPU rather than clocked latches?



Answer: In the above clocked D-latch, as long as the clock is up (1), the data will flow into the unclocked latch and appear on the output Q. However, for registers, we don't want the new data coming in a register (as the result coming from an ALU) to affect the output of the register (which may be going to the ALU), while a computation is occurring. Therefore, we build registers out of flip-flops so that the output of a register doesn't change, even if the register is being written to, until the clock falls.

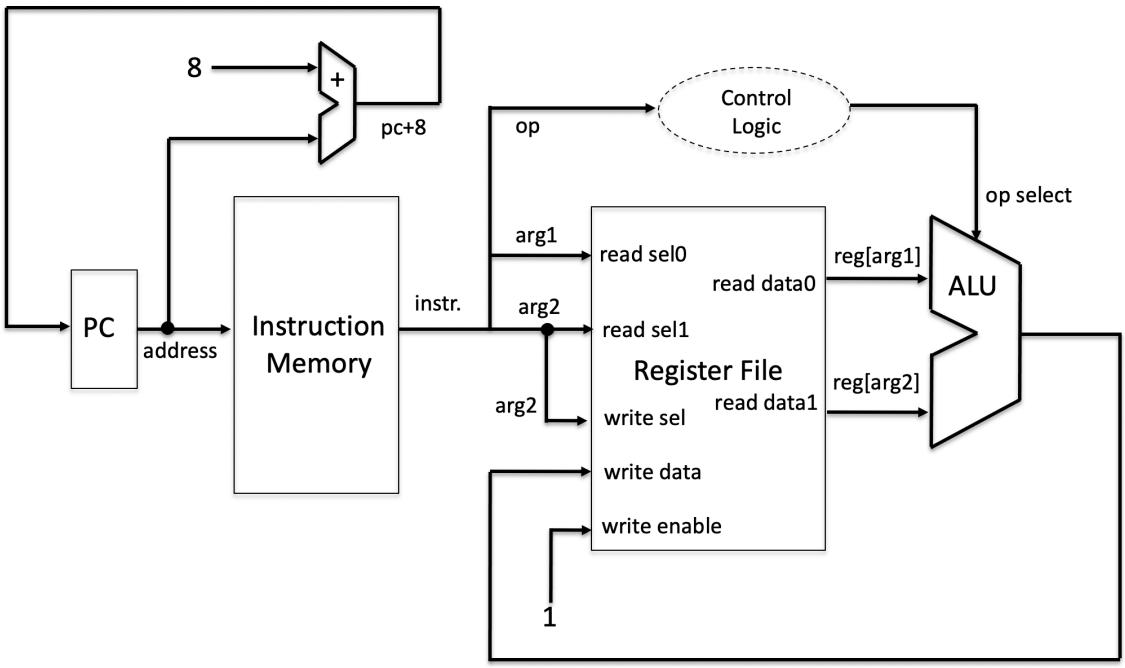
c. Build from gates, multiplexers, decoders, and/or adders (including 32-bit versions of each) a circuit that takes two 32-bit inputs, X and Y, and outputs the value of the larger of X and Y. For example, if the value of X is 20 and the value of Y is 15, then the output of your circuit should be 20.

Answer: A simple way to do this is to subtract Y from X and then use the leftmost bit of the result (bit 31, which will be 1 if Y > X) as the select line of a multiplexer that sends X or Y to the output.



d. Draw the datapath for an X64 instruction, such as `addq`, that has two registers as operands, where the second operand is also the destination register (where the result is put). Include the portions of the processor that use the PC to fetch the instruction and update the PC to point to the next instruction.

Answer: See the lecture notes and below.



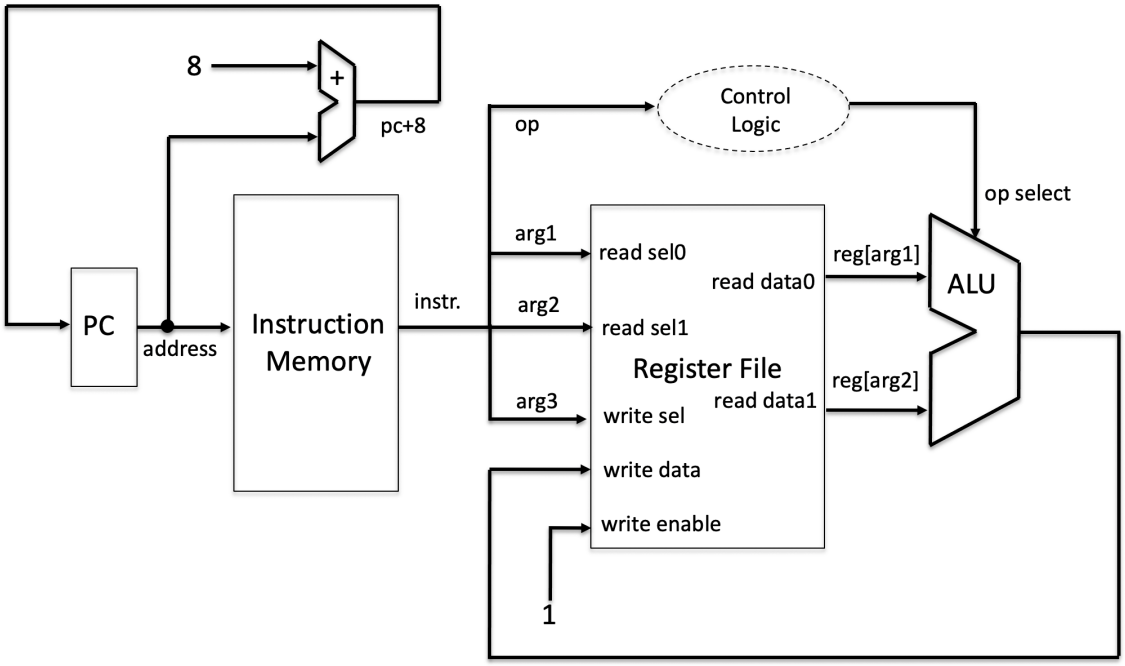
e. Suppose that, instead of having two register operands, the `addq` instruction specified three registers, where the third operand is the destination. That is, for example, suppose the `addq` instruction is written:

```
addq  %rax,%rbx,%rcx    # sets %rcx = %rax + %rbx
```

In this case, what would be different about the datapath than what you drew for the previous part?

Answer:

The only difference is that rather than sending arg2 (the second operand) in the instruction to the write-select input of the register file, arg2 (the third operand in the instruction) would be sent to the write-select input, as shown below.



X86-64 Calling Conventions

Unix (e.g. macOS and Linux)

Passing Parameters:

- First six integer/pointer parameters (in order): %rdi, %rsi, %rdx, %rcx, %r8, %r9
- Additional parameters are passed on the stack, pushed in right-to-left (reverse) order.

Return Value:

- %rax (for 64-bit result), %eax (for 32-bit result)

Caller-Saved Registers (may be overwritten by called function):

- %rax , %rcx , %rdx , %rdi , %rsi , %rsp , %r8, %r9, %r10, %r11

Callee-Saved Registers (must be preserved – or saved and restored – by called function):

- %rbx, %rbp, %r12, %r13, %r14, %r15

Microsoft (e.g. for Windows)

Passing Parameters:

- First four integer/pointer parameters (in order): %rcx, %rdx, %r8, %r9
- Additional parameters are passed on the stack, pushed in right-to-left (reverse) order.
- Important: The caller must allocate 32 bytes on stack (by subtracting 32 from %rsp) right before calling the function. Don't forget to restore the %rsp (by adding 32) after the call.

Return Value:

- %rax (for 64-bit result), %eax (for 32-bit result)

Caller-Saved Registers (may be overwritten by called function):

- %rax, %rcx, %rdx, %r8, %r9, %r10, %r11

Callee-Saved Registers (must be preserved – or saved and restored – by called function):

- %rbx, %rbp, %rdi, %rsi, %rsp, %r12, %r13, %r14, %r15