

◇ CSCI 2500 — Computer Organization ◇
Fall 2019 Quiz 2 (October 2, 2019)

Xinhao Luo	luox6@rpi.edu Lab section: 3
Room: Sage 3303 Zone: BLUE Row: 6 Seat: 2	7:00 pm - 7:50 pm



Please silence and put away all laptops, notes, books, phones, electronic devices, etc. This quiz is designed to take 50 minutes; therefore, for 50% extra time, the expected time is 1 hour and 15 minutes and 100% extra time is 1 hour and 40 minutes. Questions will not be answered except when there is a glaring mistake or ambiguity in the statement of a question. Please do your best to interpret and answer each question.

1. (15 POINTS) What is the exact terminal output of the C code below, assuming a 64-bit architecture? Note that this code compiles and executes without crashing; further, `#include` directives are not shown. Write your response next to each `printf()`.

```
int main(void)
{
    int q = 10;
    int * p = &q;
    char * s = calloc( q, sizeof( char ) );
    printf( "AA%d%lu-%lu%dB\n", q, sizeof( int ), sizeof( int * ), *p );
    strcpy( s, "LLMMNNOO" );
    char * t = s + 5;
    *t = 'Z';
    printf( "%s\n", t );
    t -= 2;
    *t = '\0';
    printf( "%s\n", s );
    free( s );
    return EXIT_SUCCESS;
}
```

AA104-810BB
ZOO
LMM

2. (15 POINTS) Consider we have a function `filesize(FILE* fp)` that tells us the size of a file in bytes. What might the expected values of `x` and `y` from the code snippet below be if we assume one byte per character in ASCII text files?

```
char* a = "Engineers";
FILE* fp1 = fopen("file1", "w");
FILE* fp2 = fopen("file2", "w");
fprintf(fp1, "%s\n", a);
fwrite(a, 1, strlen(a), fp2);
int x = filesize(fp1);
int y = filesize(fp2);
```

x:

10

y:

9

3. (20 POINTS) The code below should have a single parent process read in a two dimensional matrix, formatted similarly to inputs for Homework 1. The parent should then spawn exactly one child for each column of the matrix. The children should only determine the maximum value for one single column and then execute `/bin/echo` with an argument of that maximum value. A child's column should be assigned in the order it was spawned by the parent, and the assignment should be bijective (one-to-one).

The function listings from the man page for each library call are given below. Assume we have all necessary includes. Find and correct all the bugs. Hint: 4 points per bug found up to 20 total points; false positives will be penalized.

```

pid_t getpid(void);
void *malloc(size_t size);
int scanf(const char *format, ...);
pid_t fork(void);
pid_t wait(int *status);
int execl(const char *path, const char *arg, ..., (char *) NULL);

int main()
{
    int max, stat, i, j;
    int rows = 8;
    int cols = 4;
    int pid = getpid();
    int** matrix = (int**)malloc(rows*sizeof(int));
    for (i = 0; i < rows; ++i)
        matrix[i] = (int*)malloc(rows*sizeof(int));

    for (i = 0; i < rows; ++i)
        for (j = 0; j < cols; ++j)
            scanf("%d", matrix[i][j]);

    for (i = 0; i < cols; ++i)
        if (!fork())
            for (j = 0; j < rows; ++j)
                if (matrix[i][j] > max)
                    max = matrix[i][j];

    if (getpid() == pid)
        for (i = 0; i < cols; ++i)
            wait(&stat);
    else
        execl("/bin/echo", "echo", max, NULL);

    return EXIT_SUCCESS;
}

```

Handwritten annotations:

- ① \rightarrow size of (int*)
- ② \rightarrow &matrix[i][j]
- ③ \rightarrow max = 0
- ④ \Rightarrow matrix[j][i]
- ⑤ \rightarrow output

4. (15 POINTS) Give the values stored in temporary registers \$t0, \$t1, \$t2, and \$t3 after the following block of MIPS code executes. You can give decimal integer values or binary (you only need to show the least significant byte of each word).

```

li    $t0, 3
sll   $t1, $t0, 2 → 1100
or    $t2, $t1, $t0  1100
and   $t3, $t2, $t0  0011

```

Handwritten calculations:

$$2^0 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15$$

Register	Value
\$t0	3
\$t1	12
\$t2	15
\$t3	3

5. (10 POINTS)

Consider two processors P1 and P2. P1 has clock rate 3.0GHz and a CPI of 1.5. P2 has a clock rate of 2.5GHz and a CPI of 1.0.

Recall: $CPU\ Time = \frac{Instruction\ Count \times CPI}{Clock\ Rate}$.

$$\frac{3}{1.5} = \frac{6}{3} = 2$$

Calculate the number of instructions per second that each processor executes.

$$P_1 : \frac{3 \times 10^9}{1.5} = 2 \times 10^9 \text{ instructions / s.}$$

$$P_2 : \frac{2.5 \times 10^9}{1.0} = 2.5 \times 10^9 \text{ instructions / s.}$$

If each processor executes a program in 10 seconds, how many instructions does each processor execute?

$$P_1 : 2 \times 10^9 \times 10s = 2 \times 10^{10} \text{ inst.}$$

$$P_2 : 2.5 \times 10^9 \times 10s = 2.5 \times 10^{10} \text{ inst.}$$

1 line \Rightarrow 4 byte.

Xinhao Luo luox6@rpi.edu

6. (15 POINTS) Consider the following memory dump of the data segment in QtSpim:

The screenshot shows the QtSpim interface. On the left, the 'Int Regs [16]' panel displays register values: PC=0, EPC=0, Cause=0, BadVAddr=0, Status=3000ff10, HI=0, LO=0, RO[r0]=0, R1[at]=0. The main window shows the 'Data' segment starting at address 10000000. The memory dump is as follows:

Address	Value (hex)	Value (dec)
10000000	00000000	0
10000004	00000065	101
10000008	00000012	18
1000000c	00000000	0
10000010	00000000	0
10000014	00000000	0
10000018	00000000	0
1000001c	00000000	0
10000020	00000000	0
10000024	00000000	0
10000028	00000000	0
1000002c	00000000	0
10000030	00000000	0
10000034	00000000	0
10000038	00000000	0
1000003c	00000000	0
10000040	00000000	0
10000044	00000000	0
10000048	00000000	0
1000004c	00000000	0
10000050	00000000	0
10000054	00000000	0
10000058	00000000	0
1000005c	00000000	0
10000060	00000000	0
10000064	00000000	0
10000068	00000000	0
1000006c	00000000	0
10000070	00000000	0
10000074	00000000	0
10000078	00000000	0
1000007c	00000000	0
10000080	00000000	0
10000084	00000000	0
10000088	00000000	0
1000008c	00000000	0
10000090	00000000	0
10000094	00000000	0
10000098	00000000	0
1000009c	00000000	0
100000a0	00000000	0
100000a4	00000000	0
100000a8	00000000	0
100000ac	00000000	0
100000b0	00000000	0
100000b4	00000000	0
100000b8	00000000	0
100000bc	00000000	0
100000c0	00000000	0
100000c4	00000000	0
100000c8	00000000	0
100000cc	00000000	0
100000d0	00000000	0
100000d4	00000000	0
100000d8	00000000	0
100000dc	00000000	0
100000e0	00000000	0
100000e4	00000000	0
100000e8	00000000	0
100000ec	00000000	0
100000f0	00000000	0
100000f4	00000000	0
100000f8	00000000	0
100000fc	00000000	0
10000100	00000000	0
10000104	00000000	0
10000108	00000000	0
1000010c	00000000	0
10000110	00000000	0
10000114	00000000	0
10000118	00000000	0
1000011c	00000000	0
10000120	00000000	0
10000124	00000000	0
10000128	00000000	0
1000012c	00000000	0
10000130	00000000	0
10000134	00000000	0
10000138	00000000	0
1000013c	00000000	0
10000140	00000000	0
10000144	00000000	0
10000148	00000000	0
1000014c	00000000	0
10000150	00000000	0
10000154	00000000	0
10000158	00000000	0
1000015c	00000000	0
10000160	00000000	0
10000164	00000000	0
10000168	00000000	0
1000016c	00000000	0
10000170	00000000	0
10000174	00000000	0
10000178	00000000	0
1000017c	00000000	0
10000180	00000000	0
10000184	00000000	0
10000188	00000000	0
1000018c	00000000	0
10000190	00000000	0
10000194	00000000	0
10000198	00000000	0
1000019c	00000000	0
100001a0	00000000	0
100001a4	00000000	0
100001a8	00000000	0
100001ac	00000000	0
100001b0	00000000	0
100001b4	00000000	0
100001b8	00000000	0
100001bc	00000000	0
100001c0	00000000	0
100001c4	00000000	0
100001c8	00000000	0
100001cc	00000000	0
100001d0	00000000	0
100001d4	00000000	0
100001d8	00000000	0
100001dc	00000000	0
100001e0	00000000	0
100001e4	00000000	0
100001e8	00000000	0
100001ec	00000000	0
100001f0	00000000	0
100001f4	00000000	0
100001f8	00000000	0
100001fc	00000000	0
10000200	00000000	0
10000204	00000000	0
10000208	00000000	0
1000020c	00000000	0
10000210	00000000	0
10000214	00000000	0
10000218	00000000	0
1000021c	00000000	0
10000220	00000000	0
10000224	00000000	0
10000228	00000000	0
1000022c	00000000	0
10000230	00000000	0
10000234	00000000	0
10000238	00000000	0
1000023c	00000000	0
10000240	00000000	0
10000244	00000000	0
10000248	00000000	0
1000024c	00000000	0
10000250	00000000	0
10000254	00000000	0
10000258	00000000	0
1000025c	00000000	0
10000260	00000000	0
10000264	00000000	0
10000268	00000000	0
1000026c	00000000	0
10000270	00000000	0
10000274	00000000	0
10000278	00000000	0
1000027c	00000000	0
10000280	00000000	0
10000284	00000000	0
10000288	00000000	0
1000028c	00000000	0
10000290	00000000	0
10000294	00000000	0
10000298	00000000	0
1000029c	00000000	0
100002a0	00000000	0
100002a4	00000000	0
100002a8	00000000	0
100002ac	00000000	0
100002b0	00000000	0
100002b4	00000000	0
100002b8	00000000	0
100002bc	00000000	0
100002c0	00000000	0
100002c4	00000000	0
100002c8	00000000	0
100002cc	00000000	0
100002d0	00000000	0
100002d4	00000000	0
100002d8	00000000	0
100002dc	00000000	0
100002e0	00000000	0
100002e4	00000000	0
100002e8	00000000	0
100002ec	00000000	0
100002f0	00000000	0
100002f4	00000000	0
100002f8	00000000	0
100002fc	00000000	0
10000300	00000000	0
10000304	00000000	0
10000308	00000000	0
1000030c	00000000	0
10000310	00000000	0
10000314	00000000	0
10000318	00000000	0
1000031c	00000000	0
10000320	00000000	0
10000324	00000000	0
10000328	00000000	0
1000032c	00000000	0
10000330	00000000	0
10000334	00000000	0
10000338	00000000	0
1000033c	00000000	0
10000340	00000000	0
10000344	00000000	0
10000348	00000000	0
1000034c	00000000	0
10000350	00000000	0
10000354	00000000	0
10000358	00000000	0
1000035c	00000000	0
10000360	00000000	0
10000364	00000000	0
10000368	00000000	0
1000036c	00000000	0
10000370	00000000	0
10000374	00000000	0
10000378	00000000	0
1000037c	00000000	0
10000380	00000000	0
10000384	00000000	0
10000388	00000000	0
1000038c	00000000	0
10000390	00000000	0
10000394	00000000	0
10000398	00000000	0
1000039c	00000000	0
100003a0	00000000	0
100003a4	00000000	0
100003a8	00000000	0
100003ac	00000000	0
100003b0	00000000	0
100003b4	00000000	0
100003b8	00000000	0
100003bc	00000000	0
100003c0	00000000	0
100003c4	00000000	0
100003c8	00000000	0
100003cc	00000000	0
100003d0	00000000	0
100003d4	00000000	0
100003d8	00000000	0
100003dc	00000000	0
100003e0	00000000	0
100003e4	00000000	0
100003e8	00000000	0
100003ec	00000000	0
100003f0	00000000	0
100003f4	00000000	0
100003f8	00000000	0
100003fc	00000000	0
10000400	00000000	0
10000404	00000000	0
10000408	00000000	0
1000040c	00000000	0
10000410	00000000	0
10000414	00000000	0
10000418	00000000	0
1000041c	00000000	0
10000420	00000000	0
10000424	00000000	0
10000428	00000000	0
1000042c	00000000	0
10000430	00000000	0
10000434	00000000	0
10000438	00000000	0
1000043c	00000000	0
10000440	00000000	0
10000444	00000000	0
10000448	00000000	0
1000044c	00000000	0
10000450	00000000	0
10000454	00000000	0
10000458	00000000	0
1000045c	00000000	0
10000460	00000000	0
10000464	00000000	0
10000468	00000000	0
1000046c	00000000	0
10000470	00000000	0
10000474	00000000	0
10000478	00000000	0

7. (10 POINTS) Your MIPS program takes 60 seconds to execute. You analyze your code and conclude that arithmetic operations account for 50% of the execution time, and load and store operations account for 40% of the execution time. There is a potential of improving arithmetic operations execution time by a factor of 5, and load and store operations by a factor of 6.

Recall: $T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$

Part a: (5/10 points) Assuming you can only improve either arithmetic or load and store operations, given the improvement factors specified above, which would yield the best improvement of the total execution time of your program?

(a) arithmetic

(b) load and store

(c) improvement would be the same

(d) no improvement either way

Part b: (5/10 points) Now assume that you can improve both arithmetic and load and store operations, given the improvement factors specified above. In the box below, indicate the total execution time of your program after improvements.

$$T_{total} = \frac{60 \times 50\%}{5} + \frac{60 \times 40\%}{6} + 60 \times 10\% = 16 \text{ s.}$$