

Name: _____

Roll No: _____

(COL 216) Computer Architecture

Feb 22, 2025

Minor

Duration: 120 minutes

(60 marks)

Beware: Be concise in your writing. You can use rough sheets for calculations. But you cannot submit any additional sheet for grading on Gradescope. So make sure you are certain when you write something (after rough work, or use a dark pencil). If you cheat, you will surely get an F in this course.

1. Calculate the latency, throughput, energy, throughput increase and energy increase over non-pipelined, to fill the missing values in last two rows of Table 2. Assume n pipeline registers added, one after each stage, for a pipeline of depth n . Show briefly how you derive the values. [5 marks].

	Delay	Energy
Instruction	100 ns	100 pJ
Pipeline register	2 ns	2 pJ

Table 1: Pipeline stage and register latency and energy requirements

Pipeline depth	Latency	Throughput	Energy	Throughput increase	Energy increase
0	100 ns	100 ns/result	100 pJ	1.0x	1.0x
10					
100					

Table 2: Throughput and energy increase with increasing pipeline depth

Name: _____

Roll No: _____

2. In a single-cycle processor, all instructions use one common clock cycle. The table shows the time needed by three instruction types, and their respective percentage in a set of instructions. Show how 49% time is wasted in this single-cycle processor for this mix of instructions. If we change the design to a multi-cycle processor with each cycle of 100 ps, each instruction can take multiple clock cycles to finish. What percentage of time will be wasted for the same instruction mix? Both these processors are non-pipelined. [5 marks]

Instruction	Time	% of instructions
Type 1	800 ps	30%
Type 2	200 ps	20%
Type 3	250 ps	50%

Table 3: Instruction details

Name: _____

Roll No: _____

3. The following MIPS assembly code, corresponding to the given C code, has two branches. Consider the three possible flow of executions when (a) neither branches are taken i.e. we enter the loop and the if condition also holds, (b) the first branch is not taken and second branch is taken i.e. we enter the loop, but the if condition doesn't hold, and (c) the first branch is taken, exiting the loop.

```

1 int* myptrs[100];
2 int temp = 0;
3 for(int i = 0; i < 100; i++){
4     int* val = myptrs[i];
5     if(val != 0){
6         *val = temp;
7         temp++;
8     }
9 }
10 retval = temp;

```

```

1                                     #myptrs[0] addr in $s3
2     add $s0, $0, $0                   #temp = 0
3     add $s1, $0, $0                   #i = 0
4     addi $s2, $0, 100                 #loop limit
5     add $s4, $0, $0                   #myptrs offset
6 loop:    slt $t0, $s1, $s2
7         beq $t0, $0, end
8         add $t0, $s3, $s4             #calculate myptrs[i] addr
9         lw $t0, 0($t0)                #get myptrs[i]
10        beq $t0, $0, afterif
11        sw $s0, 0($t0)                 #*val = temp
12        addi $s0, $s0, 1               #temp++
13 afterif: addi $s1, $s1, 1             #i++
14        addi $s4, $s4, 4               #myptrs offset+4
15        j loop
16 end:    add $v0, $s0, $0              #retval=temp

```

We have a standard 5 stage pipeline with Instruction Fetch (F), Decode (D), Execute (E), Memory (M) and Writeback (W) stages. The pipeline has forwarding/bypassing enabled. Branches are resolved in second decode stage of the pipeline.

- Fill in the pipeline details (i.e. F/D/E/M/W stage happens for which instruction in which cycle) in the following three tables, for the three cases of execution flow. **[5 + 4 + 3 = 12 marks]**
- If the if condition holds true 50% of the time, what is the average CPI of running all loop iterations? **[3 marks]**

Name: _____

Roll No: _____

Instruction	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16	t17	t18
slt \$t0, \$s1, \$s2																		
beq \$t0, \$0, end																		
add \$t0, \$s3, \$s4																		
lw \$t0, 0(\$t0)																		
beq \$t0, \$0, afterif																		
sw \$s0, 0(\$t0)																		
addi \$s0, \$s0, 1																		
addi \$s1, \$s1, 1																		
addi \$s4, \$s4, 4																		
j loop																		
add \$v0, \$s0, \$0																		

Table 4: Pipeline diagram when no branches are taken

Instruction	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16	t17	t18
slt \$t0, \$s1, \$s2																		
beq \$t0, \$0, end																		
add \$t0, \$s3, \$s4																		
lw \$t0, 0(\$t0)																		
beq \$t0, \$0, afterif																		
sw \$s0, 0(\$t0)																		
addi \$s0, \$s0, 1																		
addi \$s1, \$s1, 1																		
addi \$s4, \$s4, 4																		
j loop																		
add \$v0, \$s0, \$0																		

Table 5: Pipeline diagram when the first branch is not taken, but second branch is taken as if fails

Instruction	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16	t17	t18
slt \$t0, \$s1, \$s2																		
beq \$t0, \$0, end																		
add \$t0, \$s3, \$s4																		
lw \$t0, 0(\$t0)																		
beq \$t0, \$0, afterif																		
sw \$s0, 0(\$t0)																		
addi \$s0, \$s0, 1																		
addi \$s1, \$s1, 1																		
addi \$s4, \$s4, 4																		
j loop																		
add \$v0, \$s0, \$0																		

Table 6: Pipeline diagram when the first branch is taken to exit loop

Name: _____

Roll No: _____

Name: _____

Roll No: _____

4. The j loop in the following code runs 100 million times, and the k loop runs 1 billion times. Compare the number of mispredictions for the three branch predictors: (a) 1 bit saturating counter initialized with 0, (b) 2 bit saturating counter initialized with 00 and (c) a Backward Taken Forward Not Taken (BTFNT) predictor, which always says branch taken if the branch target address is less than the current PC, and always says not taken if the branch target address is more than the current PC. BTFNT prediction is very effective in loops, every time the loop is taken.

```
1     for(j = 0; j < 100000000; j++){
2         for(k = 0; k < 10; k++){
3             //do something
4         }
5     }
```

Fill in the table below, and also explain briefly how you get the values. [5 marks]

1 bit	2 bit	BTFNT

Table 7: Number of mispredicted branches in the code above

Name: _____

Roll No: _____

5. Complete the C function `mystery1` so it behaves the same way as the RISC-V assembly below. Assume that the function arguments are in registers `x11` through `x15`, a.k.a. `a1` through `a5`. Also assume that any array length given as an input is greater than zero. **[5 marks]**

```
1 void mystery1(int *arr1, int *arr2, int *arr3, int size, int num) {
2     // ???
3 }
```

```
1     loop: lw    x5, 0(x12)
2           mul   x5, x5, x15
3           lw    x6, 0(x13)
4           add   x6, x6, x5
5           sw    x6, 0(x11)
6           addi  x11, x11, 4
7           addi  x12, x12, 4
8           addi  x13, x13, 4
9           addi  x14, x14, -1
10          bne   x14, x0, loop
11          ret
12
```

Name: _____

Roll No: _____

6. Complete the C function `mystery2` so it behaves the same way as the RISC-V assembly below. The function arguments are in registers `a1` through `a3` (a.k.a. `x11` through `x13`). Register `x10` is used to store the result of `mystery2`. You can assume that any array length given as an input is greater than zero. **[7 marks]**

```
1  int mystery2(int* arr1, int* arr2, int size) {
2  // ...
3  }

1  addi x10, x0, 0
2
3  loop: lw x6, 0(x12)
4        bne x6, x0, foo
5        j bar
6
7  foo:  sw x6, 0(x11)
8        addi x11, x11, 4
9        addi x10, x10, 1
10
11 bar:  addi x12, x12, 4
12        addi x13, x13, -1
13        bne x13, x0, loop
14
15 ret
```

Name: _____

Roll No: _____

7. Show the values of registers divisor, quotient and remainder for all iterations when 74 is divided by 21 using the basic divisor circuit below. Assume both inputs are unsigned 6-bit integers. [8 marks]

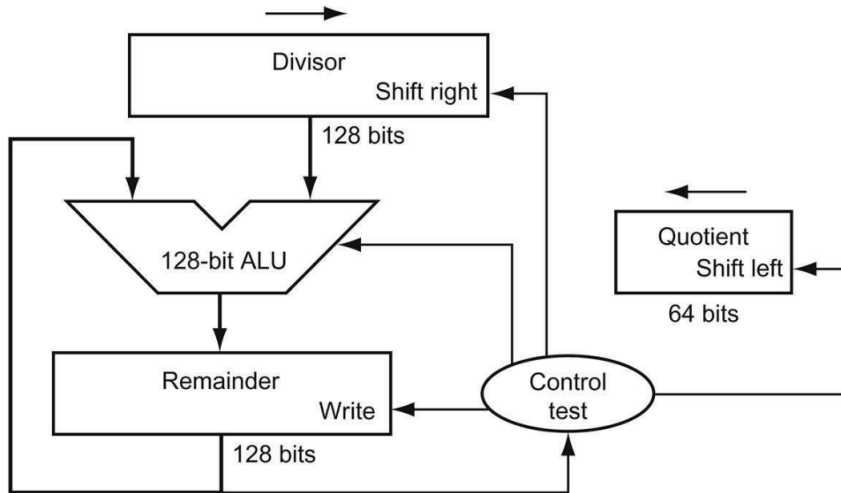


Figure 1: Basic divisor circuit from textbook.

Name: _____

Roll No: _____

Name: _____

Roll No: _____

8. Answer the following questions briefly [**10 marks**].

(a) State True or False, with reasoning: Time taken for a single instruction on a pipelined CPU is always less than or equal to the time taken on a non-pipe lined (identical) CPU. [**2 marks**]

(b) Consider a 5 stage pipeline with IF, ID, EX, MEM, and WB latencies 8, 6, 4, 6, and 4, respectively (in ns). If IF stage is made 50% faster, the percentage by which CPU performance improves is _____. [**3 marks**]

(c) The RISC-V architecture is designed to be small with only a few supported operations. Please explain how adding new instructions could make our processor slower. Now that you have studied processor design, try to give examples where new gate delays might be introduced in the control and data paths. [**3 marks**]

Name: _____

Roll No: _____

(d) Consider the following two C functions that perform the same task:

```
1 int factorial_iter(int n) {
2     int res = 1;
3     int i;
4     for (i = 2; i <= n; i++) {
5         res = res * i;
6     }
7     return res;
8 }
```

```
1 int factorial_recursive(int n) {
2     if (n <= 1) {
3         return 1;
4     }
5     return n * factorial_recursive(n - 1);
6 }
```

When both functions are compiled, each function ends up being around 20 instructions if you include the prologue and the epilogue. Which implementation do you think will take longer to compute factorial(10) and why? [2 marks]