Sample Problems for Midterm Exam
CDA4101

1.4. Assume a color display using 8 bits for each of the primary colors (red, green, blue) per pixel and a frame size of 1280 x 1024.

a) What is the minimum size in bytes of the frame buffer to store a frame?

# of bytes per pixel: 3
# of pixels per frame: $1280 \times 1024 = 1{,}310{,}720$
# of bytes per frame = # of bytes per pixel × # of pixels per frame = $3 \times 1310720 = 3932160\ bytes = 3.75\ MB$
Note: 1MB = $2^{20}\ bytes$

$$1KB = 2^{10} bytes$$
$$1GB = 2^{30} bytes$$

b) How long would it take, at a minimum, for the frame to be sent over a 100 Mbit/s network?

Bandwidth = 100Mbits/sec = 100,000,000 bits/ sec
Time= $\frac{volume}{BW} = \frac{3.75MB}{100{,}000{,}000\ bits/sec} = \frac{31457280}{100{,}000{,}000 bits/sec} = 0.3145728 \text{ sec} = 314.6\ ms$

1.5. Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2.

a) Which processor has the highest performance expressed in instructions per second?

# of Instructions/sec = clock rate (Hz)/CPI
P1: 3B/1.5 = 2 billion Instructions per seconds
P2: 2.5 billion Instructions per second
P3: 4B/2.2 = 1.82 billion Instructions per second

b) If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.

P1: # of cycles: time × clock rate = 10*3GHz = 30 billion cycles.
# of instructions: # of cycles/1.5 = 20 billion
P2: # of cycles = $10 \times 2.5GHz$ = 25 billion cycles
# of instructions: # of cycles/1 = 25 billion instructions
P3: # of cycles = $10 \times 4GHz = 40\ Billion\ cycles$
# of instructions: #of cycles/2.2 = 18.18 Billion instructions.

c) We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

Execution Time = # of instructions * CPI / clock rate = $\frac{N \times CPI}{CR} = T$

$$.7\,T = \frac{N \times 1.2\ CPI}{x\ CR}$$

$.7 \times \frac{N \times CPI}{CR} = \frac{N \times 1.2\ CPI}{x\ CR}$ replacing T by $\frac{N \times CPI}{CR}$

$$x = \frac{1.2}{.7} = 1.71\ means\ we\ need\ 71\%\ increase\ in\ Clock\ Rate$$

New Clock Rate of P1: $1.71 \times 3GHz = 5.13GHz$
New Clock Rate for P2: $1.71 \times 2.5GHz = 4.275\ GHz$
New Clock Rate for P3: $1.71 \times 4GHz = 6.84\ GHz$

1.6. Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (class A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.
Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which implementation is faster?

Instruction count: I= 1 million
# of Instructions of Type A: .1I=100,000
# of Instructions of Type B: .2I=200,000
# of Instructions of Type C: .5I = 500,000
# of Instructions of Type D: .2I = 200,000

P1: 2.5GHz →
# of clock cycles for A Instructions: CPI of A× # of A Instructions: .1 I = 100,000 clock cycles → Time for A instructions = 100,000/2.5B sec
# of clock cycles for B Instructions: CPI of B× # of B Instructions: 2×.2 I = 400,000 clock cycles → Time for B instructions = 400,000/2.5B sec
# of clock cycles for C Instructions: CPI of C× # of C Instructions: 3×.5 I = 1,500,000 clock cycles → Time for C instructions = 1,500,000/2.5B sec
# of clock cycles for D Instructions: CPI of D× # of D Instructions: 3×.2 I = 600,000 clock cycles → Time for D instructions = 600,000/2.5B sec
Total time for P1: $\frac{2.6M}{2.5B} = 1.04\ miliseconds$
Total times for P2: $\frac{1M \times 2}{3B} = .67 millseconds$

a) What is the global CPI for each implementation?
   P1: $global\ CPI$: $\frac{1 \times .1I + 2 \times .2I + 3 \times .5I + 3 \times .2I}{I} = 2.6$
   P2: global CPI = 2
b) Find the clock cycles required in both cases.
   # of clock cycles for P1 = 2.6M
   #of clock cycles for P2 = 2M

2.1. For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables f, g, h, and i are given and could be considered 32-bit integers as declared in a C program. Use a minimal number of MIPS assembly instructions.

f = g + (h − 5);

addi t0, h, -5
add f, g, t0

2.2. For the following MIPS assembly instructions above, what is a corresponding C statement?

```
add  f, g, h
add  f, i, f
```

f = i+(g+h);

2.3. For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

B[8] = A[i − j];

```
sub $t0, $s3, $s4 #t0 = i-j
sll $t0, $t0, 2 #t0 = 4*t0
add $t0, $t0, $s6 # t0 = base + offset in array A… t0 = &A[i-j]
lw  $t1, 0($t0) #t1 = A[i-j]
sw  $t1, 32($s7) # B[8] = A[i-j]
```

2.4. For the MIPS assembly instructions below, what is the corresponding C statement? Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

```
sll  $t0, $s0, 2     # $t0 = f * 4
add  $t0, $s6, $t0   # $t0 = &A[f]
sll  $t1, $s1, 2     # $t1 = g * 4
add  $t1, $s7, $t1   # $t1 = &B[g]
lw   $s0, 0($t0)     # f = A[f]
addi $t2, $t0, 4   # $t2 = &A[f+1]
lw   $t0, 0($t2)   #$t0 = A[f+1]
add  $t0, $t0, $s0 #$t0 = A[f]+A[f+1]
sw   $t0, 0($t1) #  B[g] = A[f]+A[f+1]
```

B[g] = A[f] + A[f+1];

2.7. Show how the value 0xabcdef12 would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0.

Address	Big Endian Byte	Little Endian Byte
0	ab = 1010 1011	12
1	cd	ef
2	ef	cd
3	12	ab

2.8. Translate 0xabcdef12 into decimal.

$2 \times 16^0 + 1 \times 16^1 + 15 \times 16^2 + 14 \times 16^3 + 13 \times 16^4 + 12 \times 16^5 + 11 \times 16^6 + 10 \times 16^7 =$

2.9. Translate the following C code to MIPS. Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of

the arrays A and B are in registers $s6 and $s7, respectively. Assume that the elements of the arrays A and B are 4-byte words:

B[8] = A[i] + A[j];

```
sll $t0,$s3,2     #        $t0 = 4*i, offset        of A[i]
sll $t1,$s4,2     #        $t1 = 4*j, offset        of A[j]
add $t0,$t0,$s6 # address of A[i]
add $t1,$t1,$s6 # address of A[j]
lw $t0,0($t0)    # $t0 = A[i]
lw $t1,0($t1)    # $t1 = A[j]
add $t0,$t1,$t0   # $t0=A[i]+A[j]
addi $t1,$s7,32   # address of B[8]
sw $t0,0($t1)   # B[8]=A[i]+A[j]
```

2.26 Consider the following MIPS loop:

```
LOOP: slt  $t2, $0,  $t1
      beq  $t2, $0,  DONE
      subi $t1, $t1, 1
      addi $s2, $s2, 2
      j    LOOP
DONE:
```

2.26.1. Assume that the register $t1 is initialized to the value 10. What is the value in register $s2 assuming $s2 is initially zero?

B= $2\times$ *number of iterations*+initial value = $2 \times 10 + 0 = 20$

2.26.2. For each of the loops above, write the equivalent C code routine. Assume that the registers $s1, $s2, $t1, and $t2 are integers A, B, i, and temp, respectively.

```
while(i>0){
        i--;
        B += 2;
}
```

2.26.3. For the loops written in MIPS assembly above, assume that the register $t1 is initialized to the value N. How many MIPS instructions are executed?

The first N iterations get completed, each 5 instructions = $5 \times N$

Final iteration is incomplete with 2 instructions.

Total: 5N+2

2.27. Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of a, b, i, and j are in registers $s0, $s1, $t0, and $t1, respectively. Also, assume that register $s2 holds the base address of the array D.

```
for(i = 0; i < a; i++)
```

```
    for(j = 0; j < b; j++)
       D[4 * j] = i + j;
      add   $t0, $0, $0       # i = 0
L1:   slt   $t2, $t0, $s0          # i < a
      beq   $t2, $0, Exit          # $t2 == 0, go to Exit
      add   $t1, $0, $0       # j = 0
L2:   slt   $t2, $t1, $s1          # j < b
      beq   $t2, $0, L3       # if $t2 == 0, go to L3
      add   $t2, $t0, $t1          # i+j
      sll   $t4, $t1, 4       # $t4 = 4*j (sll was written instead of mul.)
      add   $t3, $t4, $s2          # $t3 = &D[4*j]
      sw    $t2, 0($t3)       # D[4*j] = i+j
      addi  $t1, $t1, 1       # j = j+1
      j     L2
L3:   addi  $t0, $t0, 1       # i = i+1
      j     L1
Exit:
```

2.31. Implement the following C code in MIPS assembly. What is the total number of MIPS instructions needed to execute the function?

```
int fib(int n){
  if (n == 0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return fib(n − 1) + fib(n − 2);
}
```

We assume $a0 is non-negative.

```
fib:
      addi  $sp, $sp, -12           # allocate stack frame of 12 bytes
      sw    $a0, 8($sp)       # save n
      sw    $ra, 4($sp)       # save return address
      sw    $s0, 0($sp)       # save $s0

      slti  $t0, $a0, 2       # fib(i) = i for i = 0, 1
      beq   $t0, $0, else
      add   $v0, $a0, $0            # $v0 = 0 or 1
      j     exit              # go to exit
else:
      addi  $a0, $a0, -1            # fib(n-1)
      jal   fib               # recursive call
      add   $s0, $v0, $0
      addi  $a0, $a0, -1            # fib(n-2)
      jal   fib               # recursive call
      add   $v0, $v0, $s0
exit:
      lw    $a0, 8($sp)       # restore $a0
      lw    $ra, 4($sp)       # restore return address
      lw    $s0, 0($sp)       # restore $s0
      addi  $sp, $sp, 12            # free stack frame
      jr    $ra               # return to caller
```

Total number of MIPS instructions for calculating fib(n) is T(n):
T(0) =T(1) = 13. If n>1, T(n) = 17 + T(n-1)+T(n-2) -> T(n) ~ 1.6^n

3.1. What is 5ED4–07A4 when these values represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

	5ED4		0101 1110 1101 0100
-	07A4	-	0000 0111 1010 0100
	5730		0101 0111 0011 0000

3.2. What is 5ED4–07A4 when these values represent signed 16-bit hexadecimal numbers stored in sign-magnitude format? The result should be written in hexadecimal. Show your work.

	5ED4		0101 1110 1101 0100
-	07A4	+	1111 1000 0101 1100
	5730		0101 0111 0011 0000

3.4. What is 4365–3412 when these values represent unsigned 12-bit octal numbers? The result should be written in octal. Show your work.

	4365		100 011 110 101
-	3412	-	011 100 001 010
	0753		000 111 101 011

3.8. Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate 185–122. Is there overflow, underflow, or neither?

	185		1011 1001 (-71)
-	122	+	1000 0110
-	193		

Since signed 8-bit integers range is -128~127, there is underflow.

3.9. [10] <§3.2> Assume 151 and 214 are signed 8-bit decimal integers stored in two's complement format. Calculate 151+214 using saturating arithmetic. The result should be written in decimal. Show your work.

	151		0110 1001	(105)
+	214	+	0010 1010	(42)
	127		0111 1111	

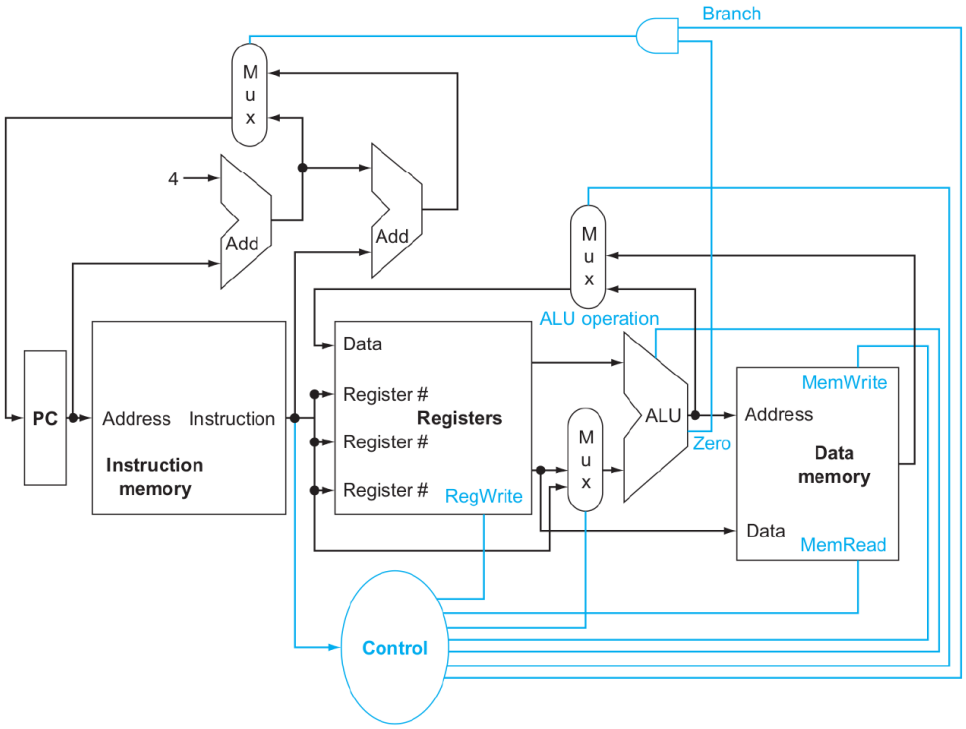
Since signed 8-bit integers range is -128~127, the result using saturating arithmetic is 127, not 365.

3.23. Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 single precision format.

$63.25 = 253 \times 2^{-2} = 1.1111101 \times 2^{5=132-127}$ Sign = 0, exponent = 132, Significand = 253×2^{16}

0 10000100 11111010000000000000000

4.2 The basic single-cycle MIPS implementation in COD Figure 4.2 (The basic implementation of the MIPS subset …) can only implement some instructions. New instructions can be added to an existing Instruction Set Architecture (ISA), but the decision whether or not to do that depends, among other things, on the cost and complexity the proposed addition introduces into the processor datapath and control. The first three problems in this exercise refer to the new instruction:
Instruction: LWI Rt, Rd(Rs)
Interpretation: Reg[Rt] = Mem[Reg[Rd] + Reg[Rs]]



4.2.1. Which existing blocks (if any) can be used for this instruction?
Instruction memory, one register read ports, the path that passed the immediate to the ALU, and the register write port.
4.2.2. Which new functional blocks (if any) do we need for this instruction?
We need to extend the existing ALU to also do shifts (SLL, to extend the offset to 32bit value).
4.2.3. What new signals do we need (if any) from the control unit to support this instruction?
We need to change the ALU operation control signals to support the SLL operation in the ALU.

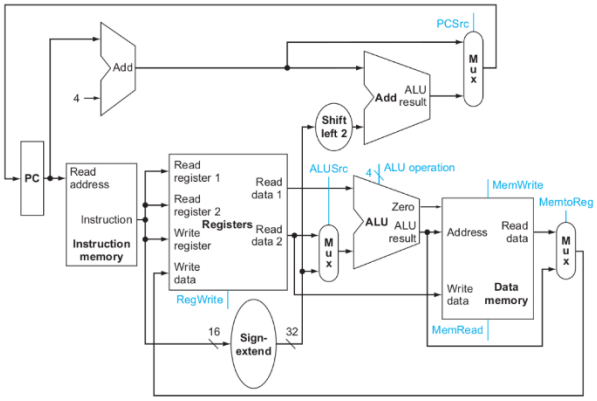
4.4 Problems in this exercise assume that logic blocks needed to implement a processor's datapath have the following latencies:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps

4.4.1. If the only thing we need to do in a processor is fetch consecutive instructions (COD Figure 4.6 (A portion of the datapath used for fetching instructions and incrementing the program counter)), what would the cycle time be?

200ps (I-Mem and Add are in parallel paths, so the maximum of 200 and 70 will be the length of cycle)

4.4.2. Consider a datapath similar to the one in COD Figure 4.11 (The simple datapath for the core MIPS architecture …), but for a processor that only has one type of instruction: unconditional PC-relative branch. What would the cycle time be for this datapath?



Critical path include: Instruction memory, Sign-extend, Shift left 2, Add and Mux.

The cycle time will be

$$\text{CycleTime} = 200 + 15 + 10 + 70 + 20 = 315\text{ps}$$

4.4.3. Repeat 4.4.2, but this time we need to support only conditional PC-relative branches.

For the PC-relative conditional branch, there are two sub-datapath to finish the instruction before entering the final MUX. (1) IM->Sign-ext-> Shift left 2-> ADD and (2) IM->Register File->MUX->ALU. Then,

$$\text{Path1} = 200 + 15 + 10 + 70 = 295\text{ps}$$

$$\text{Path2} = 200 + 90 + 20 + 90 = 400\text{ps} > \text{Path1}$$

Thus, the cycle time is determined by the longest path, CycleTime = Path2 +MUX = 420ps.

4.8 In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

Also, assume that instructions executed by the processor are broken down as follows:

alu	beq	lw	sw
45%	20%	20%	15%

4.8.1. What is the clock cycle time in a pipelined and non-pipelined processor?

Non-pipelined: 1250ps; Pipelined: 350ps.

4.8.2. What is the total latency of an LW instruction in a pipelined and non-pipelined processor?

Non-pipelined: 1250ps; Pipelined: 1750ps.

4.8.3. If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

Split ID stage. New clock cycle will be 300ps.

4.8.4. Assuming there are no stalls or hazards, what is the utilization of the data memory?

lw+sw = 35%

4.8.5. Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?

alu+lw = 65%

Extra questions: Explain two ways to resolve data hazard in pipelining. Compare them. What are the pros/cons for each one.