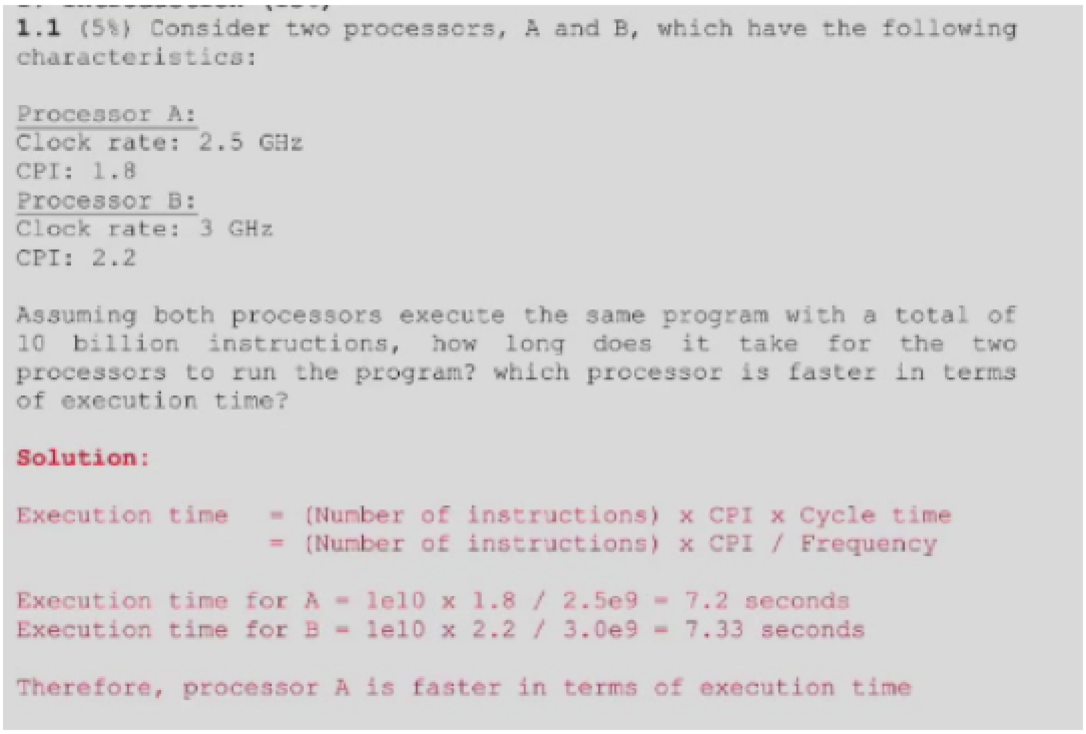


1.1 (5%) Consider two processors, A and B, which have the following characteristics:

Processor A:
Clock rate: 2.5 GHz
CPI: 1.8
Processor B:
Clock rate: 3 GHz
CPI: 2.2

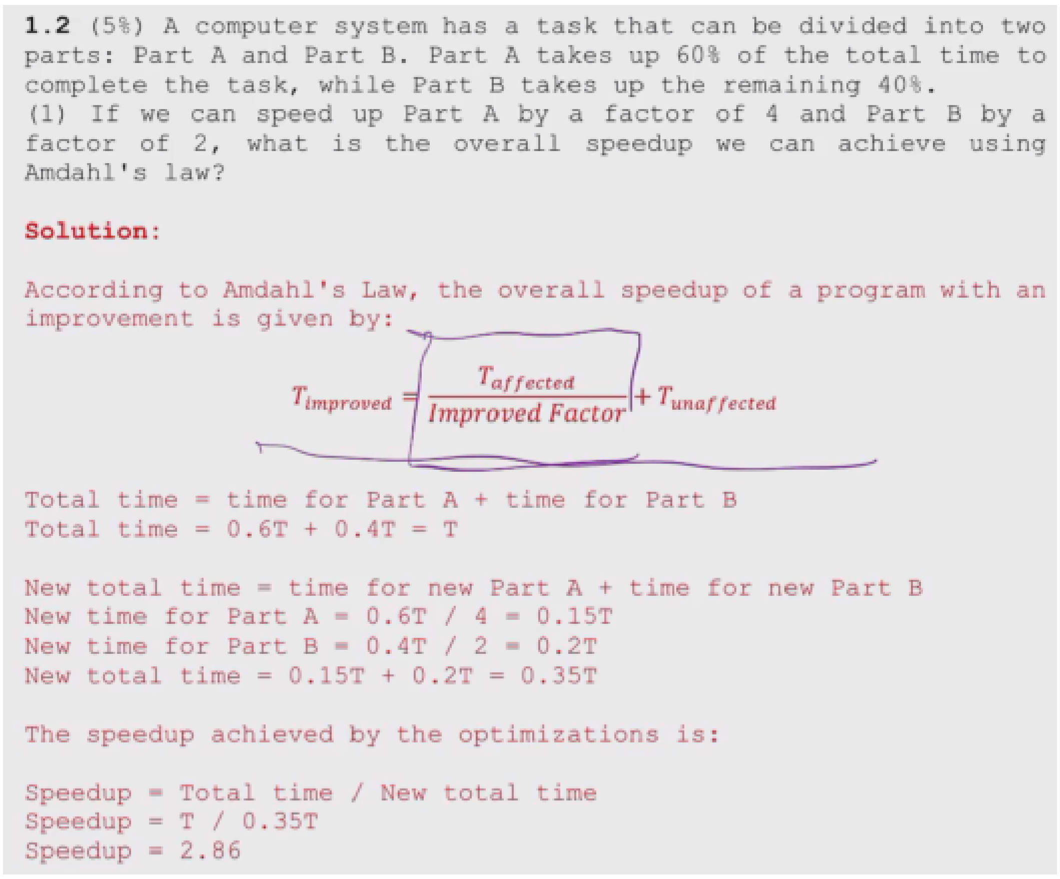
Assuming both processors execute the same program with a total of 10 billion instructions, how long does it take for the two processors to run the program? which processor is faster in terms of execution time?

Solution:

```
Execution time   = (Number of instructions) x CPI x Cycle time
                 = (Number of instructions) x CPI / Frequency

Execution time for A = 1e10 x 1.8 / 2.5e9 = 7.2 seconds
Execution time for B = 1e10 x 2.2 / 3.0e9 = 7.33 seconds
```

Therefore, processor A is faster in terms of execution time



1.2 (5%) A computer system has a task that can be divided into two parts: Part A and Part B. Part A takes up 60% of the total time to complete the task, while Part B takes up the remaining 40%.
(1) If we can speed up Part A by a factor of 4 and Part B by a factor of 2, what is the overall speedup we can achieve using Amdahl's law?

Solution:

According to Amdahl's Law, the overall speedup of a program with an improvement is given by:

$$\underline{T_{improved} = \boxed{\frac{T_{affected}}{Improved\ Factor}} + T_{unaffected}}$$

```
Total time = time for Part A + time for Part B
Total time = 0.6T + 0.4T = T

New total time = time for new Part A + time for new Part B
New time for Part A = 0.6T / 4 = 0.15T
New time for Part B = 0.4T / 2 = 0.2T
New total time = 0.15T + 0.2T = 0.35T
```

The speedup achieved by the optimizations is:

```
Speedup = Total time / New total time
Speedup = T / 0.35T
Speedup = 2.86
```

1.3 (5%) We have eight great ideas in computer design, match them to the following **seven** examples in real world (hint: one example gets two idea).

Design for Moore's Law, Use Abstraction to Simplify Design, Make the Common Case Fast, Performance via Parallelism, Performance via Pipelining, Performance via Prediction, Hierarchy of Memories, Dependability via Redundancy

1. Cooking recipes that only specify the type of sugars, oils and flours without naming the detailed brand.
   Abstraction
2. Apply for multiple waitlist when you book your train tickets on 12306.
   Redundancy
3. Organize your clothes and pack them in storage boxes while only leave your favorite ones in your closet.
   Hierarchy, common case
4. Dividing a team of workers into different groups, each team work independently on their own tasks.
   Parallel
5. Dividing a team of workers into different groups, each team is in charge of a specific task and work in series
   Pipeline
6. A supermarket manager decided to open five checkout lanes at 6pm, and 2 lanes at 9pm.
   Prediction
7. Ultralight fiber materials which can make large size backpacks with reasonable weight.
   Design for Moore's Law

2. Arithmetic for Computer (35%)

2.1 (4%) What is the octal value of 0x20230508.

A:________________________________

2.2 (4%) What is the binary value of -503 (using 2-s complement).

A:________________________________

2.3 (4%) What is the binary bit pattern representation of -11.53125 (using single precision).

A:________________________________

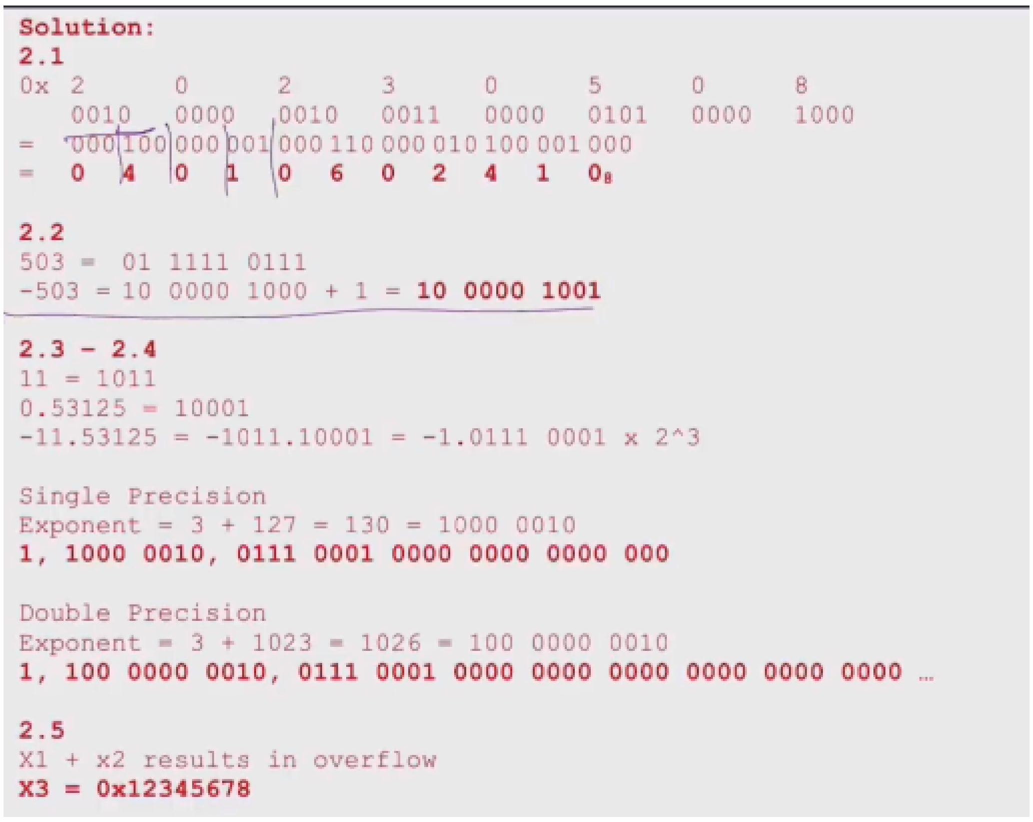
2.4 (4%) Double precision representation of -11.53125

A:________________________________

2.5 (4%) Assume that registers in RV32I (Risc-V 32-bit) hold the following values:
x1: 0x7FFFFFFF
x2: 0x00000001
x3: 0x12345678
What is the value of x3 immediate after the execution of the following assembly code?

```
add x3,x1,x2
```

A:________________________________



Solution:

2.1

```
0x  2       0       2       3       0       5       0       8
    0010    0000    0010    0011    0000    0101    0000    1000
=   000 100 000 001 000 110 000 010 100 001 000
=    0   4   0   1   0   6   0   2   4   1   0
```
$= 0\ 4\ 0\ 1\ 0\ 6\ 0\ 2\ 4\ 1\ 0_8$

2.2

503 = 01 1111 0111

-503 = 10 0000 1000 + 1 = **10 0000 1001**

2.3 – 2.4

11 = 1011

0.53125 = 10001

-11.53125 = -1011.10001 = -1.0111 0001 x 2^3

Single Precision

Exponent = 3 + 127 = 130 = 1000 0010

1, 1000 0010, 0111 0001 0000 0000 0000 000

Double Precision

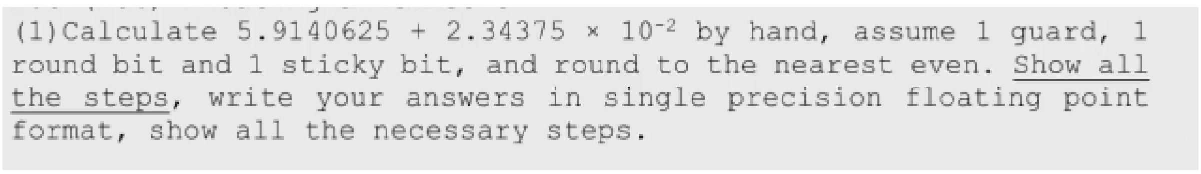
Exponent = 3 + 1023 = 1026 = 100 0000 0010

1, 100 0000 0010, 0111 0001 0000 0000 0000 0000 0000 0000 …

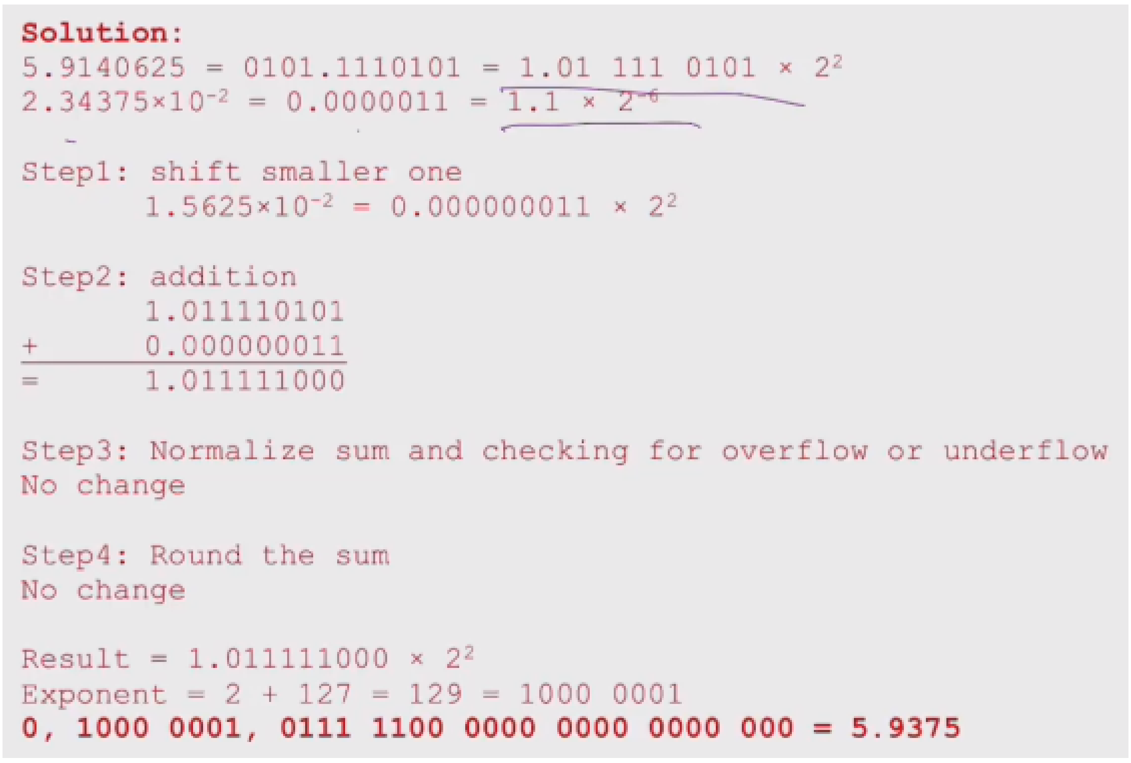
2.5

X1 + x2 results in overflow

X3 = 0x12345678



(1)Calculate 5.9140625 + 2.34375 × 10^{-2} by hand, assume 1 guard, 1 round bit and 1 sticky bit, and round to the nearest even. Show all the steps, write your answers in single precision floating point format, show all the necessary steps.



Solution:

5.9140625 = 0101.1110101 = 1.01 111 0101 × 2^2

2.34375×10^{-2} = 0.0000011 = 1.1 × 2^{-6}

Step1: shift smaller one

1.5625×10^{-2} = 0.000000011 × 2^2

Step2: addition

```
        1.011110101
+       0.000000011
=       1.011111000
```

Step3: Normalize sum and checking for overflow or underflow

No change

Step4: Round the sum

No change

Result = 1.011111000 × 2^2

Exponent = 2 + 127 = 129 = 1000 0001

0, 1000 0001, 0111 1100 0000 0000 0000 000 = 5.9375

3. Instruction (30%)

3.1 (6%) Translate the following two RISC-V assembly code into the binary and the hex:

(1) sw x9, 120(x10)

A:___________________________________

0000011 01001 01010 010 11000 0100011, 0x06952C23

(2) beq x7,x8,100

A:___________________________________

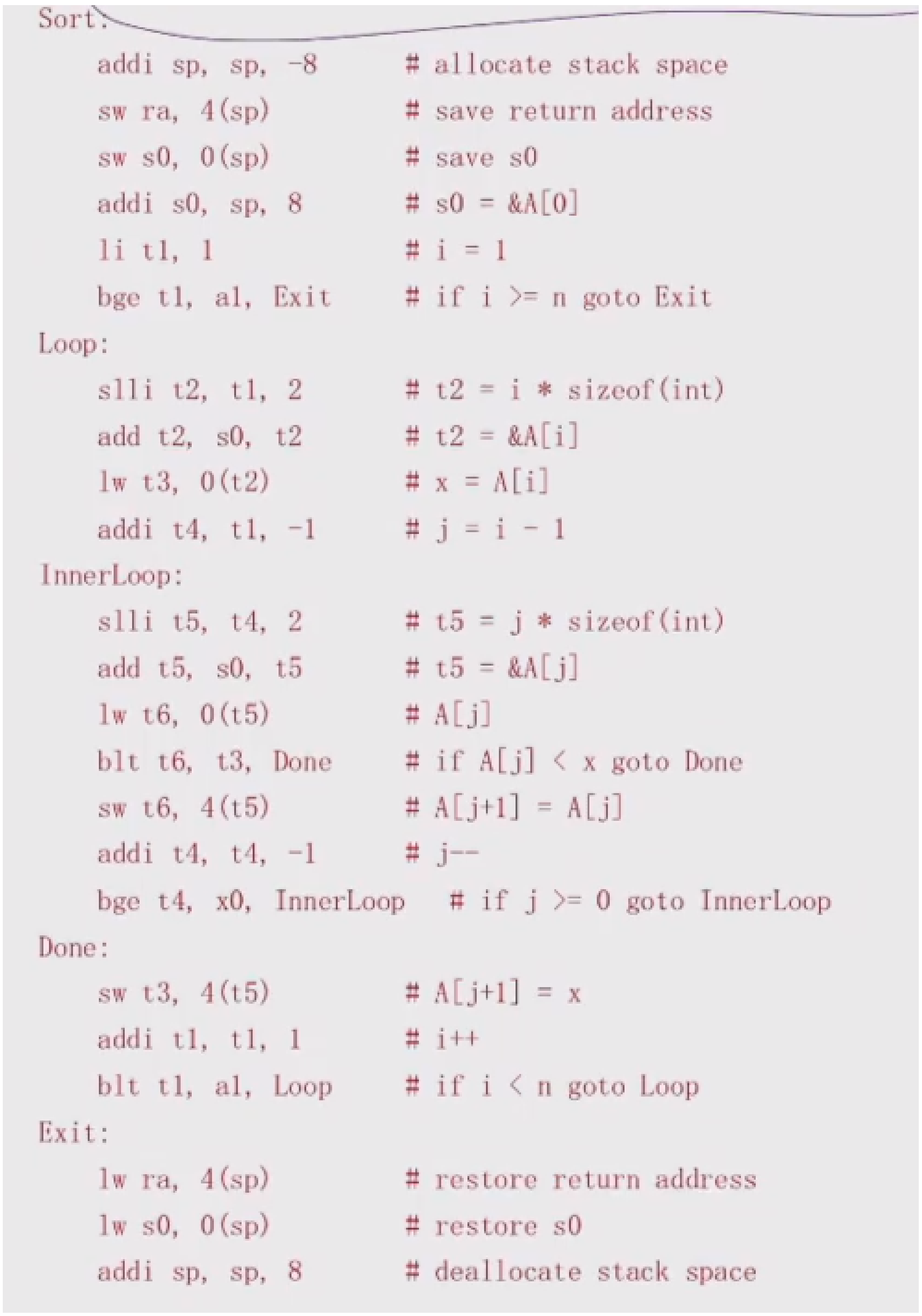
0000011 01000 00111 000 00100 1100111(or 1100011), 0x6838267 (or 0x6838263)

3.2 (4%) What is the assembly instruction of the following machine code 0x10D62C23.

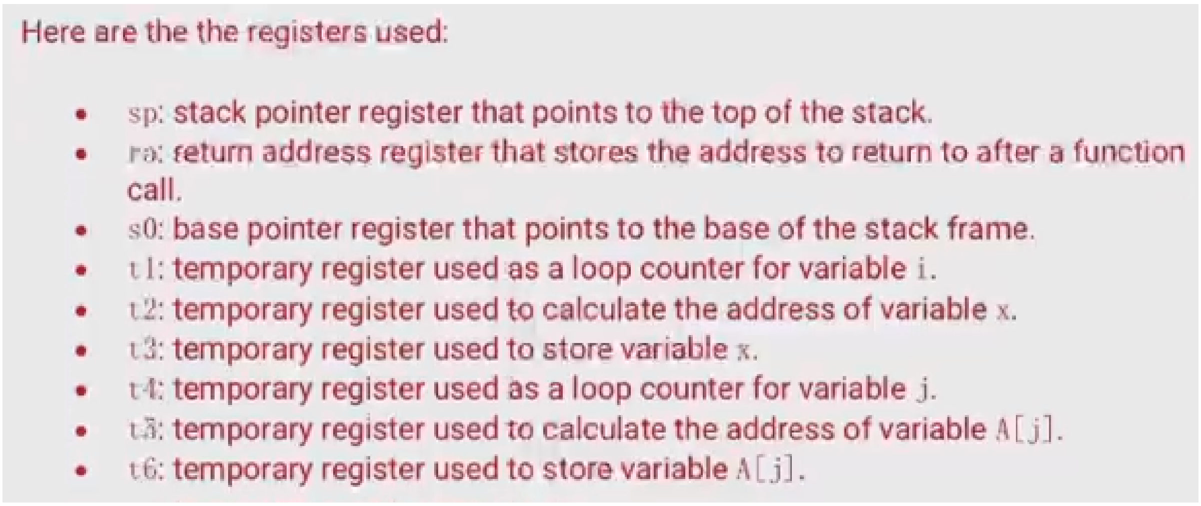
A:___________________________________

0001000 01101(x13) 01100(x12) 010(sw) 11000 0100011(s), imm = 280
sw x13,280(x12)

```
void Sort(int A[], int n)
{
    for(int i = 1; i<n; i++)
    {
        if(A[i] < A[i-1])
        {
            int j = i-1;
            int x = A[i];
            while(j > -1 && x < A[j])
            {
                A[j+1] = A[j];
                j -= 1;
            }
            A[j+1] = x;
        }
    }
}
```



```
Sort:
    addi sp, sp, -8        # allocate stack space
    sw ra, 4(sp)           # save return address
    sw s0, 0(sp)           # save s0
    addi s0, sp, 8         # s0 = &A[0]
    li t1, 1               # i = 1
    bge t1, a1, Exit       # if i >= n goto Exit
Loop:
    slli t2, t1, 2         # t2 = i * sizeof(int)
    add t2, s0, t2         # t2 = &A[i]
    lw t3, 0(t2)           # x = A[i]
    addi t4, t1, -1        # j = i - 1
InnerLoop:
    slli t5, t4, 2         # t5 = j * sizeof(int)
    add t5, s0, t5         # t5 = &A[j]
    lw t6, 0(t5)           # A[j]
    blt t6, t3, Done       # if A[j] < x goto Done
    sw t6, 4(t5)           # A[j+1] = A[j]
    addi t4, t4, -1        # j--
    bge t4, x0, InnerLoop   # if j >= 0 goto InnerLoop
Done:
    sw t3, 4(t5)           # A[j+1] = x
    addi t1, t1, 1         # i++
    blt t1, a1, Loop       # if i < n goto Loop
Exit:
    lw ra, 4(sp)           # restore return address
    lw s0, 0(sp)           # restore s0
    addi sp, sp, 8         # deallocate stack space
```

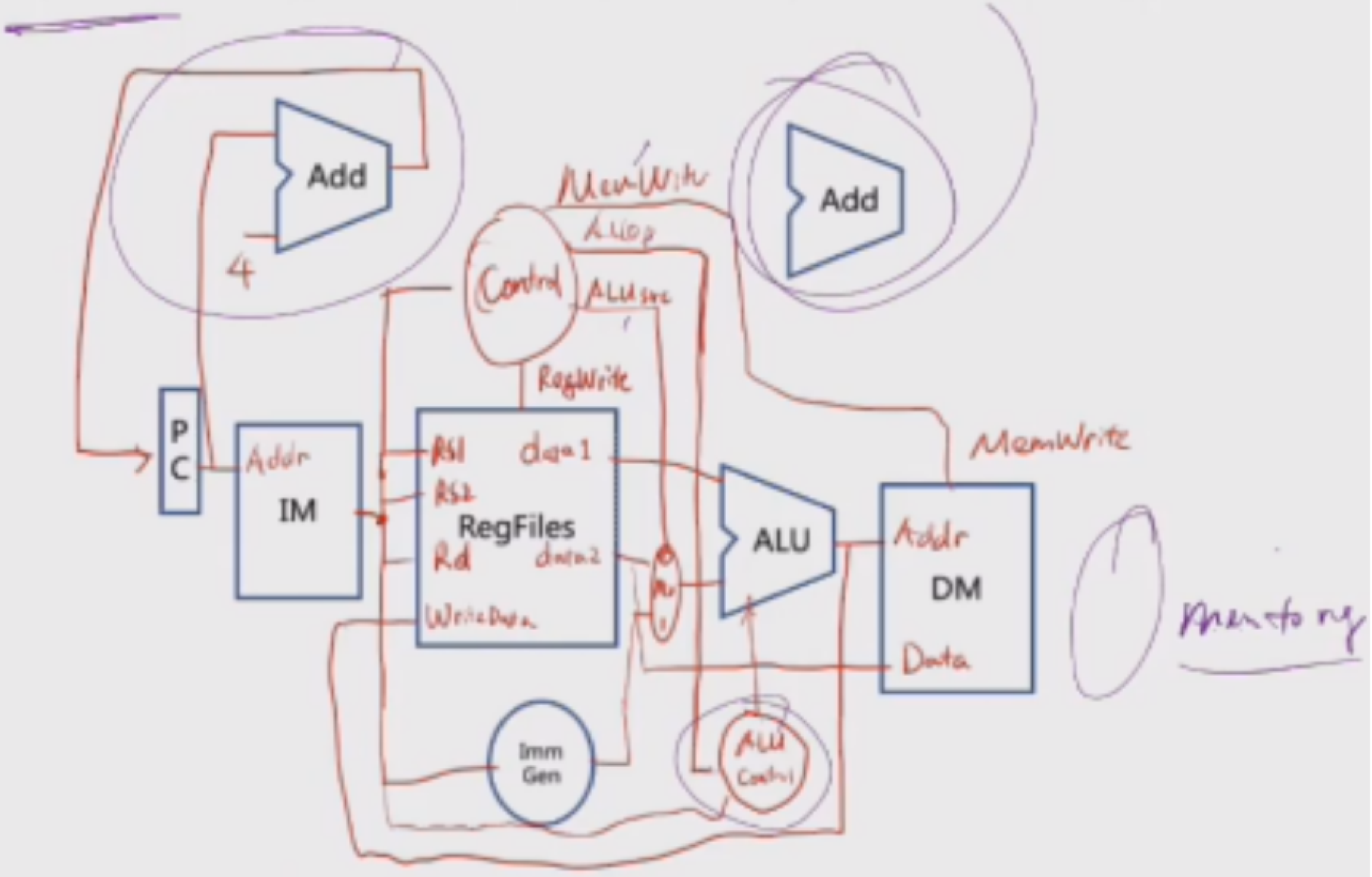


Here are the the registers used:

- sp: stack pointer register that points to the top of the stack.
- ra: return address register that stores the address to return to after a function call.
- s0: base pointer register that points to the base of the stack frame.
- t1: temporary register used as a loop counter for variable i.
- t2: temporary register used to calculate the address of variable x.
- t3: temporary register used to store variable x.
- t4: temporary register used as a loop counter for variable j.
- t5: temporary register used to calculate the address of variable A[j].
- t6: temporary register used to store variable A[j].

4. Single cycle processor (20%)

4.1 (10%) Suppose you are designing a single cycle processor that only support R-type and S-type instructions, draw the most compact processor design below, including both datapath and ~~control logic~~



4.2 (5%) How many control signals are decoded from the instruction? what their va~~lues for~~ instruction add, x1,x2,x3?

```
RegWrite, ALUsrc, ALUop, MemWrite
1         0       'add'     0
```