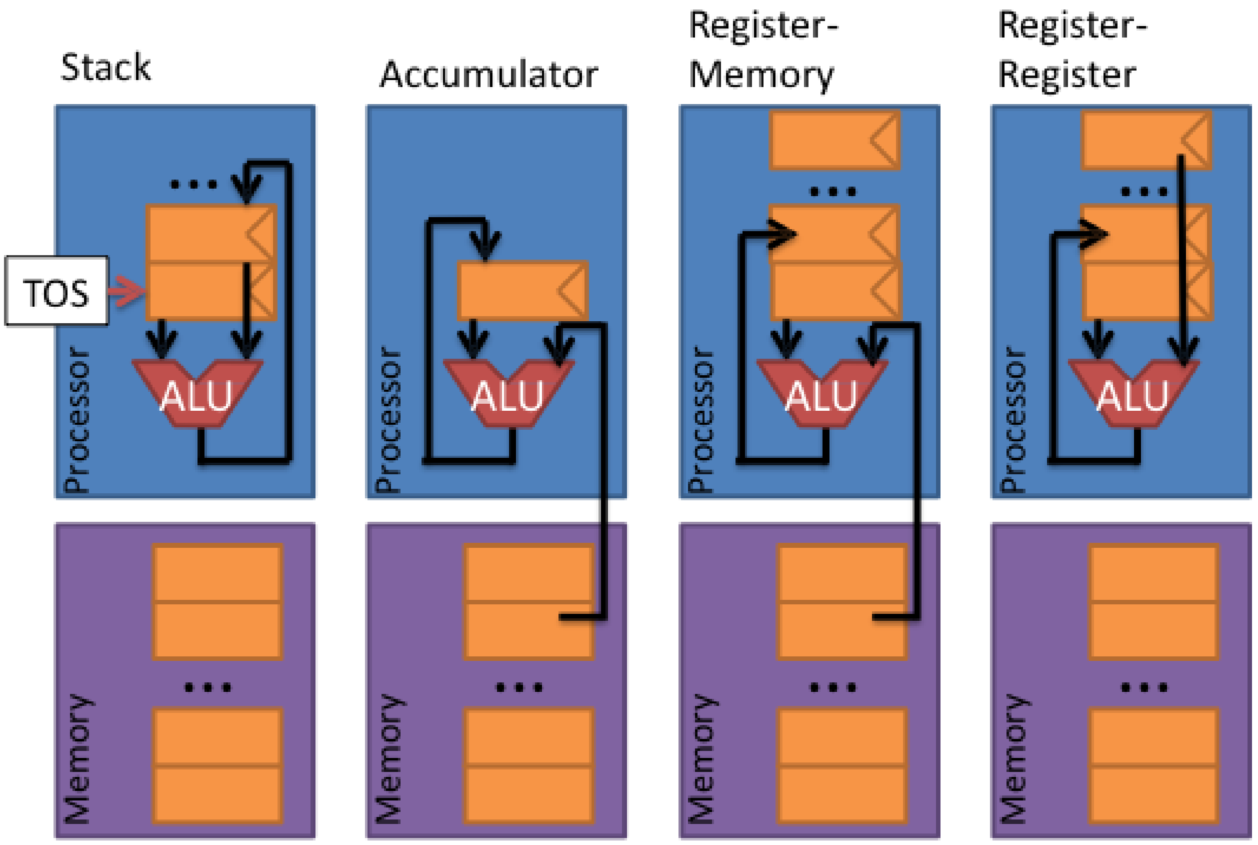
(a) [1.2 pts] This problem investigates how the same code is compiled into different instructions for different processors with different instruction set architectures.



Code to compute C = A + B in each instruction set architecture

Stack	Accumulator	Register-Memory	Register-Register
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R3, R1, B	Load R2, B
Add	Store C	Store R3, C	Add R3, R1, R2
Pop C			Store R3, C

For this problem, assume that the values A, B, C, D, E, and F reside in memory. Also assume that instruction opcodes are represented by 4 bits, memory addresses are 32 bits wide, and that the processor has 32 general purpose registers.

For each instruction set architecture shown above (Stack, Accumulator, Register-Memory, Register-Register), what is the total code size (in bits) for a code that computes C = A + B? Assume that the instruction sets use variable width instructions.

Stack:

```
Push A        // opcode + memory address = 4 + 32 = 36 bits
Push B        // opcode + memory address = 4 + 32 = 36 bits
Add           // opcode = 4 bits
Pop C         // opcode + memory address = 4 + 32 = 36 bits
```

Total code size = 36*3 + 4 = 112 bits

Accumulator:

```
Load A        // opcode + memory address = 4 + 32 = 36 bits
Add B         // opcode + memory address = 4 + 32 = 36 bits
Store C       // opcode + memory address = 4 + 32 = 36 bits
```

Total code size = 36*3 = 108 bits

Register-Memory:

```
Load R1, A         // opcode + register id + memory address = 4 + 5 + 32 = 41 bits
Add R3, R1, B      // opcode + 2x reg id + memory address = 4 + 2*5 + 32 = 46 bits
Store R3, C        // opcode + register id + memory address = 4 + 5 + 32 = 41 bits
```

Total code size = 41*2 + 46 = 128 bits

Register-Register:

```
Load R1, A         // opcode + register id + memory address = 4 + 5 + 32 = 41 bits
Load R2, B         // opcode + register id + memory address = 4 + 5 + 32 = 41 bits
Add R3, R1, R2     // opcode + 3x register id = 4 + 3*5 = 19 bits
Store R3, C        // opcode + register id + memory address = 4 + 5 + 32 = 41 bits
```

Total code size = 41*3 + 19 = 142 bits

(b)[1.5 pts] Consider a processor with a 4-way set-associative cache with one-word cache blocks and a total cache size of 16 words. The cache uses a least recently used (LRU) replacement policy and is initially empty.

The following sequence of decimal word address references is seen by the cache:
2, 86, 53, 61, 29, 37, 28, 2, 45, 20, 6, 22, 14, 6, 53, 29, 78, 4, 22, 70, 61, 54, 78, 45, 2, 61, 37, 45, 6, 29, 28

(i) Indicate whether each address reference is a hit or a miss.

Address	Set	Hit/Miss
2	2	M
86	2	M
53	1	M
61	1	M
29	1	M
37	1	M
28	0	M
2	2	H
45	1	M
20	0	M
6	2	M
22	2	M
14	2	M
6	2	H
53	1	M
29	1	H
78	2	M
4	0	M
22	2	H
70	2	M
61	1	M
54	2	M
78	2	H
45	1	H
2	2	M
61	1	H
37	1	M
45	1	H
6	2	M
29	1	H
28	0	H

(ii) Show the final cache contents.

Set	Contents
0	20
	4
	28
1	61
	37
	45
	29
2	54
	78
	2
	6
3	

(c) [0.9 pts] Consider the following short program executing on a simple 5-stage in-order pipeline (Fetch, Decode, Execute, Memory, Writeback). For arithmetic instructions, the destination register is listed first, followed by the source registers. For example, ADD R3, R2, R1 adds the contents of R1 and R2 and stores the result in R3.

```
I1: LW   R1, 0(R2)       ;load R1 from address 0+R2
I2: LW   R3, 0(R4)       ;load R3 from address 0+R4
I3: ADD  R5, R1, R1      ;R5 = R1 + R1
I4: SUB  R6, R7, R8      ;R6 = R7 - R8
I5: SW   R6, 4(R2)       ;store R6 to address 4+R2
```

Assume a pipeline that does not implement forwarding. Insert NOP instructions in the instruction schedule to avoid hazards. How many cycles does it take to execute the modified code (with NOPs inserted)? Assume that a register read can take place in the same cycle that a value is written back to the register file.

I1	F	D	X	M	W							
I2		F	D	X	M	W						
NOP			NOP									
I3				F	D	X	M	W				
I4					F	D	X	M	W			
NOP						NOP						
NOP							NOP					
I5								F	D	X	M	W
Cycle	1	2	3	4	5	6	7	8	9	10	11	12

It takes 12 cycles to execute the code with NOPs inserted.

(d) [0.4 pts] The MIPS instruction set provides 32 general purpose registers and 32 floating point registers. Describe one reason why adding more registers to an instruction set can be beneficial and one reason why adding more registers can be detrimental.

Reasons to increase the number of registers include:
1. Greater freedom to employ compilation techniques that consume registers, such as loop unrolling, common subexpression elimination, and avoiding name dependences.
2. More locations that can hold values to pass to subroutines.
3. Reduced need to store and re-load values.

Reasons not to increase the number of registers include:
1. More bits needed to represent a register name, thus increasing the overall size of an instruction or reducing the size of some other field(s) in the instruction.
2. More CPU state to save in the event of an exception.
3. Increased chip area and increased power consumption.