1. Basics of CPU instruction set

   (a) (5 points) Write a simple assembly program that computes factorial of N, i.e., the product of all positive integers less than or equal to N. You can use any instruciton set you know (e.g., x86 or RISC), you don't have to be precise (the sketch of the code will work)

```
~~sum~~ fac = 1
for(i=1; i≤N; i++)
        fac = fac*i;

   1·2·3·4·5· ... ·N
```

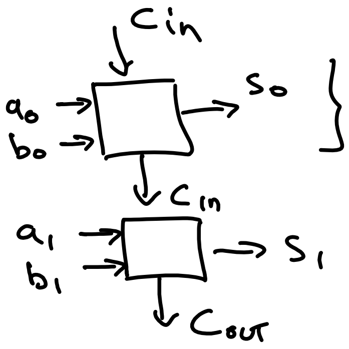
```
// EAX = fac ; EBX = i ; ECX = N

LOOP:   MUL  EBX
        INC  EBX
        CMP  EBX,ECX
        JLE  LOOP
```

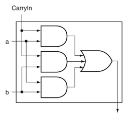
2. Basics of digital design

   (a) (5 points) Design a logical circuit that adds two two-bit numbers. Make sure that an additional output signals the overflow.

$a = \boxed{a_1 \mid a_0}$

$b = \boxed{b_1 \mid b_0}$



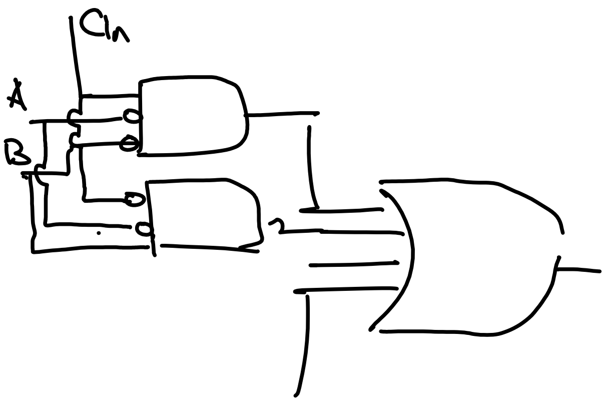


Source: H&P textbook

$$\text{Sum} = \text{Cin} \,.\, \overline{A} \,.\, \overline{B} + B \,.\, \overline{\text{Cin}} \,.\, \overline{A} + A \,.\, \overline{\text{Cin}} \,.\, \overline{B} + A \,.\, B \,.\, \text{Cin}$$

Truth Table for the above operations:

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



3. Pipelines and stalls

(a) (5 points) Consider a 32-bit in-order pipeline with full bypassing that has the following stages:

P.O.P.

Fetch	Decode/Regread	IntALU	Writeback		
		IntALU	Datamem	Datamem	Writeback
		FPALU1	FPALU2	Writeback	

P.O.C   P.O.P.   P.O.C   OFPALU1

After decode, Int-adds go through the stages labeled "IntALU" and "Writeback"; loads/stores go through the stages labeled "IntALU", "Datamem", "Datamem", and "Writeback", while FP-adds go through the stages labeled "FPALU1", "FPALU2", and "Writeback".

How many stall cycles are introduced between the following pairs of successive instructions (remember, the processor implements full bypassing)?

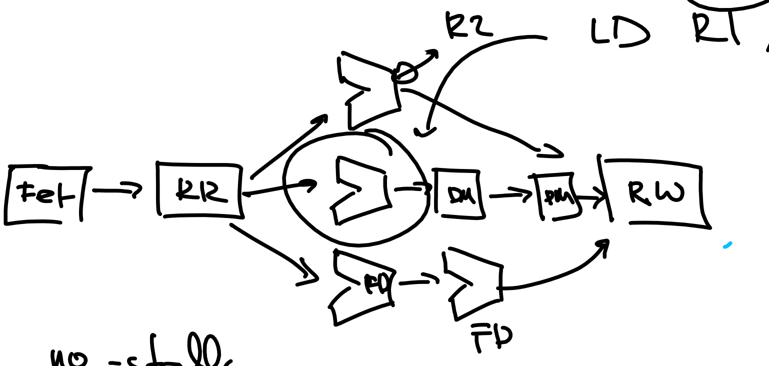
(i) Int-add, providing the address for a load/store,

P.O.P.

I1	Fet	D/RR	IntALU	
I2		Fetch	D/RR	IntALU ...

P.O.C

ADD R2, R3, R4
LD R1, 8(R2)



no-stalls

(ii) Load, providing the data for an FP-add

2-stalls

I1	Fetch	Decode/Regread	IntALU	Writeback			
			IntALU	Datamem	Datamem	Writeback	
			FPALU1	FPALU2	Writeback		
I2		Fet	D/RR	∅	∅	FPU ALU1	
		Fet	D/RR	∅	IntALU	Datamem	Dat

P.O.P   P.O.C.

(iii) Load, providing the data for a store,

1-cycle

(iv) FP-add, followed by dependent FP-add.

1-cycle

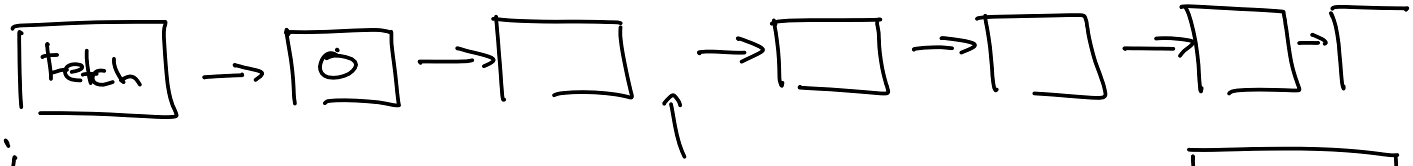
4. Branch delay slot

Consider a 7-stage in-order processor, where the instruction is fetched in the first stage, and the branch outcome is known after 3 stages. Estimate the CPI (cycles per instruction) of the processor under the following scenarios (assume that all stalls in the processor are branch-related and branches account for 20% of all executed instructions, assume that all branches are taken 60% of the time and not-taken 40% of the time).

60% -

40% - no-bubbles

→ (a) (5 points) On every branch, fetch is stalled until the branch outcome is known.



i

I1 . . .

I2 . . .

I3 . . .

I4 . . .

I5 ∅ (branch)

I6

I7

→ I_1 $i+7$

→ I_2 $i+S$

→ I_3 $i+9$

→ I_4 $i+10$

→ ∅ $i+11$

→ ∅ $i+12$

→ I_5 $i+13$

$$\boxed{\frac{7 \text{ cycles}}{5 \text{ instr}}}$$

4

1.4

(b) (5 points) Every branch is predicted not-taken and the mis-fetched instructions are squashed if the branch is taken.

I1

I2

I3

I4

I5 □ □

$\tilde{I6}$

$\tilde{I7}$

I6

I7

∅

∅

branch = 0.2 × 0.6 =

= 0.12

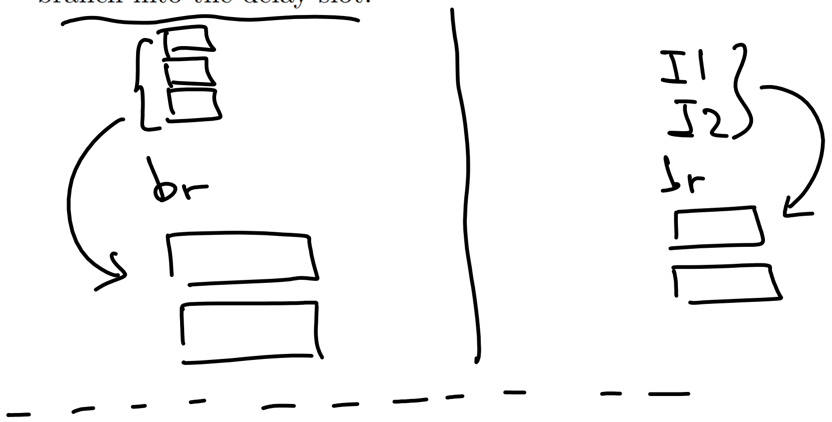
CPI = 1 + 0.12 × 2 = 1.24

$$\boxed{CPI = 1 + P \cdot 2}$$

$$\boxed{\frac{100 + 12 \times 2}{100} = \frac{124}{100} = 1.24}$$

$\frac{1}{5}$ | $\frac{12}{100}$

(c) (5 points) The processor has two delay slots and the two instructions following the branch are always fetched and executed, and You are able to move three instructions before the branch into the delay slot.



CPI = 1

5. Loop unrolling

Consider a basic in-order pipeline with bypassing (one instruction in each pipeline stage in any cycle). The pipeline has been extended to handle FP add and FP mult. Assume the following delays between dependent instructions:

- Load feeding any instruction: 3 stall cycles
- FP ALU feeding any instruction (except stores): 4 stall cycles
- FP MULT feeding any instruction (except stores): 7 stall cycles
- FP ALU feeding store 3 stall cycles
- Int add feeding any instruction: 0 stall cycles
- A conditional branch has 1 delay slot (an instruction is fetched in the cycle after the branch without knowing the outcome of the branch and is executed to completion)

Below is the source code and default assembly code for a loop.

Source Code:

```
for (i=1000; i>0; i--) {
    x[i] = y[i] + z[i];
}
```

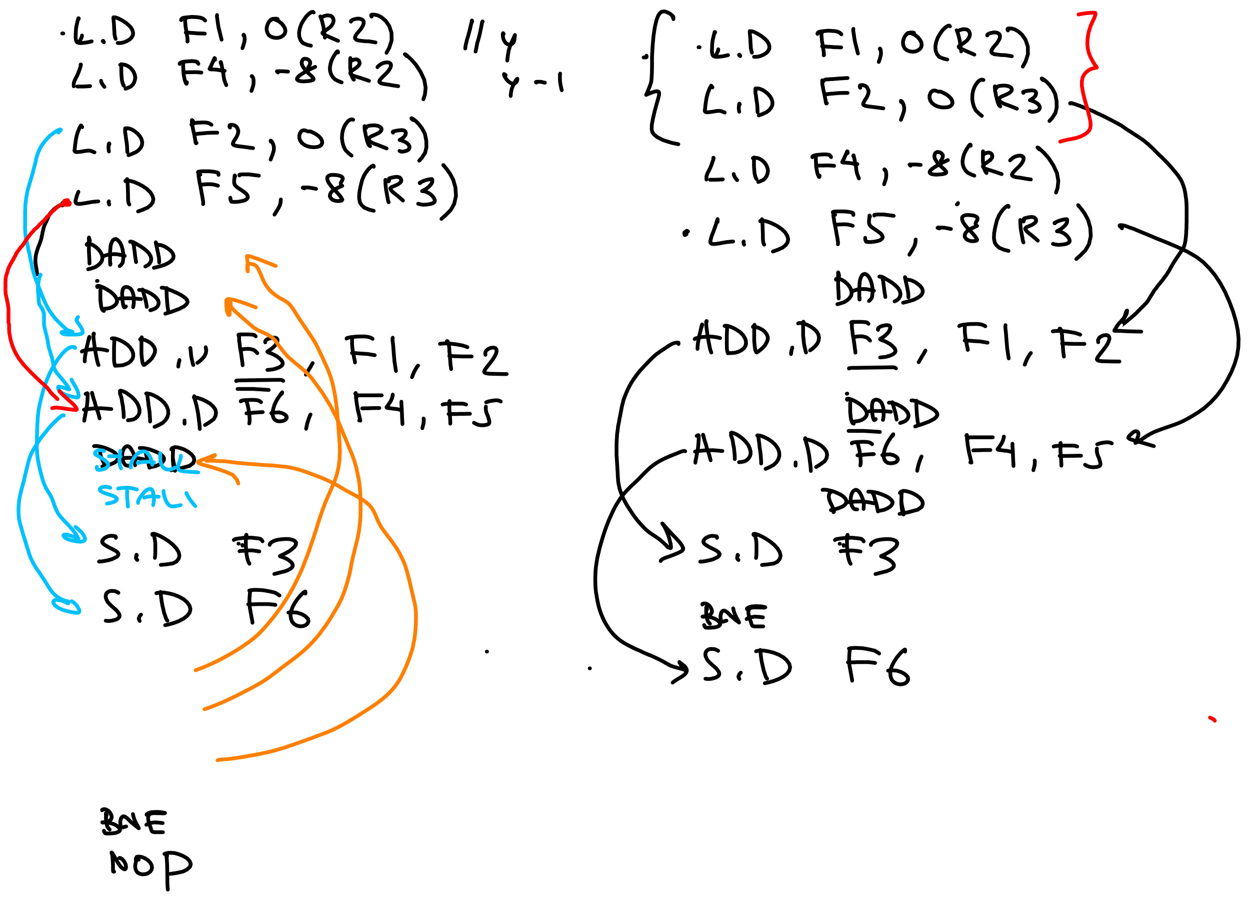
Assembly Code:

```
Loop:
    L.D F1, 0(R2) // Get y[i]
    L.D F2, 0(R3) // Get z[i]
    ADD.D F3, F2, F1 // Add the two numbers
    S.D F3, 0(R4) // Store the result into x[i]
    DADDUI R2, R2, #-8 // Decrement R2
    DADDUI R3, R3, #-8 // Decrement R3
    DADDUI R4, R4, #-8 // Decrement R4
    BNE R2, R1, Loop // Check if we've reached the end of the loop
    NOP
```

(a) (5 points) Show the schedule (what instruction issues in what cycle) for the default code.

```
L.D F1, 0(R2) // Get y[i]
L.D F2, 0(R3) // Get z[i]
DADDUI R2, R2, #-8 // Decrement R2
DADDUI R3, R3, #-8 // Decrement R3
DADDUI R4, R4, #-8 // Decrement R4
ADD.D F3, F2, F1 // Add the two numbers
STALL //
STALL
BNE R2, R1, Loop // Check if we've reached the end of the loop
NOP
S.D F3, 8(R4) // Store the result into x[i]
```

```
L.D  F1, 0(R2)    // y
L.D  F4, -8(R2)      y-1
L.D  F2, 0(R3)
L.D  F5, -8(R3)
DADD
DADD
ADD.D F3, F1, F2
ADD.D F6, F4, F5
~~STALL~~ DADD
STALL
S.D  F3
S.D  F6



BNE
NOP
```



```
·L.D  F1, 0(R2)     // y
L.D   F4, -8(R2)       y-1
L.D   F2, 0(R3)
L.D   F5, -8(R3)
 DADD
 DADD
ADD.D F3, F1, F2
ADD.D F6, F4, F5
 ~~STALL~~ DADD
 STALL
 S.D  F3
 S.D  F6


 BNE
 NOP
```

```
{ ·L.D  F1, 0(R2)
    L.D F2, 0(R3) }
  L.D   F4, -8(R2)
 ·L.D   F5, -8(R3)
        DADD
ADD.D F3, F1, F2
        DADD
ADD.D F6, F4, F5
        DADD
  S.D  F3
  BNE
  S.D  F6
```