## 1. Loop unrolling and SW pipelining

Source Code:

```
for (i=1000; i>0; i--) {
w[i] = x[i] * w[i];
}
```

Assembly Code:

```
Loop:
L.D F2, 0(R1) // Get w[i]
L.D F4, 0(R2) // Get x[i]
MUL.D F6, F2, F4 // Multiply two numbers
S.D F6, 0(R1) // Store the result into w[i]
DADDUI R1, R1, #-8 // Decrement R1
DADDUI R2, R2, #-8 // Decrement R2
BNE R1, R3, Loop // Check if we've reached the end of the loop
NOP
```

Assumptions:

1. Load feeding any instruction: 2 stall cycles
2. FP MUL feeding any instruction (except stores): 6 stall cycles
3. FP MUL feeding store: 5 stall cycles
4. Int add feeding a branch: 1 stall cycles
5. Int add feeding any other instruction: 1 stall cycle
6. 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)

1. Show the schedule (what instruction issues in what cycle) for the default code. (15 points)

Solution:

**Default Schedule:**

```
L.D F2, 0(R1)
L.D F4, 0(R2)          -> Load to any instruction: 2 stalls
STALL
STALL
MUL.D F6, F2, F4       -> FP Mul feeding store: 5 stalls
STALL
STALL
STALL
STALL
STALL
S.D F6, 0(R1)
DADDUI R1, R1, #-8
DADDUI R2, R2, #-8
BNE R1, R3, Loop
NOP (Delay slot)
```

Total number of cycles/stages = 15 cycles per iteration

2. How should the compiler order instructions to minimize stalls (without unrolling)(note that the execution of a NOP instruction is effectively a stall)? Show the schedule. How many cycles can you save per iteration, compared to the default schedule? (15 points)

Solution:

**Compiler Smart Schedule (No unrolling):**

```
L.D F2, 0(R1)
L.D F4, 0(R2)          -> Load to any instruction: 2 stalls (filled by int add after reordering)
DADDUI R1, R1, #-8
DADDUI R2, R2, #-8
MUL.D F6, F2, F4       -> FP Mul feeding store: 5 stalls (4 stalls unavoidable)
STALL
STALL
STALL
STALL
BNE R1, R3, Loop
S.D F6, 8(R1)          -> Delay slot filled by Store
```

Total number of cycles/stages = 11 cycles per iteration

**Total cycles saved from default schedule** = 15 - 11 = **4 cycles per iteration**

3. What is the minimum unroll degree to eliminate stall cycles? Show the schedule for the unrolled code. (20 points)

Solution:

If we unroll by a degree of 3 we still get 1 stall cycle, Hence to Completely eliminate stall cycles:

**Unroll by a degree of 4**

**Unroll Schedule:**

```
L.D F2, 0(R1)
L.D F4, 0(R2)
L.D F8, -8(R1)
L.D F10, -8(R2)
L.D F14, -16(R1)
L.D F16, -16(R2)
L.D F20, -24(R1)
L.D F22, -24(R2)
MUL.D F6, F2, F4
MUL.D F12, F8, F10
MUL.D F18, F14, F16
MUL.D F24, F20, F22
DADDUI R1, R1, #-32
DADDUI R2, R2, #-32
S.D F6, 32(R1)
S.D F12, 24(R1)
S.D F18, 16(R1)
BNE R1, R3, Loop
S.D F24, 8(R1)
```

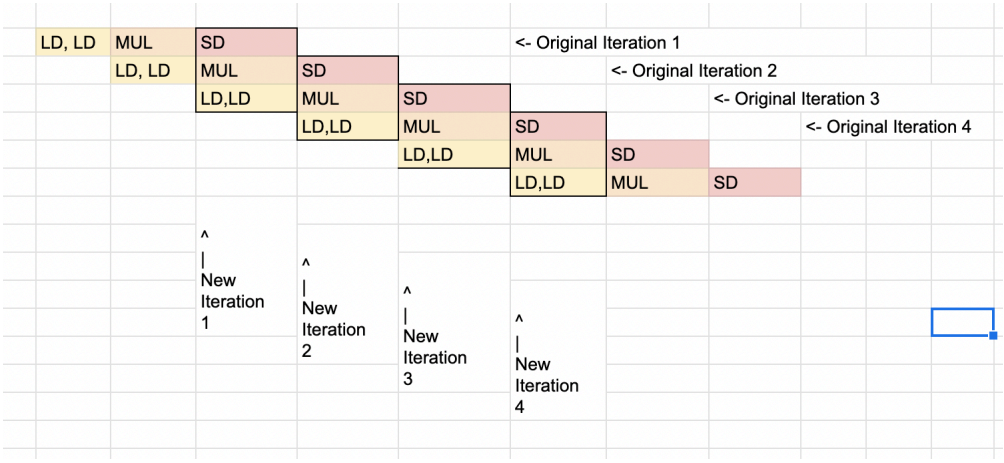
No stalls, 19 cycles for 4 iterations

4. Come up with a software-pipelined version of the code (no unrolling). Use appropriate register names and displacements in your code. When this code executes, will it experience any stalls? (20 points)

Solution:

**Software Pipelining Version**

Consider the key operations of the default code as shown in the diagram below, We can convert the original iterations to the new iterations without unrolling the loop to reduce stall cycles. The order of execution would reverse, still maintaining correctness.




Caption

**Software Pipelining Schedule:**

```
S.D F6, 16(R1)
MUL.D F6, F2, F4
L.D F4, 0(R2)
L.D F2, 0(R1)
DADDUI R2, R2, #-8
BNE R1, R3, Loop
DADDUI R1, R1, #-8
```

**When the code executes there will be no stalls.**

**5. Branch predictors (30 points)**

**Solution:**

Tournament branch predictor: selector with 64K entries (2-bit saturating counters).

Global: 16 bit global history XOR-ed with 16 bits of branch PC to index into a table of 3-bit saturating counters.

Local: 8 bits of branch PC index into level-1, 11 bits of local history from level-1 are concatenated with 4 other bits of branch PC to generate the index into level-2 that has 3-bit saturating counters.

Note: 2^10 bits = 1024 bits = 1 Kbit

a. Consider the Tournament branch predictor

Capacity = 64K * 2 bit
= 128 Kbits

b. Consider the Global predictor

16 bits Xor-ed with 16 bits gives a 16 bit index

Hence,

Capacity = (2^16) * 3 + 16 bit (Global history)
= ((2^6 * 3) * 2^10) + 16
= 64 * 3 Kbits + 16 bits
= 192 Kbits + (16/1024) Kbits
= 192.015 Kbits

c. Consider the Local predictor

8 bits required to index in level 1 (11 bit values):

Level 1 Capacity = 2^8 * 11
= 256 * 11
= 2816 bits
= 2.75 Kbits

11 bit of Level 1 concatenated with 4 bits of PC:

Level 2 Capacity = 2^15 * 3
= (2^5 * 3) * 2^10
= 96Kbits

Local Capacity = 96 + 2.75
= 98.75 Kbits

**Total capacity of the entire branch prediction system:**

= 128 + 192.015 + 98.75
= **418.765 Kbits**