Basic Pipeline Scheduling and Loop Unrolling

To keep a pipeline full, parallelism among instructions must be exploited by finding sequences of unrelated instructions that can be overlapped in the pipeline.
For these examples we will assume:

- The function units are fully pipelined, i.e. an instruction can be issued every cycle,
- there are no structural hazards,
- 1 branch delay slot, and
- the latencies are:

Instruction Producing Result	Instruction Using Result	Latency in cycles (# of nops between)
FP ALU op	Another FP ALU op	3
FP ALU op	Store Double	2
Load Double	FP ALU op	1
Load Double	Store Double	0

Suppose we want to execute:

```
for(i=1; i<=100; i++)
   x[i] = x[i] + s;
```

Each iteration is independent.
In assembly:

```
loop:
     ld   $f0, 0($t1)
     addd $f4, $f0, $f2
     sd   $f4, 0($t1)
     subi $t1, $t1, 8
     bneq $t1, loop
```

Unscheduled on MIPS — Clock Cycle Issued

```
loop:
     ld   $f0, 0($t1)                1


     addd $f4, $f0, $f2


     sd   $f4, 0($t1)


     subi $t1, $t1, 8


     bneq $t1, loop
```

_____ cycles required.

Scheduled Clock Cycle Issued

```
loop:
     ld   $f0, 0($t1)                1

     addd $f4, $f0, $f2

     subi $t1, $t1, 8

     bneq $t1, loop

     sd   $f4, _______($t1)
```

_____ cycles required.

Much faster!

This is a non-trivial change. Most compilers would see that the `sd` depends on the `subi` and wouldn't exchange them. A smarter compiler/assembler would do it.
One load-add-store loop requires 6 clock cycles, but load-add-store only requires 3 cycles. The other 3 cycles are `subi`, `bnez`, and *stall*.

Use **loop unrolling** to do more per loop.
Unroll the loop 4 times:

```
loop:
     ld   $f0, 0($t1)
     addd $f4, $f0, $f2
     sd   $f4, 0($t1)

     ld   ______, ______ ($t1)

     addd ______, $f6, $f2

     sd   ______, ______ ($t1)

     ld   $f10, ______ ($t1)
     addd $f12, $f10, $f2

     sd   $f12, ______ ($t1)

     ld   $f14, ______ ($t1)
     addd $f16, $f14, $f2

     sd   $f16, ______ ($t1)

     subi $t1, $t1, ______
     bneq $t1, loop
```

Note: Saved 2*3=6 cycles
Used different registers so we can schedule better.

1 loop takes 27 cycles or 6.8 clock cycles per element.

This unrolled version is slower than the standard.

In real programs we don't always know the number of loops. Suppose we have *n* loops, we can unroll to have *k* copies. Then we would have 2 copies of code. The first would execute *n mod k* times, then second would execute *n/k* times.

Scheduled – unrolled loop

```
loop:
     ld   $f0,   0($t1)
     ld   $f6,  -8($t1)
     ld   $f10,-16($t1)
     ld   $f14,-24($t1)

     addd $f4,  $f0, $f2
     addd $f8,  $f6, $f2
     addd $f12, $f10,$f2
     addd $f16, $f14,$f2

     sd   $f4,   0($t1)
     sd   $f8,  -8($t1)
     sd   $f12,-16($t1)

     subi $t1, $t1, ______
     bneq $t1, loop

     sd   $f16, ______ ($t1)
```

Unrolling exposes more computation that can be scheduled to minimize stalls.