Modern Processor Architectures

L25: Modern Compiler Design

The 1960s - 1970s

- Instructions took multiple cycles
- Only one instruction in flight at once
- Optimisation meant minimising the number of instructions executed
- Sometimes replacing expensive general-purpose instructions with specialised sequences of cheaper ones

The 1980s

- CPUs became pipelined
- Optimisation meant minimising pipeline stalls
- Dependency ordering such that results were not needed in the next instruction
- Computed branches became very expensive when not correctly predicted

Stall Example


```
for (int i=100 ; i!=0 ; i--)
{
    ...
}
```

```
start:
    ...
    add r1, r1, 1
    jne r1, 0, start
```

Stall Example


```
for (int i=100 ; i!=0 ; i--)
{
    ...
}
```

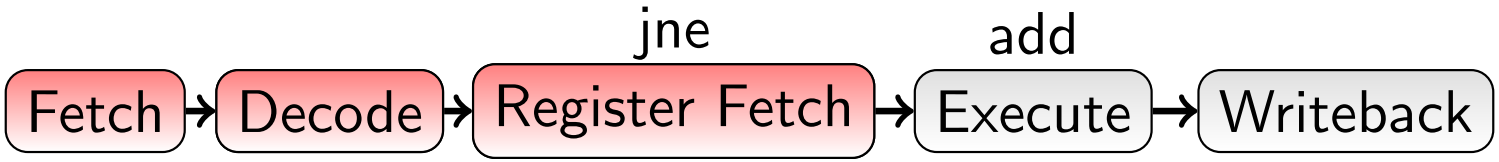
```
start:
    ...
    add r1, r1, 1
    jne r1, 0, start
```

Stall Example


```
for (int i=100 ; i!=0 ; i--)
{
    ...
}
```

```
start:
    ...
    add r1, r1, 1
    jne r1, 0, start
```

Stall Example




```
for (int i=100 ; i!=0 ; i--)
{
    ...
}
```

```
start:
    ...
    add r1, r1, 1
    jne r1, 0, start
```

Stall Example


```
for (int i=100 ; i!=0 ; i--)
{
    ...
}
```

```
start:
    ...
    add r1, r1, 1
    jne r1, 0, start
```

Stall Example


```
for (int i=100 ; i!=0 ; i--)
{
    ...
}
```

```
start:
    ...
    add r1, r1, 1
    jne r1, 0, start
```

Stall Example


```
for (int i=100 ; i!=0 ; i--)
{
    ...
}
```

```
start:
    ...
    add r1, r1, 1
    jne r1, 0, start
```

Fixing the Stall

```
for (int i=100 ; i!=0 ; i--)
{
    ...
}
```

```
start:
    add r1, r1, 1
    ...
    jne r1, 0, start
```

Fixing the Stall

```
for (int i=100 ; i!=0 ; i--)
{
    ...
}
```

```
start:
    add r1, r1, 1
    ...
    jne r1, 0, start
```

Is this a good solution?

The Early 1990s

- CPUs became much faster than memory
- Caches hid some latency
- Optimisation meant maximising locality of reference, prefetching
- Sometimes, recalculating results is faster than fetching from memory

The Mid 1990s

- CPUs became superscalar
  - Independent instructions executed in parallel
- CPUs became out-of-order
  - Reordered instructions to reduce dependencies
- Optimisation meant structuring code for highest-possible ILP
- Loop unrolling no longer such a big win

Superscalar CPU Pipeline Example: Sandy Bridge

Can dispatch up to six instructions at once, via 6 pipelines:

1. ALU, VecMul, Shuffle, FpDiv, FpMul, Blend
2. ALU, VecAdd, Shuffle, FpAdd
3. Load / Store address
4. Load / Store address
5. Load / Store data
6. ALU, Branch, Shuffle, VecLogic, Blend

Branch Predictors

- Achieve 95+% accuracy on modern CPUs
- No cost when branch is correctly predicted
- Long and wide pipelines mean very expensive for the remaining 5%!

The Late 1990s

- SIMD became mainstream
- Factor of 2-4$\times$ speedup when used correctly
- Optimisation meant ensuring data parallelism
- Loop unrolling starts winning again, as it exposes later optimisation opportunities (more on this later)

The Early 2000s

- (Homogeneous) Multicore became mainstream
- Power efficiency became important
- Parallelism provides both better throughput and lower power
- Optimisation meant exploiting fine-grained parallelism

The Late 2000s

- Programmable GPUs became mainstream
- Hardware optimised for stream processing in parallel
- Very fast for massively-parallel floating point operations
- Cost of moving data between CPU and CPU is high
- Optimisation meant offloading operations to the GPU

The 2010s

- Modern processors come with multiple CPU and GPU cores
- All cores behind the same memory interface, cost of moving data between them is low
- Increasingly contain specialised accelerators
- Often contain general-purpose (programmable) cores for specialised workload types (e.g. DSPs)
- Optimisation is hard.

Questions?