Compilers

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

- Instruction Scheduling

Compiler Phases: Front and Back Ends

- The front-end phases are:
  - Scanning
  - Parsing
  - Semantic analysis
- The back-end phases are:
  - Register Allocation
  - **Instruction Scheduling**
  - Code generation

- Instruction Scheduling – Choose the order of instructions that will minimize the time it takes for the program to run.

Instruction Scheduling

- Latency – How long it takes before result becomes available.
- We will measure latency in clock cycles.
- Assume the following latencies for operations (assumes a cache hit when loading):

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

This load time assumes that it found the variable in the cache

- If a cache miss occurs during a load, then the number of cycles required for the load will be in the hundreds.

Latency

Program Duration

- Estimate the cycles to run the following program for x*y.
- Cycles start counting from 1.

```
load x, r1
load y, r2
mult r1, r2, r3
```

Cycle Start	Cycle End	Instruction
???		

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Program Duration (no pipelining)

- Count the cycles to run program x*y.

Cycle Start	Cycle End	Instruction
1	3	load x, r1
4	6	load y, r2
7	8	mult r1, r2, r3

Start is the cycle the instruction starts in and end is the cycle that it completes

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Total Cycles
8

Program Duration

- Estimate the cycles to run the following program for x*y+z:

```
load x, r1
load y, r2
mult r1, r2, r3
load z, r4
add r3, r4, r5
```

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Program Duration (no pipelining)

- Count the cycles to run program x*y+z.

Cycle Start	Cycle End	Instruction
1	3	load x, r1
4	6	load y, r2
7	8	mult r1, r2, r3
9	11	load z, r4
12	12	add r3, r4, r5

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Total Cycles
12

Program Duration

- Estimate the cycles to run the following program for w=x*y+z:

```
load x, r1
load y, r2
mult r1, r2, r3
load z, r4
add r3, r4, r5
store r5, w
```

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Program Duration (no pipelining)

- Count the cycles to run program w=x*y+z.

Cycle Start	Cycle End	Instruction
1	3	load x, r1
4	6	load y, r2
7	8	mult r1, r2, r3
9	11	load z, r4
12	12	add r3, r4, r5
13	15	store r5, w

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Total Cycles
15

Pipelining

- Current processors allow you the option to execute instructions in parallel.
- You can start a new instruction during each cycle.
- You are allowed to start another instruction even if there are instructions that have not completed yet.
- The new instruction must be independent of any instructions that are currently in the pipeline (the new instruction cannot depend on results from other instructions currently in the pipeline).
- Pipelining allows you to execute some instructions in parallel.

Pipelining

Pipelining Example.

- load r1,a starts at clock cycle 1.
- load r2,b starts at the next clock cycle even though the first load instruction has not finished yet.
- add r1,r2,r3 must wait for the other two loads to finish because it needs the data from them.

Inst\Cycle	1	2	3	4	5
load r1,a	running	running	running		
load r2,b		running	running	running	
add r1,r2,r3					running

load r2,b does not wait for load r1,a to finish (starts immediately)

Add instruction must wait for loads to finish because it needs their data

Pipelining Example

Program Duration (pipelining)

- Estimate the cycles to run the following program for x*y <u>with pipelining</u>:

load x, r1
load y, r2
mult r1, r2, r3

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Program Duration (pipelining)

- Count the cycles to run program x*y with pipelining.

Start	End	Instruction
1	3	load x, r1
2	4	load y, r2
5	6	mult r1, r2, r3

Start load y,r2 even though the previous load has not finished

mult must wait until both loads are finished

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Total Cycles

6

Pipelining vs No Pipelining

- Count the cycles to run program x*y.

Pipelining

Start	End	Instruction
1	3	load x, r1
2	4	load y, r2
5	6	mult r1, r2, r3

Total Cycles (pipelining)
6

No Pipelining

Start	End	Instruction
1	3	load x, r1
4	6	load y, r2
7	8	mult r1, r2, r3

Total Cycles (no pipelining)
8

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

- Estimate the cycles to run the following program for x*y+z with pipelining:

load x, r1
load y, r2
mult r1, r2, r3
load z, r4
add r3, r4, r5

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Program Duration (pipelining)

Program Duration (pipelining)

- Count the cycles to run program x*y+z with pipelining.

Start	End	Instruction
1	3	load x, r1
2	4	load y, r2
5	6	mult r1, r2, r3
6	8	load z, r4
9	9	add r3, r4, r5

Start load y,r2 even though the previous load has not finished

mult must wait until both loads are finished

Start load z,r4 even though the mult load has not finished

add must wait for load z,r4 to finish

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Total Cycles

9

- Count the cycles to run program x*y+z.

Pipelining

Start	End	Instruction
1	3	load x, r1
2	4	load y, r2
5	6	mult r1, r2, r3
6	8	load z, r4
9	9	add r3, r4, r5

Total Cycles (pipelining)
9

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

No Pipelining

Start	End	Instruction
1	3	load x, r1
4	6	load y, r2
7	8	mult r1, r2, r3
9	11	load z, r4
12	12	add r3, r4, r5

Total Cycles (no pipelining)
12

Pipelining vs No Pipelining

Program Duration (pipelining)

- Estimate the cycles to run the following program for w=x*y+z <u>with pipelining</u>:

```
load x, r1
load y, r2
mult r1, r2, r3
load z, r4
add r3, r4, r5
store r5, w
```

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Program Duration (pipelining)

- Count the cycles to run program w=x*y+z with pipelining.

Start	End	Instruction
1	3	load x, r1
2	4	load y, r2
5	6	mult r1, r2, r3
6	8	load z, r4
9	9	add r3, r4, r5
10	12	store r5, w

Start load y,r2 even though the previous load has not finished

mult must wait until both loads are finished

Start load z,r4 even though the mult load has not finished

add must wait for load z,r4 to finish

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Total Cycles
12

- Count the cycles to run program w=x*y+z.

Pipelining

Start	End	Instruction
1	3	load x, r1
2	4	load y, r2
5	6	mult r1, r2, r3
6	8	load z, r4
9	9	add r3, r4, r5
10	12	store r5, w

Total Cycles (pipelining)
12

No Pipelining

Start	End	Instruction
1	3	load x, r1
4	6	load y, r2
7	8	mult r1, r2, r3
9	11	load z, r4
12	12	add r3, r4, r5
13	15	store r5, w

Total Cycles (no pipelining)
15

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Pipelining vs No Pipelining

- Pipelining helped decrease the number of cycles, but we can still do better.
- If the instructions are reordered it is possible to further reduce the number of cycles required to run the program.
- Instructions that are independent of other instructions can be moved.
- An independent instruction can be run while another instruction is waiting for data that it needs.
- The idea is to increase the instruction-level parallelism.
- The instruction scheduler is responsible for this reordering.

Instruction Scheduling

Program Duration (pipelining and instruction scheduling)

- Estimate the cycles to run the following program for x*y+z with pipelining and instruction scheduling (reorder instructions to minimize the time).

```
load x, r1
load y, r2
mult r1, r2, r3
load z, r4
add r3, r4, r5
```

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Program Duration (pipelining and instruction scheduling)

- Count the cycles to run program x*y+z.

Start	End	Instruction
1	3	load x, r1
2	4	load y, r2
3	5	**load z, r4**
6	7	**mult r1, r2, r3**
8	8	add r3, r4, r5

Move load z,r4 before the mult (this load is independent).

All three loads are being done in parallel (instruction-level parallelism).

mult is waiting for x and y to be loaded. The load of z can take place during this downtime. This "hides" the latency of the x and y loads.

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Total Cycles
8

- Count the cycles to run program x*y+z.

No Speedups

Start	End	Instruction
1	3	load x, r1
4	6	load y, r2
7	8	mult r1, r2, r3
9	11	load z, r4
12	12	add r3, r4, r5

Total Cycles
12

Pipelining

Start	End	Instruction
1	3	load x, r1
2	4	load y, r2
5	6	mult r1, r2, r3
6	8	load z, r4
9	9	add r3, r4, r5

Total Cycles
9

Pipelining and Instr. Scheduling

Start	End	Instruction
1	3	load x, r1
2	4	load y, r2
3	5	**load z, r4**
6	7	**mult r1, r2, r3**
8	8	add r3, r4, r5

Total Cycles
8

Instruction	Latency (in cycles)
load	3
store	3
add	1
mult	2

Pipelining and Instr. Scheduling

End of Slides

- **End of Slides**