Lecture 1
An Overview of High-Performance Computer Architecture

ECE 463/521
Fall 2002
Edward F. Gehringer

Automobile Factory
(note: non-animated version)

Station 1 Station 2 Station 3 Station 4 Station 5
Build frame Connect doors Connect headlights Embed engine Connect wheels & transmission
Automobile Factory

Station 1: Build frame
Station 2: Connect doors
Station 3: Connect headlights
Station 4: Embed engine
Station 5: Connect wheels & transmission

Basic Assembly Line

- Unchangeable truth
  - It takes a long time to build one car
  - Example: Time spent in assembly line is 1 hour (12 min. per station)
- Basic assembly line
  - Throughput = 1 car per hour
  - We wait until first car is fully assembled before starting the next one:
    - ⇒ only 1 car in assembly line at a time
    - ⇒ only 1 station is active at a time; other 4 are idle
Automobile Factory (note: non-animated version)

Station 1: Build frame
Station 2: Connect doors
Station 3: Connect headlights
Station 4: Embed engine
Station 5: Connect wheels & transmission

Pipelined Assembly Line

• Unchangeable truth
  – It still takes a long time to build one car
• Pipelining
  – Time to fill pipeline = 1 hour
  – Once filled, throughput = 1 car per 12 minutes
  – Speedup due to pipelining is (unusual definition)...

\[
\text{speedup} = \frac{\text{rate of pipeline}}{\text{rate of unassembled}} = \frac{1 \text{ car per 12 min}}{1 \text{ car per 60 min}} = 4
\]
Simple Processor Pipeline

![Pipeline Diagram]

Example Instruction

- **ADD r1, r2, r3**
  - r1 ← r2 + r3
  - IF: Fetch the ADD instruction from memory using the current PC (program counter), then PC ← PC + 1
  - ID: Decode the ADD instruction to determine the opcode, read values of r2 and r3 from the register file
  - EX: Perform r2 + r3 in the ALU (arithmetic/logic unit)
  - MEM: Do nothing (only loads/stores access memory)
  - WB: Write result of r2 + r3 into r1, in the register file

Pipeline Performance Problems (1)

- Data dependences
  - **ADD r1, r2, r3**
  - **SUB r4, r1, r9**
  - SUB must wait ("stall") in ID stage until ADD completes
    - ADD writes the result r1 into register file in WB
    - SUB reads the result r1 from register file in ID
Data Dependence Stalls

ADD
IF (instruction fetch)
Stage 1
Fetch instruction

ID (instruction decode)
Stage 2
Decode & read registers

EX (execute)
Stage 3
Execute memory

MEM (memory)
Stage 4
Access memory

WB (write-back)
Stage 5
Write result

SUB ADD
Stage 1
Fetch instruction

ADD IF (instruction fetch)
Stage 2
Decode & read registers

EX (execute)
Stage 3
Execute memory

MEM (memory)
Stage 4
Access memory

WB (write-back)
Stage 5
Write result

Data Dependence Stalls

r1
Register file
r1

IF (instruction fetch)
Stage 1
Fetch instruction

SUB (fetch) ID (instruction decode)
Stage 2
Decode & read registers

ADD (fetch) EX (execute)
Stage 3
Execute memory

MEM (memory)
Stage 4
Access memory

WB (write-back)
Stage 5
Write result
Data Dependence Stalls

Stage 1: Fetch instruction
Stage 2: Decode & read registers
Stage 3: Execute
Stage 4: Access memory
Stage 5: Write result

SUB (stalled)
ADD (bubble)

IF (instruction fetch)
ID (instruction decode)
EX (execute)
MEM (memory)
WB (write-back)
Speedup with data dependences

- What is the speedup of this pipeline $(T_{sequential}/T_{pipelined})$ if 1/10th of all instructions contain a data dependence?
- Can you give a general formula for a $k$-stage pipeline? What other information do you need to know?

Reducing Data Dependence Stalls

- We could directly forward results from producer to consumer, bypassing the register file.
  - The hardware is called “data bypass,” “result bypass,” or “register file bypass.”
  - The technique is called “bypassing” or “forwarding.”

Data Bypass
Data Bypass

Stage 1
Fetch instruction

Stage 2
Decode & read registers

Stage 3
Execute

Stage 4
Access memory

Stage 5
Write result

SUB
IF (instruction fetch)

ADD
ID (instruction decode)

EX
Execute

MEM
(memory)

WB
(write-back)

r1 (garbage)

Register file

data bypass

Stage 1
Fetch instruction

Stage 2
Decode & read registers

Stage 3
Execute

Stage 4
Access memory

Stage 5
Write result

SUB
IF (instruction fetch)

ADD
ID (instruction decode)

EX
Execute

MEM
(memory)

WB
(write-back)
Pipeline Performance Problems (2)

- Branches
  
  ```
  ADD r1, r2, r3
  BEQ X, r5, r7
  SUB r4, r1, r9
  LD r4, 10(r4)
  ```

- Which instruction should be fetched after the branch?
- IF stage stalls until BEQ reaches EX stage.
- EX stage evaluates branch condition (r5 = r7).

Branch Stalls

```
ADD IF
  (instruction fetch)
Stage 1
  Fetch instruction

ID
  (instruction decode)
Stage 2
  Decode & read registers

EX
  (execute)
Stage 3
  Access memory

MEM
  (memory)
Stage 4
  Write result

WB
  (write-back)
Stage 5
  ```
Branch Stalls

Branch outcome: taken

Branch Stalls
Reducing Branch Stalls

• Branch prediction
  – “Learn” which way a given branch tends to go.
  – Like predicting the economy, branch prediction is based on past history.
  – Even simple predictors can be 80% accurate.
  – If correct: no branch stalls.
  – In incorrect:
    • “Quash” instructions in previous pipeline stages.
    • Performance degrades to the stall case.
    • May have additional penalties to “clean up” the pipeline.

Branch Prediction (correct)

Branch Prediction (correct)
Branch Prediction (correct)

Predict taken

AND  BEQ  ADD  MEM  WB
IF (instruction fetch) ID (instruction decode) EX (execute) (memory) (write-back)
Stage 1  Stage 2  Stage 3  Stage 4  Stage 5
Fetch instruction  Decode & read registers  Execute memory  Write result

Branch Prediction (incorrect)

Stage 1  Stage 2  Stage 3  Stage 4  Stage 5
Fetch instruction  Decode & read registers  Execute memory  Write result

Branch Prediction (incorrect)

Stage 1  Stage 2  Stage 3  Stage 4  Stage 5
Fetch instruction  Decode & read registers  Execute memory  Write result
Branch Prediction (incorrect)

Predict not taken

SUB
IF (Instruction fetch)

BEQ
ID (Instruction decode)

ADD
EX (execute)

MEM
(Memory)

WB
(write-back)

Stage 1
Fetch instruction

Stage 2
Decode & read registers

Stage 3
Execute

Stage 4
Access memory

Stage 5
Write result

Branch Prediction (incorrect)

Branch outcome: taken

BEQ
IF (Instruction fetch)

ADD
EX (execute)

MEM
(Memory)

WB
(write-back)

Stage 1
Fetch instruction

Stage 2
Decode & read registers

Stage 3
Execute

Stage 4
Access memory

Stage 5
Write result

Branch Prediction (incorrect)

AND
IF (Instruction fetch)

BEQ
ID (Instruction decode)

MEM
EX (execute)

WB
(Memory)

(write-back)

Stage 1
Fetch instruction

Stage 2
Decode & read registers

Stage 3
Execute

Stage 4
Access memory

Stage 5
Write result
Speedup with branch stalls

• What is the speedup of the pipeline if 1/5 of the instructions are branches, and 4/5 of those are correctly predicted?
• Can you give a general formula for a $k$-stage pipeline? What other information do you need to know?

Sears Tower Repairman

• Repair shop is in the basement
  – Has many tools.
  – A few are used frequently,
    • e.g., hammer, crescent wrench, screwdriver
  – Most are used infrequently,
    • e.g., socket wrenches

Sears Tower Repairman

• Problem
  – Sears Tower has 110 stories!
  – Today, you are working on the top floor.
  – Can’t bring entire shop with you.
  – Don’t know exactly which tools to bring with you from the basement.
Sears Tower Repairman

• Solution
  – Carry frequently used tools in your tool belt.
  – Tool-belt becomes a “cache” of tools — drastically reduces the number of trips down to the basement.
  – When you have to fetch ¼” socket wrench, common sense says to also fetch ½”, ¾”, etc., just in case.

Caches

• The processor-memory speed gap
  – Processor is very fast
    • Intel Pentium 4: 1 GHz, 1 clock cycle = 1 ns
  – Large memory is slow!
    • Main memory: 50 ns to access, 50 times slower than Pentium 4!
  – Processor wants large and fast memory.
    • LARGE: O/S and applications consume lots of memory
    • FAST: Otherwise, processor stalls nearly 100% of time waiting for memory.

Caches

```
Processor
1-ns clock

64KB cache memory
2-ns read time

Tool-belt
Hits 95% of time

256MB Main Memory
50-ns read time

Basement shop
```
Average access time

- What is the average access time in this memory system?

Caches

- Caches are effective because of locality of reference.
  - Temporal locality: If you access an item, you are likely to access it again in near future.
  - Spatial locality: If you access an item, you are likely to access a nearby item in the near future.
    - This is why repairman also fetched ½” and ¾” socket wrenches when they only needed ¼”.

Overview of Topics in 463/521

- Measuring performance and cost
- Caches and memory hierarchies
- Instruction-set architecture (ISA)
  - Defines software/hardware interface
- Simple pipelining
  - Data and control (branch) dependences
  - Data bypasses
  - Branch prediction
Overview of Topics in 463/521

- Complex pipelining and instruction-level parallelism (ILP).
  - Data hazards
  - Dynamic instruction scheduling, register renaming, Tomasulo’s algorithm.
  - Precise interrupts
  - Superscalar, VLIW, and vector processors.

Projects

- Three projects
  - Cache simulator
  - Branch predictor simulator
  - Dynamic instruction scheduling pipeline simulator
- Programming for projects is harder than anything most of you have encountered before.

Course Web site

- http://courses.ncsu.edu/ece463/lec/001
- http://courses.ncsu.edu/ece521/lec/001
- These two homepages are linked to a “common” Web site.
- Any info specific to one course will be listed in the announcements section of that course’s home page.