CS 2461: Computer Architecture 1
Program performance and High Performance Processors
Instructor: Prof. Bhagi Narahari

Course Objectives: Where are we...

- Bits&bytes: Logic devices
  - HW building blocks
- Processor: ISA, datapath
  - Using building blocks to assemble a processor (LC3)
- Programming the processor: Assembly
- Translating high level programs to Proc
  - Implementing C on LC3

Bits&Bytes to High level Programs

- User application written in high level language
- Program runs on a processor

- How are high level programs implemented on processor?
  - Run-time stack, allocation of variables, translation of high level code to machine code
  - Map high level data structures to low level data structures
    - Struct to linear mapping in memory
- What else does software developer want after program is implemented correctly?

  - PERFORMANCE!

Next...

- Performance of programs
  - What to measure
  - Model?
  - Technology trends
- "real" processors...how to improve performance
  - Pipelining, ILP, Multi-core
- Memory organization basics
  - Memory hierarchy: cache, main memory, etc.
- How to rewrite your program to make it run faster...code optimization
Performance of Programs

- "Complexity" of algorithms
- How good/efficient is your algorithm
  - Measure using Big-O notation: \( O(N \log N) \)
- Next question: How well is the code executing on the machine ???????
  - Actual time to run the program
  - What are the factors that come into play
  - Where is the program and data stored
  - What are the actual machine instructions executed
  - Why is some HW better than others for different programs?
  - What factors of system performance are HW related
  - How does machine instruction set affect performance
  - What are the technology trends and how do they play a role?

Program Performance: The Great Reality – Our focus

- There's more to performance than asymptotic complexity
- Must optimize at multiple levels:
  - algorithm, data representations, procedures, and loops
- Must understand system to optimize performance
  - How programs are compiled and executed
  - How is data stored
  - What data structures are used
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Technology Trends & Performance

- Speed will depend on clock cycle (frequency) of the circuits
  - How fast can we switch the transistors
    - Feed the signal to the gate of MOS transistor, how long for the transistor to throw the switch
  - How large is the transistor – feature size
- Moore's Law
  - Founder of Intel hypothesized on rate of increase in performance
  - It is not a law in the sense of laws of physics, etc.
  - Observations: performance doubles every 18 months
    - If you knew this, how would it guide your business decisions?
    - Case study: Apple Computers in '85

Delay vs. Feature Size

- Speed will depend on clock cycle (frequency) of the circuits
- Moore's Law
- How large is the transistor – feature size

Delay vs. Feature Size

The CPU-Memory Gap

- The increasing gap between DRAM, disk, and CPU speeds.

Performance Trends: Summary

- Workstation performance (measured in Spec Marks) improves roughly 50% per year (2X every 18 months)
  - Performance will include not just processor, but memory and disk I/O
- Improvement in cost performance estimated at 70% per year
Performance: What to measure?

Which of these airplanes has the best performance?

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 747</td>
<td>6.5 hours</td>
<td>610 mph</td>
<td>470</td>
<td></td>
</tr>
<tr>
<td>BAD/Sud Condoré</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td></td>
</tr>
</tbody>
</table>

The Bottom Line: Performance metric depends on application

Computer Performance: TIME, TIME, TIME

- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?

- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

Metric chosen usually depends on user community: sys admin vs single user?

- If we upgrade a machine with a new processor what do we increase?
- If we add a new machine to the lab what do we increase?
**Execution Time**

- **Elapsed Time**
  - counts everything (disk and memory accesses, I/O, etc.)
  - a useful number, but often not good for comparison purposes
- **CPU time**
  - doesn’t count I/O or time spent running other programs
  - can be broken up into system time, and user time
- Our focus in this course: user CPU time
  - time spent executing the lines of code that are "in" our program

**How to Model Performance**

- The asymptotic complexity – “big O”
  - Time = O( f(n)) : function of the size of the input
  - Sorting O(n log n)
  - This measures efficiency of your algorithm
  - i.e., how ‘good’ is solution technique
  - Is this enough when we talk of actual time measured on the processor ???
- There’s more to performance than asymptotic complexity
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- Must understand system to optimize performance
  - How programs are compiled and executed, data storage, data structures, I/O management

**Processor time: how to measure?**

- Number of clock cycles it takes to complete the execution of your program
- What is your program
  - A number of instructions
    - Different types: load, store, ALU, branch
  - Stored in memory
  - Executed on the CPU

**Aspects of CPU Performance**

<table>
<thead>
<tr>
<th>CPU time = Seconds * Instructions * Cycles * Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
</tr>
<tr>
<td>Program</td>
</tr>
<tr>
<td>Instruction</td>
</tr>
<tr>
<td>‘Cycle’</td>
</tr>
</tbody>
</table>

\[
CPU = IC * CPI * Clk
\]
CPI

- Cycles per instruction
  - Different instructions may take different time
  - Example in LC 3?
- We observed that not every instruction needs to go through all the instruction execution steps
  - E.g.: no need to calculate effective address, fetch from memory or registers
  - Reality: different times associated with different operations
    - Especially true of memory operations

Average CPI

- Application has an “instruction mix”
  - Profile of application instruction types
    - ALU, Load/Store (memory), Branch, Jumps, etc.
    - $x_1, x_2, x_3, ...$ as percentage ($x_1=0.4$)
- Processor has CPI for each type of instruction
  - Part of ISA of a processor…specifications doc
    - Example: ALU=1.0, Load/Store=2.0, etc.
    - $t_1, t_2, t_3, ...$
- What is effective CPI?
- Weighted average
  - $CPI = x_1t_1 + x_2t_2 + ...$

CPI: Cycles per instruction

- Depends on the instruction
  - $CPI_i = \text{Execution time of instruction } / \text{Cycle time}$
- Average cycles per instruction
  - $CPI = \sum_{i=0}^{n} CPI_i \times F_i$, where $F_i = \frac{IC_i}{IC_{tot}}$
- Example:

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI</th>
<th>%Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>0.5</td>
<td>32%</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
<td>0.4</td>
<td>27%</td>
</tr>
<tr>
<td>Store</td>
<td>15%</td>
<td>2</td>
<td>0.2</td>
<td>13%</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>0.4</td>
<td>27%</td>
</tr>
<tr>
<td>CPI$_{tot}$</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Principles of Computer Architecture Design: Thumb Rules

- Common case fast
  - Focus on improving those instructions that are frequently used
  - Amdahl’s Law
    - Fraction enhanced/optimized runs faster
- Principle of Locality:
  - Program spends 90% of its time in 10% of code
    - E.g.: word processing
  - Spatial: items near each other tend to be accessed
  - Temporal: recently used items tend to be used again
- Concurrency/Parallelism
  - Overlap the instruction execution steps
    - Pipeline processors
    - Multi-core processors
Amdahl's Law: Speedup

- Application takes X time
- How to run it faster
  - Enhance/optimize a portion of it
  - Which portion
  - Can we enhance all of it
  - Note that we are talking of solving the enhanced part in a different way, and possibly using different (more costly) resources
- Where to focus our optimizations?
  - Look at return on investment
  - Code segments that take long time can give us the best returns
  - Profile your code to understand which parts are dominating

Improving Performance of processors: quick review

- Are “real” processors like LC 3?
- How can we improve the performance of the processor?
  - What design principles?
- Quick overview of techniques used in ‘real’ processor designs
  - Pipelining…
  - Instruction level parallel (ILP) processors
  - Multithreaded processors → multi-core

Real-World Pipelines: Car Washes

Sequential

Parallel

Pipelined

- Idea
  - Divide process into independent stages
  - Move objects through stages in sequence
  - At any given times, multiple objects being processed

Instruction Pipeline

- Instruction execution process lends itself naturally to pipelining
  - overlap the subtasks of instruction fetch, decode and execute

| Inst Fetch | Decode | Execute | Mem Access | Write Back Result |
Speedup of Pipelines

- If we have a $k$ stage pipeline, and $n$ tasks (instructions) to process:
  - time to complete $n$ tasks (instructions) $T_n = k + (n-1)$ cycles
  - Time for non-pipelined = $nk$ cycles
  - Therefore speedup using $k$ stage pipeline $S_k = T_n / T_k$
    - $= nk / (k + n-1)$ ...for large $n$, this is
    - $\sim nk/n = k$
  - Challenges:
    - Data hazards...solved in internal forwarding
    - Control hazards (branches)...prediction but still a problem

Example

- Suppose we execute 100 instructions
  - Single Cycle Machine
    - 45 ns/cycle x 1 CPI x 100 inst = 4500 ns
  - Multicycle Machine
    - 10 ns/cycle x 4.6 CPI (due to inst mix) x 100 inst = 4600 ns
  - Ideal pipelined machine
    - 10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns
So how hard is it to design a Pipelined Processor

- Go back and examine your datapath and control diagram
- associated resources with states
- ensure that flows do not conflict, or figure out how to resolve
- assert control in appropriate stage
Can't be that easy....Problems?

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle and introduce stall cycles which increase CPI.
- Structural hazards: HW cannot support this combination of instructions - two dogs fighting for the same bone.
- Data hazards: Instruction depends on result of prior instruction still in the pipeline.
  - Data dependencies
- Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).
  - Control dependencies
- Can always resolve hazards by stalling
- But, more stall cycles = more CPU time = less performance
  - Increase performance = decrease stall cycles

Back to our old friend: CPU time equation

- Recall equation for CPU time

\[
\text{CPU time} = \frac{\text{RC * CPI * CR}}{\text{Instructions} \times \text{Cycles} \times \text{Seconds}}
\]

- So what are we doing by pipelining the instruction execution process?
  - Clock?
  - Instruction Count?
  - CPI?
  - How is CPI effected by the various hazards?

Speed Up Equation for Pipelining

\[
\text{CPI}_{\text{pipelined}} = \text{Ideal CPI} + \text{Average Stall cycles per Inst}
\]

\[
\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{rep} \text{lined}}}{\text{Cycle Time}_{\text{pipelined}}}
\]

More stalls means lower performance!

One Memory Port/Structural Hazards
Data Dependencies

- True dependencies and False dependencies
  - false implies we can remove the dependency
    - i.e., compiler can remove them
  - true implies we are stuck with it!
- Three types of data dependencies defined in terms of how succeeding instruction depends on preceding instruction
  - RAW: Read after Write or Flow dependency
  - WAR: Write after Read or anti-dependency
  - WAW: Write after Write

Data Hazards

- Read After Write (RAW)
  Instr\(j\) tries to read operand before Instr\(i\) writes it
  - I: add r1, r2, r3
  - J: sub r4, r1, r3
- Caused by a “Dependence” (in compiler nomenclature). This hazard results from an actual need for communication.

False Data Hazards

- Write after Read (WAR)
  Instr\(j\) tries to write operand before Instr\(i\) reads it
  - I: add r1, r2, r3
  - J: mul r2, r5, r6
- Caused by a “register dependence” (in compiler nomenclature) – can be removed at compile time by assigning different registers
  - Assign different register to output of instruction J: mul r7, r5, r6
Internal Forwarding: Getting rid of some hazards

- In some cases the data needed by the next instruction at the ALU stage has been computed by the ALU (or some stage defining it) but has not been written back to the registers.
- Can we “forward” this result by bypassing stages?

Data Hazard on R1

Control Hazards: Branches

- Instruction flow
  - Stream of instructions processed by Inst. Fetch
  - Speed of “input flow” puts bound on rate of outputs generated
- Branch instruction affects instruction flow
  - Do not know next instruction to be executed until branch outcome known
- When we hit a branch instruction
  - Need to compute target address (where to branch)
  - Resolution of branch condition (true or false)
  - Might need to ‘flush’ pipeline if other instructions have been fetched for execution

Forwarding to Avoid Data Hazard
Control Hazard on Branches
Three Stage Stall

Solution?

- Branch prediction “algorithms”
- Implemented in hardware
- Use history to predict a branch
  - Example: for loop
    - Branch always taken except for last iteration

Is this how real processors look? NO…more stuff..

- Pipelining is first step….next is Instruction Level parallelism (ILP)
  - What if we had many pipeline units?
- ILP is transparent to the user
  - Multiple operations executed in parallel even though the system is handed a single program written with a sequential processor in mind
  - Same execution hardware as a normal RISC machine
    - May be more than one of any given type of hardware

Architectures for ILP

Scalar Pipeline (baseline)
Instruction Parallelism = \( D \)
Operation Latency = 1
Peak IPC = 1 (IPC: Instructions Per Cycle)

\[
\begin{array}{ccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
IF & DE & EX & WB & \hline
2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \hline
\end{array}
\]
Superscalar (Pipelined) Execution

- IP = \( D \times N \)
- OL = 1 baseline cycles
- Peak IPC = \( N \) per baseline cycle

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c}  

Is it that simple . . .

- Opportunities (to speed up things) are more but problems become more challenging

So can SW do anything about the problems?

- This is where you get to claim SW folks are smarter than HW folks!
- Compiler can look at the entire code
  - Analyze dependencies at compile time
  - Rewrite code
  - Rearrange instructions to improve parallelism
  - Make better use of registers
  - These are all things that modern compilers do by default!

Example

1. ADD r1, r2, r3 \{1,2,3\} are dependent
2. MUL r4, r1, r2 on each other: sequential
3. ADD r2, r4, r3
4. MUL r10, r11, r12 \{4,5,6\} dependent on each other: sequential
5. ADD r14, r10, r11
6. SUB r15, r14, r12

No parallelism in code when parsing sequentially
Example

1. ADD r1, r2, r3 \{1,2,3\} are dependent
2. MUL r4, r1, r2 on each other: sequential
3. ADD r2, r4, r3
4. MUL r10, r11, r12 \{4,5,6\} dependent on each
5. ADD r14, r10, r11 other: sequential
6. SUB r15, r14, r12

As a group \{1,2,3\} and \{4,5,6\} are not dependent on each other…..therefore:

Example

1. ADD r1, r2, r3 \{1,2\} are independent
2. MUL r10, r11, r12
3. MUL r4, r1, r2 \{3,4\} independent
4. ADD r14, r10, r11
5. ADD r2, r4, r3 \{5,6\} independent
6. SUB r15, r14, r12

Now we have parallelism in code

So are iLP processors the real thing. .

• NO!
• Even more techniques:
• Have you written programs with multiple threads (in Java)?
• Question: can we run threads in parallel?
• Now we enter the realm of multi-core processors
• Problems become even more challenging but opportunities for performance improvement explode!

Multithreaded Processing

<table>
<thead>
<tr>
<th>Fine Grain</th>
<th>Coarse Grain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>Thread 3</td>
</tr>
<tr>
<td>Thread 2</td>
<td>Thread 4</td>
</tr>
<tr>
<td>idle slot</td>
<td>idle slot</td>
</tr>
</tbody>
</table>
Simultaneous Multi-threading...

One thread, 8 units

Two threads, 8 units

Time (processor cycle)

Now this is some real seriou stuff... but NO – this is not yet the kcka** stuff

And NOW we are talking real stuff...

Intel Xeon Dual-core

- Dual-core Intel Xeon processors
- Each core is hyper-threaded
- Private L1 caches
- Shared L2 caches
Next: Did we forget a key part of a computer system?

- Key component in a computer?
- Memory

- How are “real” memory systems organized?
  - How do they affect performance?