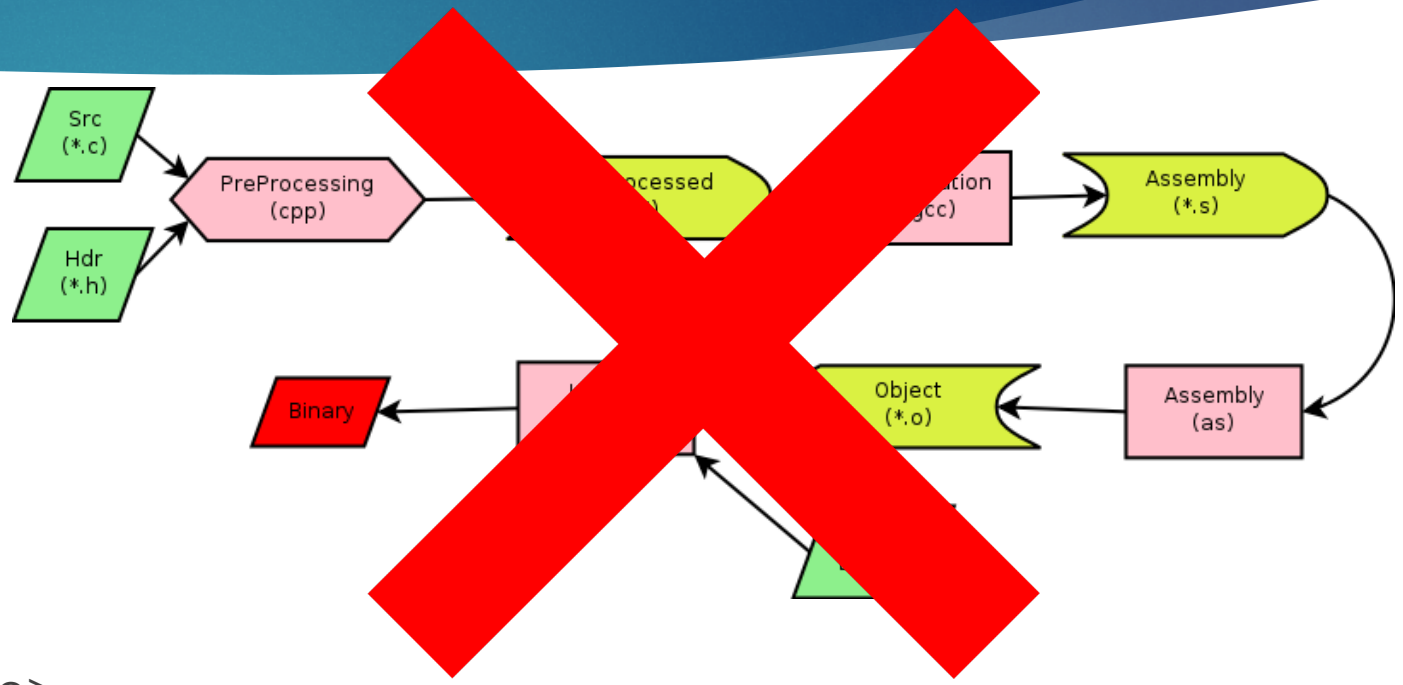
Processor/Compiler Optimizations: The Road to Faster Applications

BESHOI GENIDY

What this talk isn't about ☹

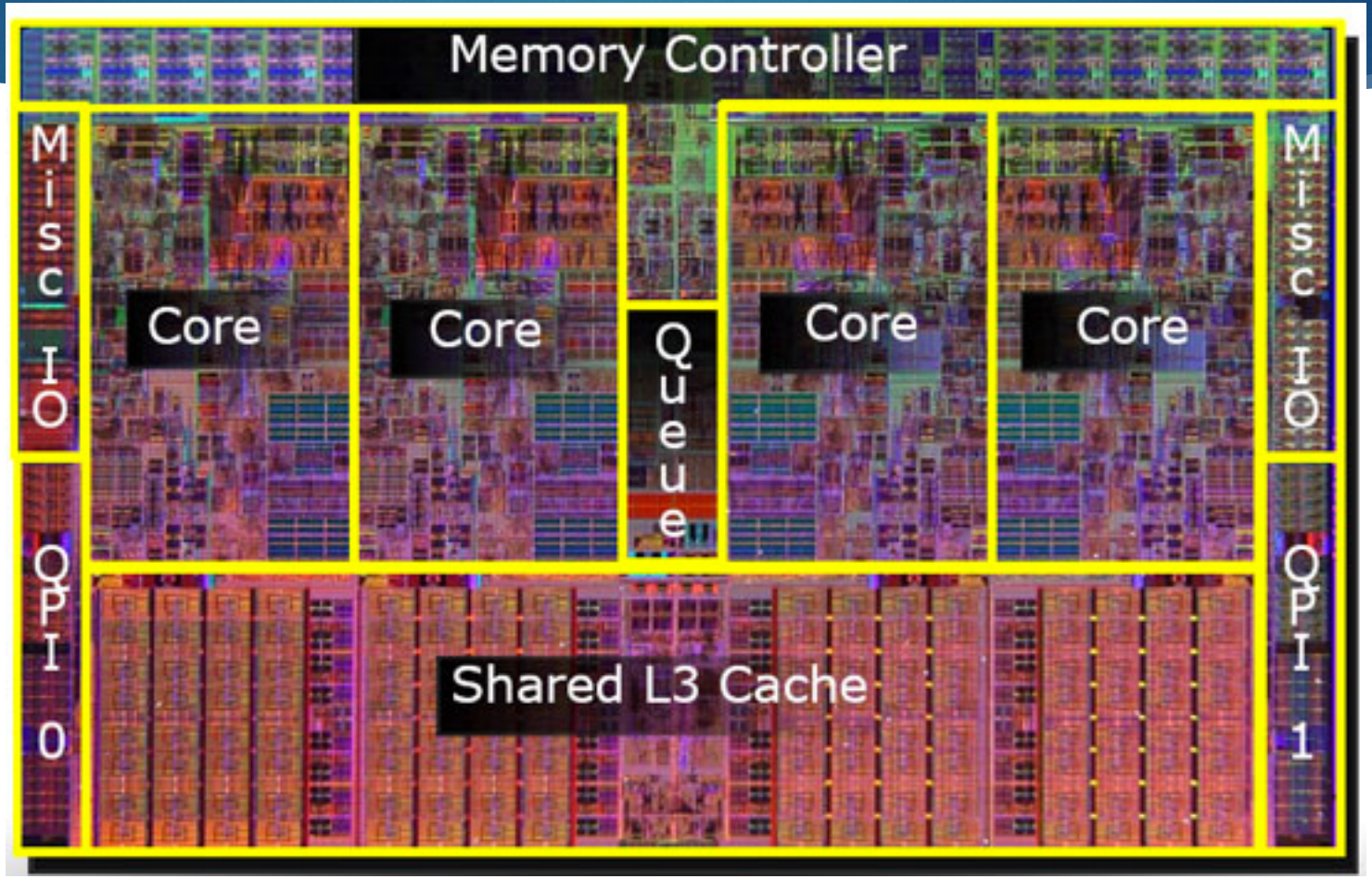
- How do compilers compile
- Caches
- Just in Time Execution
- Increased clock speed
- More transistors
- Dataflow architectures
- <Insert other important topic here>



Motivations for this Talk

Its on the edge of the Software/Hardware paradigm!!!

Processors

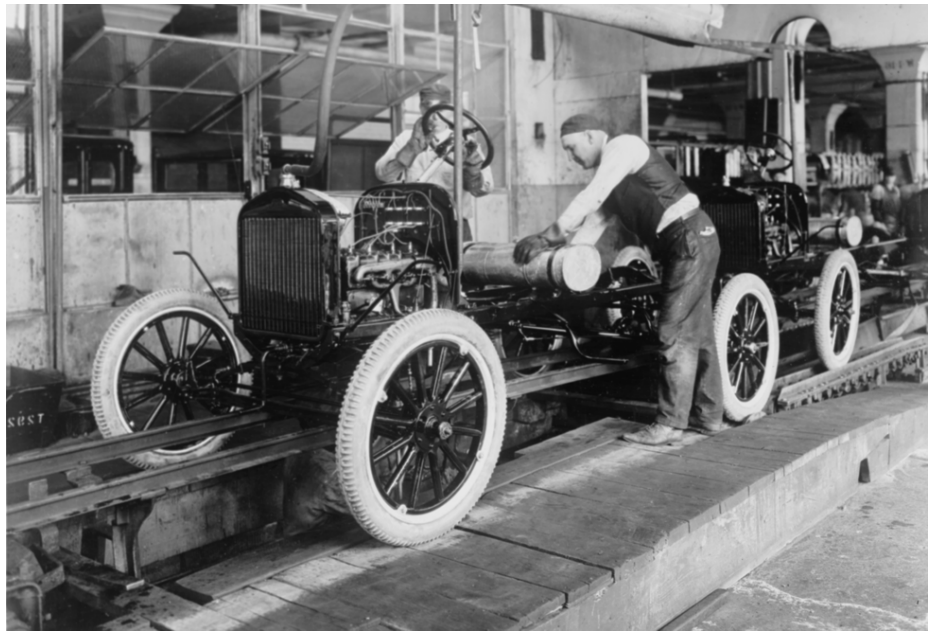

Memory Controller
Misc IO
Core
Core
Queue
Core
Core
Misc IO
QPI 0
Shared L3 Cache
QPI 1

Processor Optimization Topics

- Pipelines
- Superscalar
- Out-of-order execution
- Register Renaming
- Branch Prediction
- Speculative execution
- ILP (Instruction Level Parallelism)

What are Pipelines

- Pipeline is a set of data processing elements connected in series, where the output of one element is the input of the next one.
- A typical 5 stage pipeline consists of the following 5 stages: IF (instruction fetch), ID (instruction decode), EX (execute), MEM (memory access), WB (write back)
- Think of computer pipelines as the equivalence of the assembly line. Due to there use throughput is much higher!!!



https://en.wikipedia.org/wiki/Classic_RISC_pipeline

Pipeline drawbacks

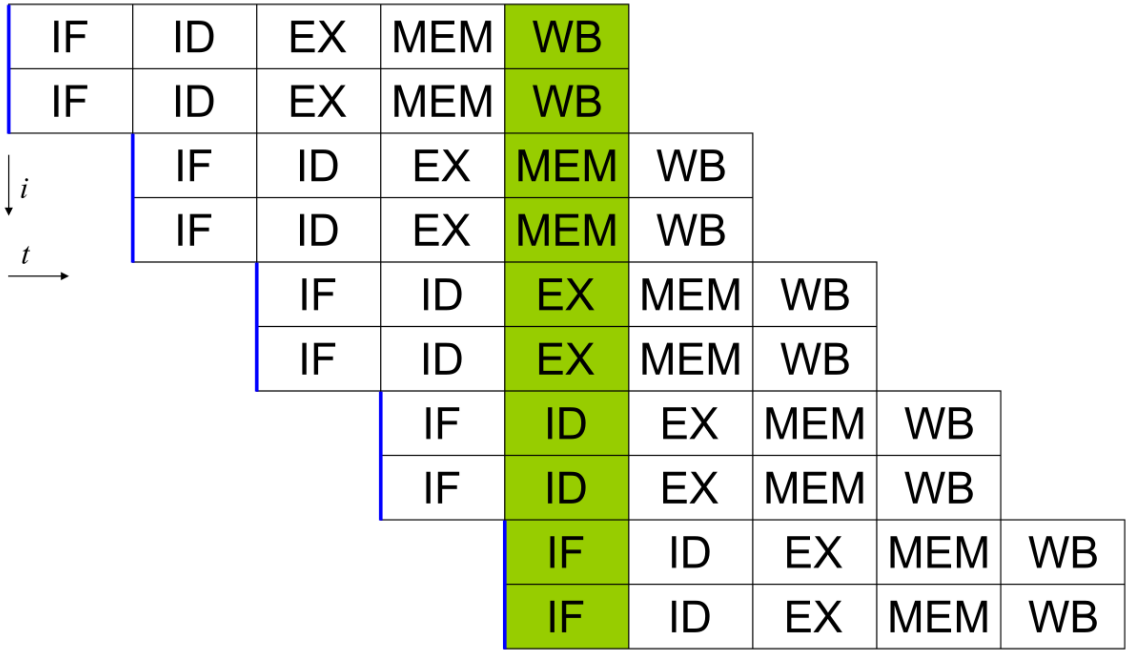
- Hazards!!!!!!!!!

Example Assembly:

1. Add 1 to R5
2. Copy R5 to R6

Superscalar processors

- It is a form of parallelism!
- They can execute more than one instruction during a clock cycle by dispatching multiple instructions to different execution units.
- Important to realize this is not referring to multi-core processing or software level parallelism but is referring to a single CPU core having multiple pipelines for increased throughput.
- Also important to realize that while superscalar CPU is typically also pipelined they are not the same thing.



https://en.wikipedia.org/wiki/Superscalar_processor

Out-of-order Execution

- Commonly referred to as dynamic execution
- Rather than doing sequential execution of instructions which often times results in stalls do to data dependencies, the processor attempts to reorder some of the operations to execute instructions based on availability (i.e. if an instruction can be executed without any harm and the proceeding instructions aren't ready yet then rearrange the order of the ops to have that instruction go first).

Example:

1. Load R1, 0(R2)
2. Add R2, R1, R3
3. Add R4, R3, R5

http://hpca23.cse.tamu.edu/taco/utsa-www/cs5513-fall07/lecture5.html

Register Renaming

- Is the technique that eliminates the false data dependencies arising from the reuse of architectural registers by successive instructions that do not have any real data dependencies between them.
- Register renaming increases ILP!!!

Consider the following piece of code:

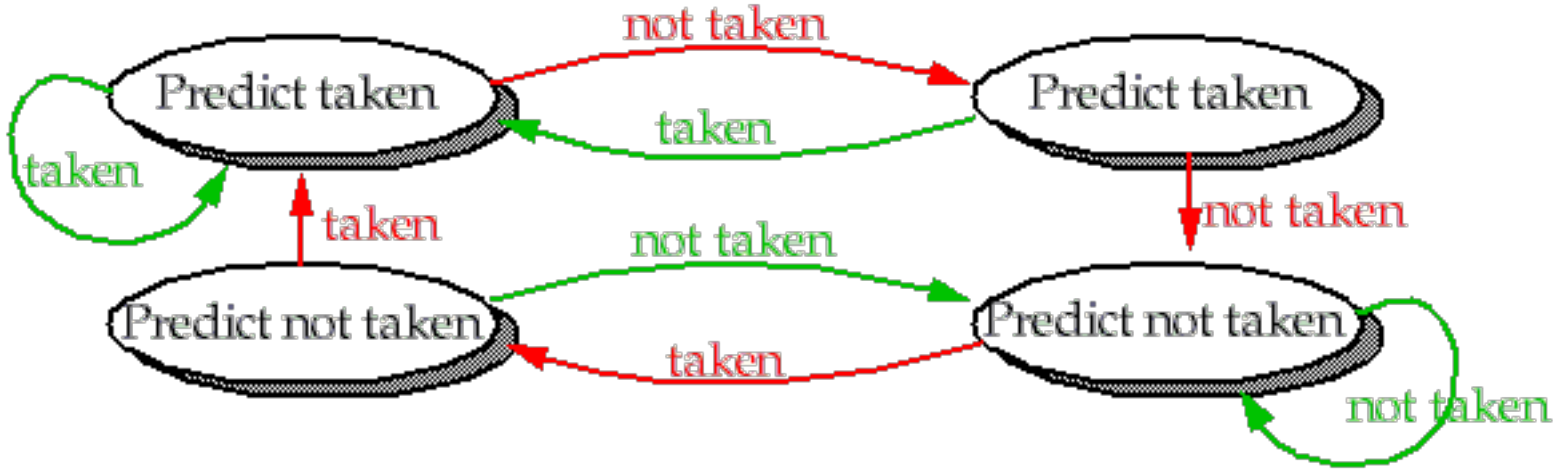
1. R1 = M[1024]
2. R1 = R1 + 2
3. M[1032] = R1
4. R1 = M[2048]
5. R1 = R1 + 4
6. M[2056] = R1

Can you rename some of these registers to eliminate false data dependencies?

https://en.wikipedia.org/wiki/Register_renaming

Branch Prediction

- A simple as it sounds (at least the concept is ☺)
- Computer architecture attempts to predict if a branch will be taken or not.



http://ece-research.unm.edu/jimp/611/slides/chap4_5.html

What is speculative execution

- Work is done before it is known whether it is actually needed, to prevent delays that would have to be incurred by doing the work after it is known that it is needed.
- If the work is found as not needed then simply discard the results and revert back all changes. (thought to be ok in a pre SPECTRE and MELTDOWN world)

Example:

```
t = a+b
u = t+c
v = u+d
if v:
        w = e+f
        x = w+g
        y = x+h
```

https://www.raspberrypi.org/blog/why-raspberry-pi-isnt-vulnerable-to-spectre-or-meltdown/

ILP (Instruction Level Parallelism)

- Two approaches to ILP
- Hardware
  - Dynamic parallelism (i.e at runtime the processor decides which instructions should be execute in parallel)
- Software
  - Static parallelism (i.e at compile time the compiler decides which instructions should be executed in parallel)
- ILP is very application specific (it depends on how much each piece of data in your application relies on one another)

Example Program:

1. e = a + b
2. f = c + d
3. m = e * f

Step 1 and Step 2 don't rely on each other hence can be executed in parallel!!

So in an ILP capable system the following program would take 2 cycles to complete rather than 3 cycles (assuming that each operation takes 1 cycle to complete)

https://en.wikipedia.org/wiki/Instruction-level_parallelism

Compilers

"Computes are dumb actors and compilers/programmers are the master playwrights"

"A large part of modern out of order processors is hardware that could have been eliminated if a good compiler existed…. A large part of modern out of order processors was designed because computer architects thought compiler writers could not do a good job."

Compiler logic, hmmmmmm: replace while(1) to for(;;) …. I have optimized all the codes

www.cs.utexas.edu/users/mckinley/pl...2007/presentations/.../panel-plw-2007.ppt

Compiler Optimization Topics

- Algebraic simplifications
- Loops
  - Invariant code (also known as code hoisting)
  - Unrolling
- Function Inlining
- Dead code elimination
- Global optimizations

A bit about compilers…

- Must preserve the semantic equivalence of the programs.
- Algorithm should not be modified.
- Transformations should hopefully make the program faster.
- When in doubt be as conservative as possible (back to the first point).

- Compilers have machine dependent and machine independent optimizations.

www.iitg.ernet.in/dgoswami/resource/Code-Optimization.ppt

Algebraic simplifications/Constant folding

- Replace costly operations with simpler ones. For example (16*x) should be replaced with (x<<4)
- Constant folding is the evaluation of constant expressions at compile time rather than computing them at runtime. For example (x = 54*12*17) would be replaced with (x = 11016)

https://en.wikipedia.org/wiki/Constant_folding

Code Hoisting

- Statements or expressions that can be moved outside of the body of a loop without affecting the semantics of the program.

Example:

```
for (int i = 0; i < n; i++) {
    x = y + z;
    a[i] = 6 * i + x * x;
}
```

https://en.wikipedia.org/wiki/Loop-invariant_code_motion

Loop Unrolling

- Is a loop transformation technique that attempts to optimize program execution speed at the expense of its binary size.
- Optimization works by expanding the loop so less loop control (loop iterations) instructions are executed, obviously this will result in a larger code base.

Normal loop	After loop unrolling

`int x;`
`for (x = 0; x < 100; x++)`
`{`
`  delete(x);`
`}`
	`int x;`
	`for (x = 0; x < 100; x += 5 )`
	`{`
	`  delete(x);`
	`  delete(x + 1);`
	`  delete(x + 2);`
	`  delete(x + 3);`
	`  delete(x + 4);`
	`}`

https://en.wikipedia.org/wiki/Loop_unrolling

Function inlining

- is the replacement of a function call with the actual body of the function.

```
int add(int a, int b) {
        return a+b;
}

int main(void) {
        int x = add(4, 5);
        return 0;
}
```

```
int main(void) {
        int x = 4 + 5;
        return 0;
}
```

Dead code elimination

- The removal of code which does not affect the program results.

Benefits:

- include faster computation time due to less operations to execute.
- Smaller application size due to the removal of the unneeded code.

```
int foo(void)
{
  int a = 24;
  int b = 25; /* Assignment to dead variable */
  int c;
  c = a * 4;
  return c;
  b = 24; /* Unreachable code */
  return 0;
}
```

https://en.wikipedia.org/wiki/Dead_code_elimination

Global optimizations

- Optimizations can be extended to an entire control-flow graph.
- Usually done on function levels with worst case assumptions regarding outside calls or use of global variables.

Optimize the following…

```
int foo(int *a, int *b) {
        *a = 2;
        *b = 3;
        return *a+*b;
}
```

Future of compilers

"Optimization for scalar machines is a problem that was solved ten years ago." – David Kuck, Fall 1990

Maybe for 1990 standards but in todays world a few things are evolving:

- Architectures keep changing
- Languages keep changing
- Applications keep changing
- When to compile keeps changing

www.cs.utexas.edu/users/mckinley/380C/lecs/01.pp

Questions?


NOT SURE IF I DID A GREAT JOB
OR NO ONE PAID ATTENTION
VIA 9GAG.COM