Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Lecture 13:

Implementing a Processor 5/5

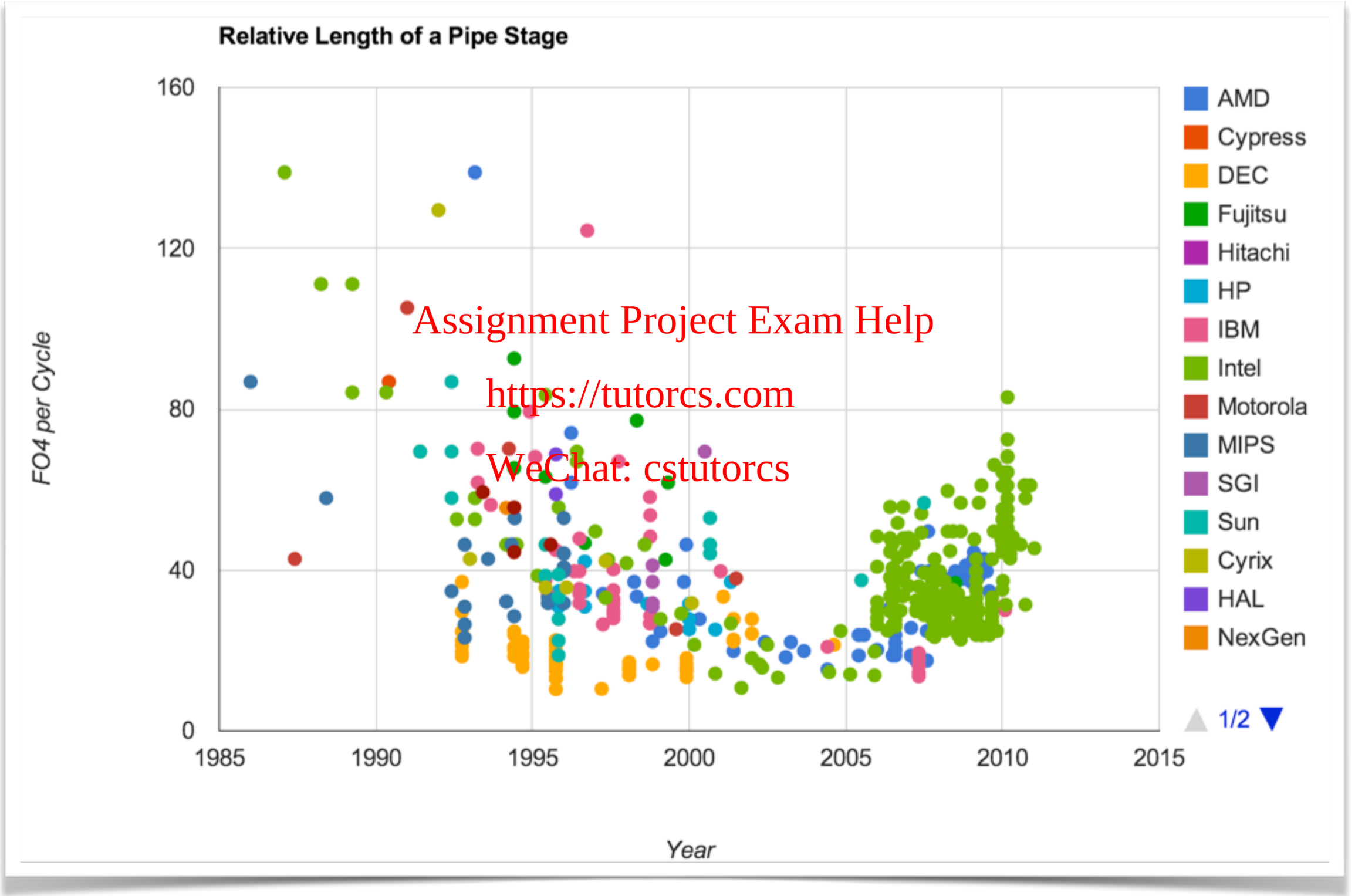
Introduction to Computer Architecture
UC Davis EEC 170, Fall 2019

Superpipelined vs. Superscalar

- Superpipelined processors have longer instruction latency (in terms of cycles) than the SS processors, which can degrade performance in the presence of true dependencies
  - Note we're improving throughput at the expense of latency!
- Superscalar processors are more susceptible to resource conflicts —but we can fix this with hardware!

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Hardware limits to superpipelining?



Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

http://cpudb.stanford.edu/visualize/fo4_per_cycle

Does Multiple Issue Work?

The BIG Picture

- Yes, but not as much as we'd like
- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
  - e.g., pointer aliasing
- Some parallelism is hard to expose
  - Limited window size during instruction issue
- Memory delays and limited bandwidth
  - Hard to keep pipelines full
- Speculation (next slide) can help if done well

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Speculation

- "Guess" what to do with an instruction
  - Start operation as soon as possible
  - Check whether guess was right
    - If so, complete the operation
    - If not, roll back and do the right thing
- Common to static and dynamic multiple issue
- Examples
  - Speculate on branch outcome
    - Roll back if path taken is different
  - Speculate on load
    - Roll back if location is updated

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Compiler/Hardware Speculation

- Compiler can reorder instructions
  - e.g., move load before branch
  - Can include "fix-up" instructions to recover from incorrect guess
- Hardware can look ahead for instructions to execute
  - Buffer results until it determines they are actually needed
  - Flush buffers on incorrect speculation

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Context for VLIW

- Last lecture we looked at how hardware can (re)schedule instructions at runtime
- Now we'll look at instruction scheduling done at compile time ("static") using a compiler

Assignment Project Exam Help

https://tutorcs.com

WeChat: cstutorcs

VLIW Beginnings

- **VLIW: Very Long Instruction Word**

[4] J. A. Fisher, "Very long instruction word architectures and the ELI-512," in *Proc. 10th Symp. Comput. Architecture*, IEEE, June 1983, pp. 140–150.

Assignment Project Exam Help

- **Josh Fisher: idea grew out of his Ph.D (1979) in compilers**

https://tutorcs.com

- **Led to a startup (Multiflow) whose computers worked, but which went out of business … the ideas remain influential.**

WeChat: cstutorcs

History of VLIW Processors

- Started with (horizontal) microprogramming
  - Very wide microinstructions used to directly generate control signals in single-issue processors (e.g., IBM 360 series)
- VLIW for multi-issue processors first appeared in the Multiflow and Cydrome (in the early 1980's)
- Recent commercial VLIW processors
  - Intel i860 RISC (dual mode: scalar and VLIW)
  - Intel I-64 (EPIC: Itanium and Itanium 2) [later today]
  - Transmeta Crusoe [also later today]
  - Lucent/Motorola StarCore, ADI TigerSHARC, Infineon (Siemens) Carmel

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Static Multiple Issue Machines (VLIW)

- Static multiple-issue processors (aka VLIW) use the compiler to decide which instructions to issue and execute simultaneously
  - Issue packet—the set of instructions that are bundled together and issued in one clock cycle—think of it as one large instruction with multiple operations
  - The mix of instructions in the packet (bundle) is usually restricted—a single "instruction" with several predefined fields
  - The compiler does static branch prediction and code scheduling to reduce (ctrl) or eliminate (data) hazards

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Static Multiple Issue Machines (VLIW)

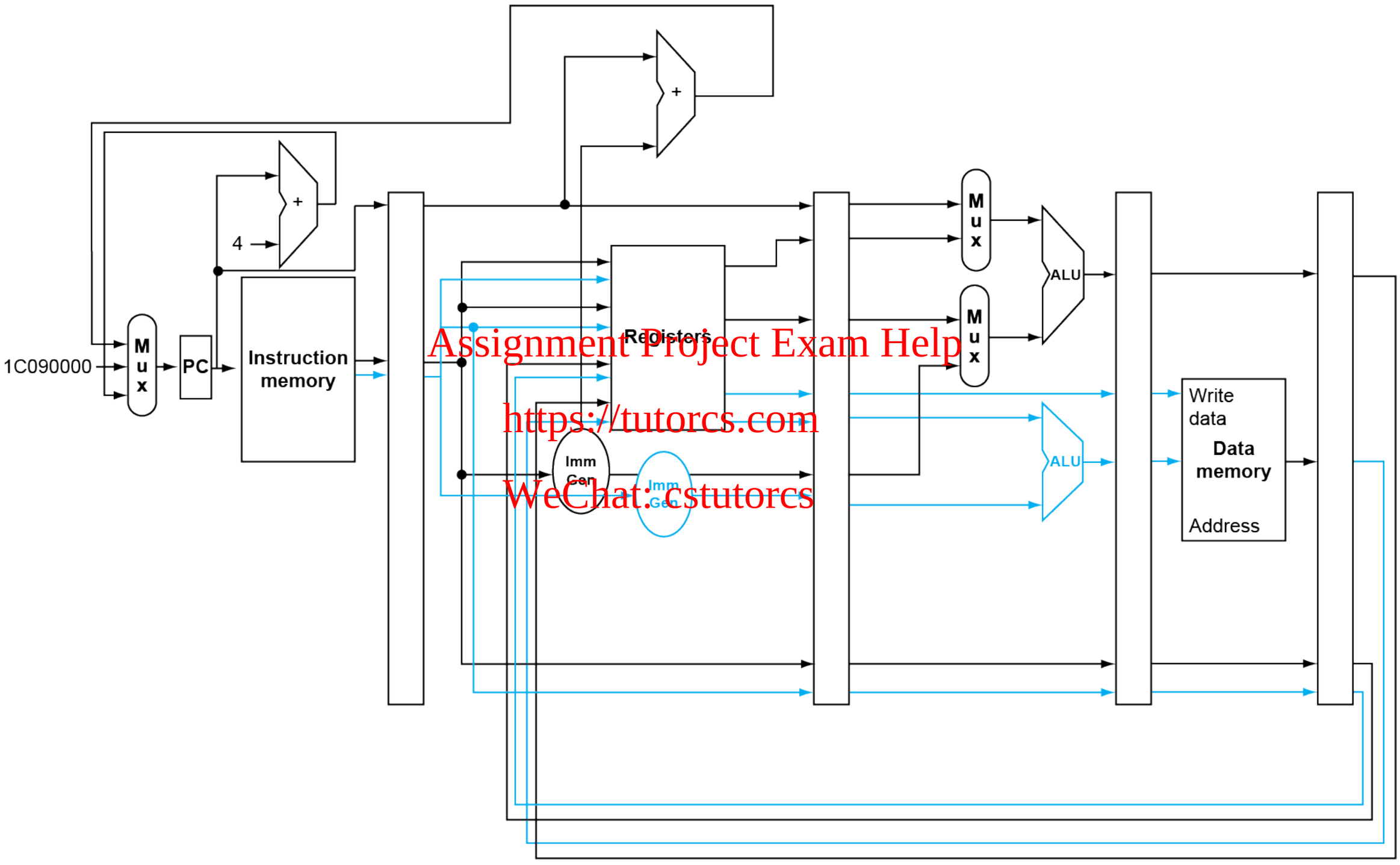
- VLIWs have
  - Multiple functional units (like SS processors)
  - Multi-ported register files (again like SS processors)
  - Wide program bus

Assignment Project Exam Help

https://tutorcs.com

WeChat: cstutorcs

RISC-V with Static Dual Issue



Assignment Project Exam Help
https://tutorcs.com
WeChat:cstutorcs

Scheduling Example

- **Schedule this for dual-issue RISC-V**

```
Loop: ld   x31,0(x20)     // x31=array element
      add  x31,x31,x21    // add scalar in x21
      sd   x31,0(x20)     // store result
      addi x20,x20,-8     // decrement pointer
      blt  x22,x20,Loop   // branch if x22 < x20
```

Assignment Project Exam Help
https://tutorcs.com

	ALU/branch	Load/store	cycle
Loop:	nop	ld x31,0(x20)	1
	addi x20,x20,-8	nop	2
	add x31,x31,x21	nop	3
	blt x22,x20,Loop	sd x31,8(x20)	4

WeChat: cstutorcs

- **IPC = 5/4 = 1.25 (c.f. peak IPC = 2)**

Loop Unrolling

- Loop unrolling—multiple copies of the loop body are made and instructions from different iterations are scheduled together as a way to increase ILP
- Apply loop unrolling (4 times for our example) and then schedule the resulting code
  - Eliminate unnecessary loop overhead instructions
  - Schedule so as to avoid load use hazards
- During unrolling the compiler applies register renaming to eliminate all data dependencies that are not true dependencies

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Unrolled Code Example

This is a slightly different loop than the one a few slides ago but your instructor is lazy and did not want to rewrite it.

```
lp: lw $t0,0($s1)      # $t0=array element
    lw $t1,-4($s1)     # $t1=array element
    lw $t2,-8($s1)     # $t2=array element
    lw $t3,-12($s1)    # $t3=array element
    add $t0,$t0,$s2    # add scalar in $s2
    add $t1,$t1,$s2    # add scalar in $s2
    add $t2,$t2,$s2    # add scalar in $s2
    add $t3,$t3,$s2    # add scalar in $s2
    sw $t0,0($s1)      # store result
    sw $t1,-4($s1)     # store result
    sw $t2,-8($s1)     # store result
    sw $t3,-12($s1)    # store result
    addi $s1,$s1,-16   # decrement pointer
    bne $s1,$0,lp      # branch if $s1 != 0
```

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

The Scheduled Code (Unrolled)

	ALU or branch	Data transfer	CC
lp:	addi $s1,$s1,-16	lw $t0,0($s1)	1
		lw $t1,12($s1)	2
	add $t0,$t0,$s2	lw $t2,8($s1)	3
	add $t1,$t1,$s2	lw $t3,4($s1)	4
	add $t2,$t2,$s2	sw $t0,16($s1)	5
	add $t3,$t3,$s2	sw $t1,12($s1)	6
		sw $t2,8($s1)	7
	bne $s1,$0,lp	sw $t3,4($s1)	8

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

- Eight clock cycles to execute 14 instructions for a
  - CPI of 0.57 (versus the best case of 0.5)
  - IPC of 1.8 (versus the best case of 2.0)

What does N = 14 assembly look like?

- Two instructions from a scientific benchmark (Linpack) for a MultiFlow CPU with 14 operations per instruction.

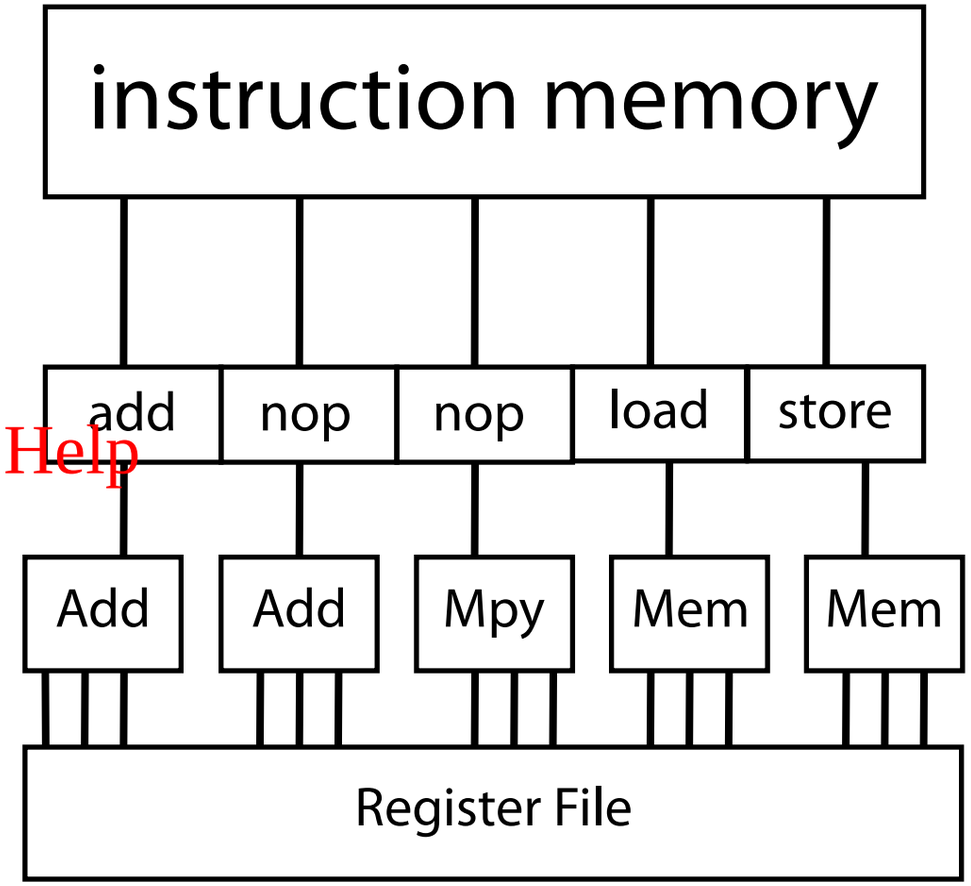
```
instr   cl0     ialu0e   st.64      sb1.r0,r2,17#144
        cl0     ialu1e   cgt.s32    li1bb.r4,r34,6#31
        cl0     falu0e   add.f64    lsb.r4,r8,r0
        cl0     falu1e   add.f64    lsb.r6,r40,r32
        cl0     ialu0l   dld.64     fb1.r4,r2,17#208
        cl1     ialu0e   dld.64     fb1.r34,r1,17#216
        cl1     ialu1e   cgt.s32    li1bb.r3,r32,zero
        cl1     falu0e   add.f64    lsb.r4,r8,r6
        cl1     falu1e   add.f64    lsb.r6,r40,r38
        cl1     ialu0l   st.64      sb1.r2,r1,17#152
        cl1     ialu1l   add.u32    lib.r32,r36,6#32
        cl1     br       true and r3  L23?3
        cl0     br       false or r4  L24?3;

instr   cl0     ialu0e   dld.64     fb0.r0,r2,17#224
        cl0     ialu1e   cgt.s32    li1bb.r3,r34,6#30
        cl0     falu0e   mpy.f64    lfb.r10,r2,r10
        cl0     falu1e   mpy.f64    lfb.r42,r34,r42
        cl0     ialu0l   st.64      sb0.r4,r2,17#160
        cl1     ialu0e   dld.64     fb0.r32,r1,17#232
        cl1     ialu1e   cgt.s32    li1bb.r4,r35,6#29
        cl1     falu0e   mpy.f64    lfb.r10,r0,r10
        cl1     falu1e   mpy.f64    lfb.r42,r32,r42
        cl1     ialu0l   st.64      sb0.r6,r1,17#168
        cl1     ialu1l   bor.32     ib0.r32,zero,r32
        cl1     br       false or r4  L25?3
        cl0     br       true and r3  L26?3;
```

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Defining Attributes of VLIW

- **Compiler:**
  1. **MultiOp: instruction containing multiple independent operations**
  2. **Specified number of resources of specified types**
  3. **Exposed, architectural latencies**

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs



VLIW instruction = 5 independent operations

Compiler Support for VLIW Processors

- The compiler packs groups of independent instructions into the bundle
  - Because branch prediction is not perfect, done by code re-ordering (trace scheduling)
- The compiler uses loop unrolling to expose more ILP
- The compiler uses register renaming to solve name dependencies and ensures no load use hazards occur

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Compiler Support for VLIW Processors

- While superscalars use dynamic prediction, VLIWs primarily depend on the compiler for extracting ILP
  - Loop unrolling reduces the number of conditional branches
  - Predication eliminates if-the-else branch structures by replacing them with predicated instructions
    - We'll cover this shortly
- The compiler predicts memory bank references to help minimize memory bank conflicts

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

VLIW Advantages

- Advantages
  - Simpler hardware (potentially less power hungry)
  - Potentially more scalable
    - Allow more instr's per VLIW bundle and add more FUs

Assignment Project Exam Help

https://tutorcs.com

WeChat: cstutorcs

VLIW Disadvantages

- Programmer/compiler complexity and longer compilation times
  - Deep pipelines and long latencies can be confusing (making peak performance elusive)
- Lock step operation, i.e., on hazard all future issues stall until hazard is resolved (hence need for predication)
- Object (binary) code incompatibility
- Needs lots of program memory bandwidth
- Code bloat
  - Noops are a waste of program memory space
  - Loop unrolling to expose more ILP uses more program memory space

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Review: Multi-Issue Datapath Responsibilities

- Must handle, with a combination of hardware and software
  - Data dependencies – aka data hazards
    - True data dependencies (read after write)
      - Use data forwarding hardware
      - Use compiler scheduling
    - Storage dependence (aka name dependence)
      - Use register renaming to solve both
        - Antidependencies (write after read)
        - Output dependencies (write after write)

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Review: Multi-Issue Datapath Responsibilities

- Must handle, with a combination of hardware and software
  - Procedural dependencies—aka control hazards
    - Use aggressive branch prediction (speculation) (next)
    - Use predication (future lecture)

Assignment Project Exam Help

https://tutorcs.com

WeChat: cstutorcs

Review: Multi-Issue Datapath Responsibilities

- Must handle, with a combination of hardware and software
  - Resource conflicts—aka structural hazards
    - Use resource duplication or resource pipelining to reduce (or eliminate) resource conflicts
    - Use arbitration for result and commit buses and register file read and write ports

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Review: Multiple-Issue Processor Styles

- Dynamic multiple-issue processors (aka **superscalar**)
  - Decisions on which instructions to execute simultaneously (in the range of 2 to 8 in 2005) are being made dynamically (at run time by the **hardware**)
  - E.g., IBM Power 2, Intel x86, MIPS R10K, HP PA 8500 IBM
- Static multiple-issue processors (aka **VLIW**)
  - Decisions on which instructions to execute simultaneously are being made statically (at compile time by the **compiler**)
  - E.g., Intel Itanium and Itanium 2 for the IA-64 ISA – EPIC (Explicit Parallel Instruction Computer)
    - 128 bit "bundles" containing 3 instructions each 41 bits + 5 bit template field (specifies which FU each instr needs)
    - Five functional units (IntALU, MMedia, DMem, FPALU, Branch)
    - Extensive support for speculation and predication

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Avoiding branches

- **Consider the following code:**

```
// x is either 0 or 1
if (x == 0) {
  a = b;
} else if (x == 1) {
  a = c;
}
```

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

- **How many instrs does this take on average (as is)?**
- **Write this code with no branches (4 instructions)**
  - **Why is this a useful exercise?**

Conditional move

- **Consider a "conditional move" instruction:**

  `CMOVZ dst, src, cond` ***// copies src to dst if cond != 0***

  - **MIPS, Alpha, PowerPC, SPARC, x86 (Pentium) have this**
  - **RISC-V does not (we'll talk about this shortly)**

Assignment Project Exam Help

-
```
// x is either 0 or 1
if (x == 0) {
  a = b;
} else if (x == 1) {
  a = c;
}
```

https://tutorcs.com

WeChat: cstutorcs

- **Write this code with no branches (2 instructions)**

Predication

- Predication can be used to eliminate branches by making the execution of an instruction dependent on a "predicate", e.g.,

```
if (p) { statement 1 } else { statement 2 }
```

would normally compile using two control-flow instructions. (Why?) With predication it would compile as

Assignment Project Exam Help
https://tutorcs.com

```
(p) statement 1
```

WeChat: cstutorcs

```
(~p) statement 2
```

- The use of `(condition)` indicates that the instruction is committed only if `condition` is true
- Predication can be used to speculate as well as to eliminate branches

Predication Main Idea

- Convert *control* dependence to *data* dependence

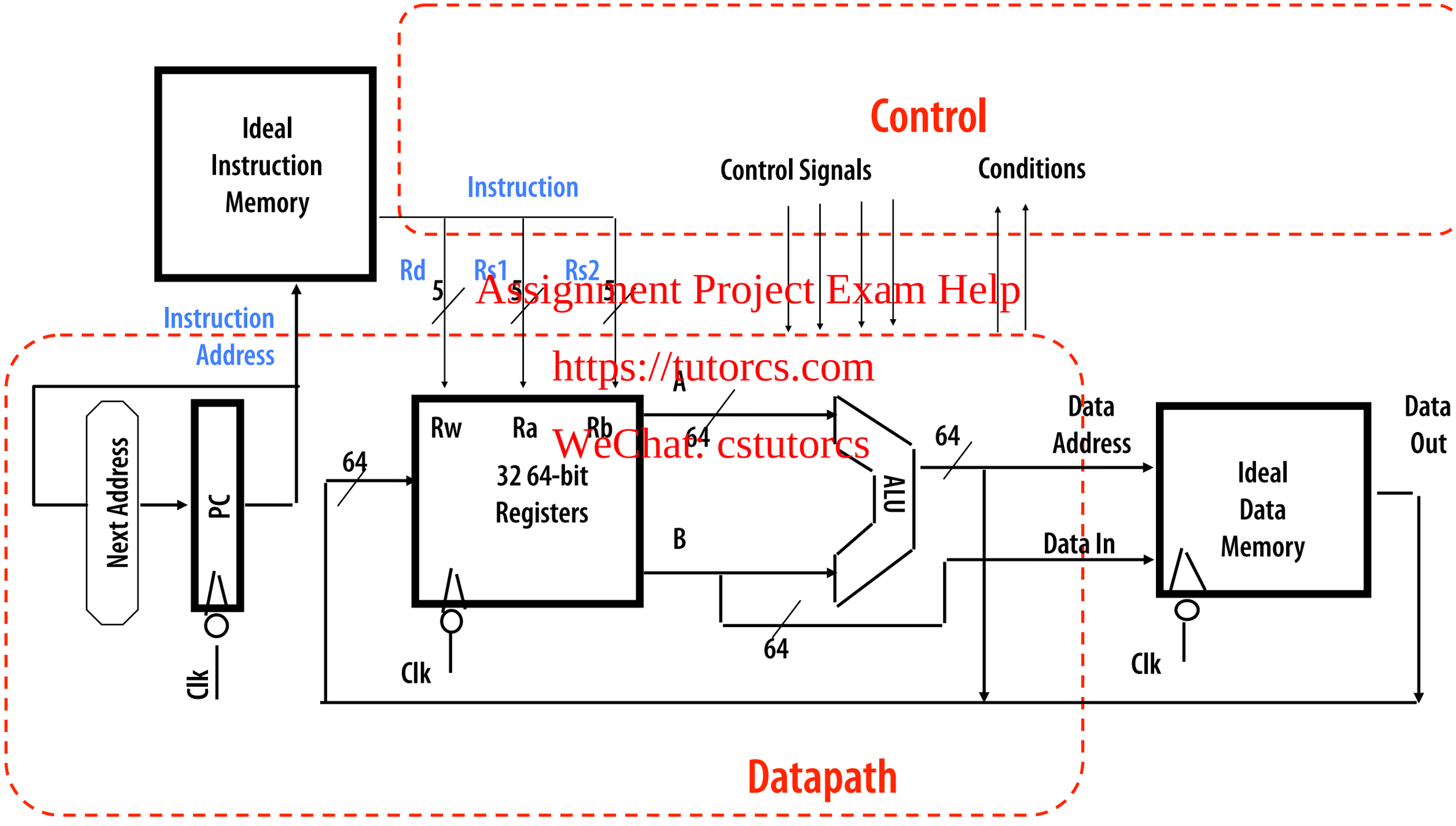
- Why is this better?

Assignment Project Exam Help

https://tutorcs.com

WeChat: cstutorcs

How do we support predication in hw?



Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Where to evaluate predicate

- Can we evaluate the predicate early in the pipeline (annul the predicated instruction), before it gets to the ALUs?

- Or should we evaluate the predicate (annul the predicated instruction) late in the pipeline, after the operation has been through the ALU?

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Predicates Are Good

- Implementing short alternative control flows
- Eliminating unpredictable branches
- Reducing the overhead of global code scheduling

Assignment Project Exam Help

https://tutorcs.com

WeChat: cstutorcs

Predicates Are Bad

- Annulled predicated instructions still take resources
- If the predicate is evaluated late, it might cause a data hazard
- What about executing an operation across multiple branches?
- Possible speed penalty

Assignment Project Exam Help

https://tutorcs.com

WeChat: cstutorcs

RISC-V designers on conditional moves/predication

- "We consciously omitted support for conditional moves and predication. Both enable some form of if-conversion, a transformation by which some control hazards can be traded for data hazards. Conditional move instructions are much weaker than predication: they add to the critical code path, and they cannot in general be used to if-convert instructions that might cause exceptions, like loads and stores. Full predication is much more general, but adds to the architectural state and consumes substantial opcode space, as each instruction must be given an additional predicate operand. Both techniques complicate implementations with register renaming, since the old value of the destination register must be copied to the new physical register when the predicate is false. Finally, if-conversion is usually not profitable in the common case that the condition is predictable: branch prediction will succeed, sometimes with higher performance, since it obviates the extra data dependence."

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

EPIC Goal

- Support compiler-based exploitation of ILP
  - Predication
  - Compiler-based parallelism detection
  - Support for memory reference speculation
  - …

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

How EPIC extends VLIW

- Greater flexibility in indicating parallelism between instructions & within instruction formats
  - VLIW has a fixed instruction format
    - All ops within instr must be parallel
  - EPIC has more flexible instruction formats
    - EPIC indicates parallelism between neighboring instructions
- Extensive support for software speculation

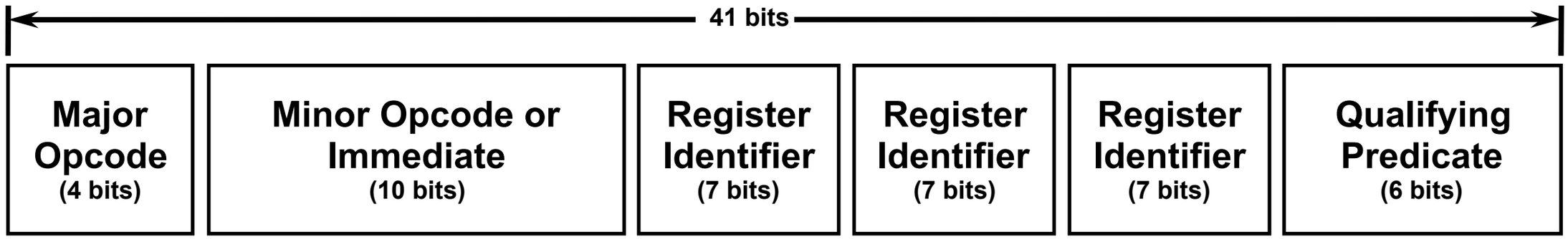
Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Intel/HP IA-64 "Explicitly Parallel Instruction Computer (EPIC)"

- IA-64: instruction set architecture
- 128 64-bit integer regs + 128 82-bit floating point regs
  - Not separate register files per functional unit as in some VLIW architectures
- Hardware checks dependencies (interlocks => binary compatibility over time)
- Predicated execution (select 1 out of 64 1-bit flags) => 40% fewer mispredictions?

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

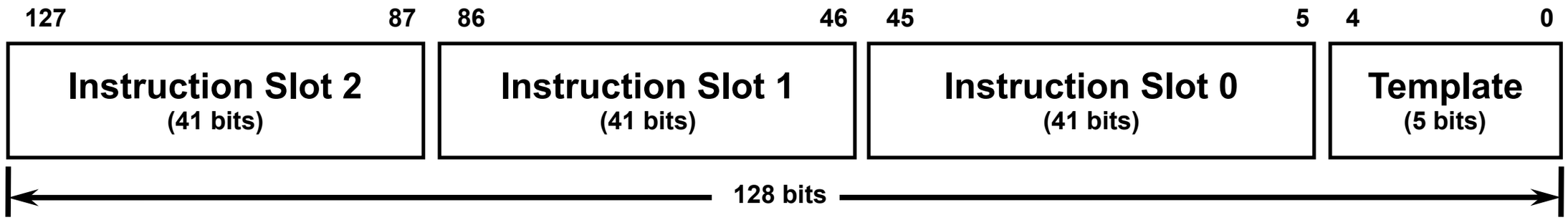
EPIC Instruction Format



- Major opcode (4 bits)
- Minor opcode
- Immediate operands (8–22 bits)
- Register result identifier(s) (6 or 7 bits)
- Register operand identifiers (7 bits)
- Qualifying predicates (6 bits)
  - A few instructions do not have a QP (nearly all do!)

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Instruction Formats: Bundles



Template identifies types of instructions in bundle and delineates independent operations (through "stops")

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

- **Instruction types**
  - **M: Memory**
  - **I: Shifts and multimedia**
  - **A: ALU**
  - **B: Branch**
  - **F: Floating point**
  - **L+X: Long**
- **Template encodes types**
  - **MII, MLX, MMI, MFI, MMF, MI_I, M_MI**
  - **Branch: MIB, MMB, MFB, MBB, BBB**
- **Template encodes parallelism**
  - **All come in two flavors: with and without stop at end**

EPIC Rules

Execution unit slot	Instruction type	Instruction description	Example instructions
I-unit	A	Integer ALU	add, subtract, and, or, compare
	I	Non-ALU integer	integer and multimedia shifts, bit tests, moves
M-unit	A	Integer ALU	add, subtract, and, or, compare
	M	Memory access	Loads and stores for integer/FP registers
F-unit	F	Floating point	Floating-point instructions
B-unit	B	Branches	Conditional branches, calls, loop branches
L + X	L + X	Extended	Extended immediates, stops and no-ops

Figure G.6 The five execution unit slots in the IA-64 architecture and what instructions types they may hold are shown. A-type instructions, which correspond to integer ALU instructions, may be placed in either an I-unit or M-unit slot. L + X slots are special, as they occupy two instruction slots; L + X instructions are used to encode 64-bit immediates and a few special instructions. L + X instructions are executed either by the I-unit or the B-unit.

Template	Slot 0	Slot 1	Slot 2
0	M	I	I
1	M	I	I
2	M	I	I
3	M	I	I
4	M	L	X
5	M	L	X
8	M	M	I
9	M	M	I
10	M	M	I
11	M	M	I
12	M	F	I
13	M	F	I
14	M	M	F
15	M	M	F
16	M	I	B
17	M	I	B
18	M	B	B
19	M	B	B
22	B	B	B
23	B	B	B
24	M	M	B
25	M	M	B
28	M	F	B
29	M	F	B

Figure G.7 The 24 possible template values (8 possible values are reserved) and the instruction slots and stops for each format. Stops are indicated by heavy lines and may appear within and/or at the end of the bundle. For example, template 9 specifies that the instruction slots are M, M, and I (in that order) and that the only stop is between this bundle and the next. Template 11 has the same type of instruction slots but also includes a stop after the first slot. The L + X format is used when slot 1 is L and slot 2 is X.

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Evaluating Itanium

- "The EPIC approach is based on the application of massive resources. These resources include more load-store, computational, and branch units, as well as larger, lower-latency caches than would be required for a superscalar processor. Thus, IA-64 gambles that, in the future, power will not be the critical limitation, and that massive resources, along with the machinery to exploit them, will not penalize performance with their adverse effect on clock speed, path length, or CPI factors."
—M. Hopkins, 2000

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Itanium Criticism

- In a 2009 article on the history of the processor — "How the Itanium Killed the Computer Industry" — journalist John C. Dvorak reported "This continues to be one of the great fiascos of the last 50 years". Tech columnist Ashlee Vance commented that the delays and underperformance "turned the product into a joke in the chip industry". In an interview, Donald Knuth said "The Itanium approach ... was supposed to be so terrific—until it turned out that the wished-for compilers were basically impossible to write."

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

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

Intel/HP IA-64 "Explicitly Parallel Instruction Computer (EPIC)"

- Itanium™ was first implementation (2001)
  - 6-wide, 10-stage pipeline at 800 MHz on 0.18µ process
- Itanium 2™ ("Tukwila") is 2nd implementation (2005)
  - 6-wide, 8-stage pipeline at 1666 MHz on 0.13µ process
  - Caches: 32 KB I, 32 KB D, 128 KB L2I, 128 KB L2D, 9216 KB L3
  - 2008: HP pays Intel $440M to keep building Itanium from 2009–14
- Itanium 9500 ("Poulson"): Nov 2012
  - 8 cores, 12-wide, 1.73–2.53 GHz, 32 nm process, 3.1B transistors
  - Caches: 32 MB shared L3; per-core: 6 MB L2, 512 L1I, 256 L1D
- Itanium 9700 "Kittson": launched in May 2017
  - "Notably, Kittson has no microarchitecture improvements over Poulson, only higher clock speeds. … Intel has announced that the 9700 series will be the last Itanium chips produced"

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Intel to Discontinue Itanium 9700 'Kittson' Processor, the Last of the Itaniums by Anton Shilov on January 31, 2019

- Intel on Thursday notified its partners and customers that it would be discontinuing its Itanium 9700-series (codenamed Kittson) processors, the last Itanium chips on the market. Under their product discontinuance plan, Intel will cease shipments of Itanium CPUs in mid-2021, or a bit over two years from now. The impact to hardware vendors should be minimal – at this point HP Enterprise is the only company still buying the chips – but it nonetheless marks the end of an era for Intel and their interesting experiment into a non-x86 VLIW-style architecture.

Assignment Project Exam Help
https://tutorcs.com

- The current-generation octa and quad-core Itanium 9700-series processors were introduced by Intel in 2017, in the process becoming the final processors based on the IA-64 ISA. Kittson for its part was a clockspeed-enhanced version of the Itanium 9500-series 'Poulson' microarchitecture launched in 2012, and featured a 12 instructions per cycle issue width, 4-way Hyper-Threading, and multiple RAS capabilities not found on Xeon processors back then. It goes without saying that the writing has been on the wall for Itanium for a while now, and Intel has been preparing for an orderly wind-down for quite some time.

WeChat: cstutorcs

https://www.anandtech.com/show/13924/intel-to-discontinue-itanium-9700-kittson-processor-the-last-itaniums

Jon Stokes, arstechnica, "Itanium uber alles? Not so fast", 10 Nov '05

- "In the end, Itanium was designed from the ground up with ILP in mind. It turns out that in today's world, ILP is not the main factor limiting performance. The memory wall and the power wall are much bigger performance limiters, and Itanium was not designed to surmount either of those barriers. Yes, two side effects of its design are that its core is relatively small and its pipeline is short, but that by no means makes it somehow intrinsically better suited to multicore computing than a small-footprint OOOE design that needs less cache and that relies on TLP and other tricks to increase application performance."

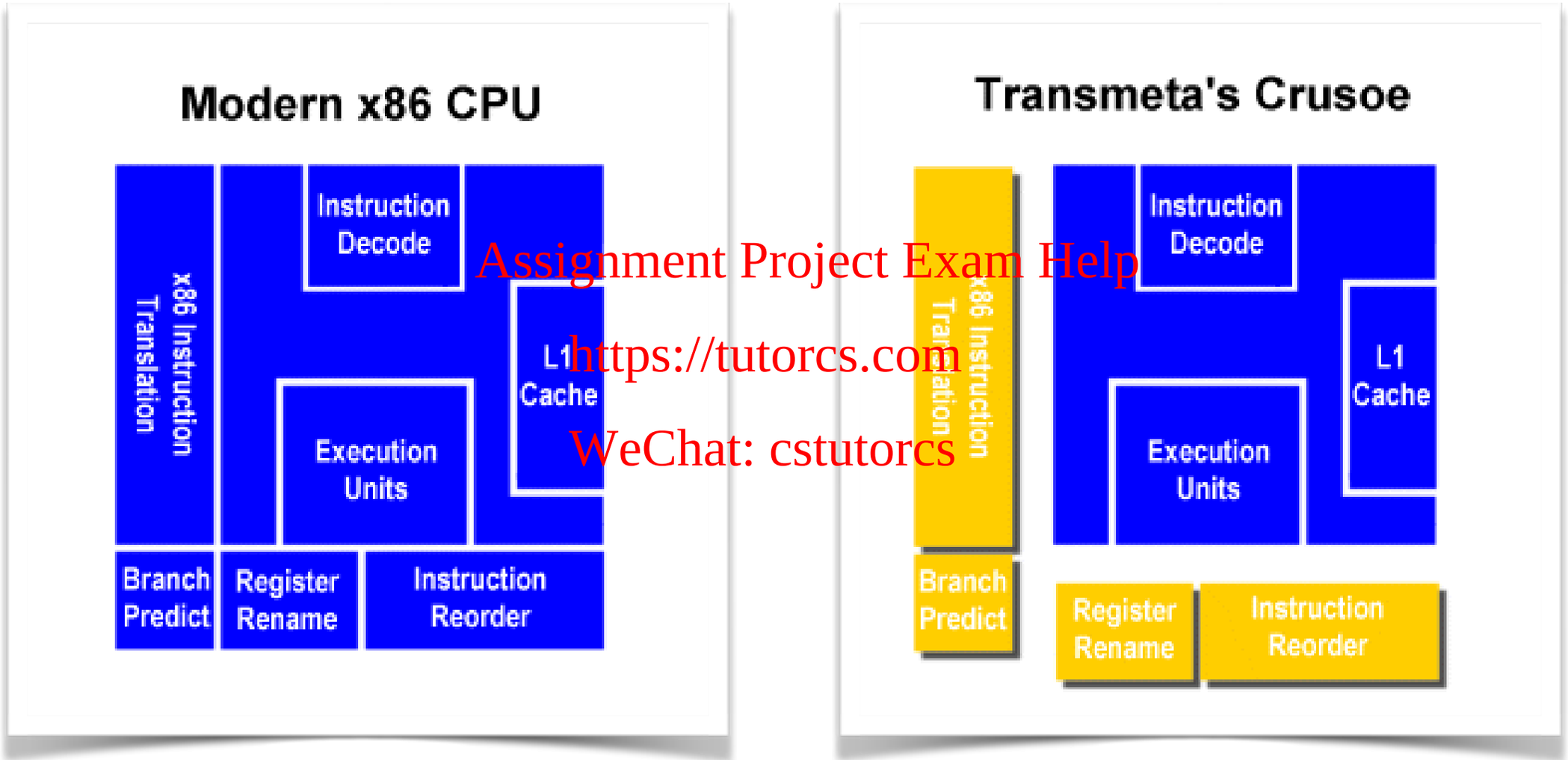
Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Transmeta motivation

- Intel/AMD goals:
  - x86 compatibility
  - Fastest performance
- Transmeta goals:
  - x86 compatibility
  - lowest possible power consumption
  - reasonable performance

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

HW vs. SW approaches



Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Crusoe is VLIW

- Functional units
  - 1 FPU
  - 2 ALUs
  - 1 load-store unit
  - 1 branch unit
- 64 registers

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

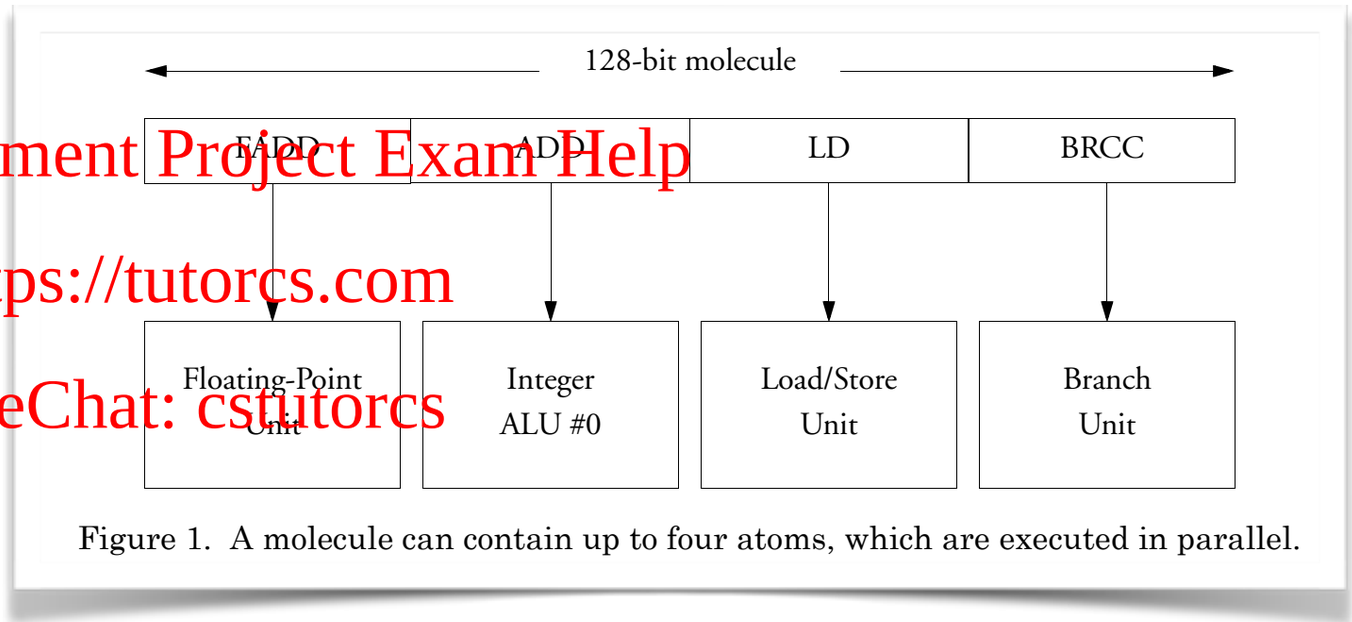
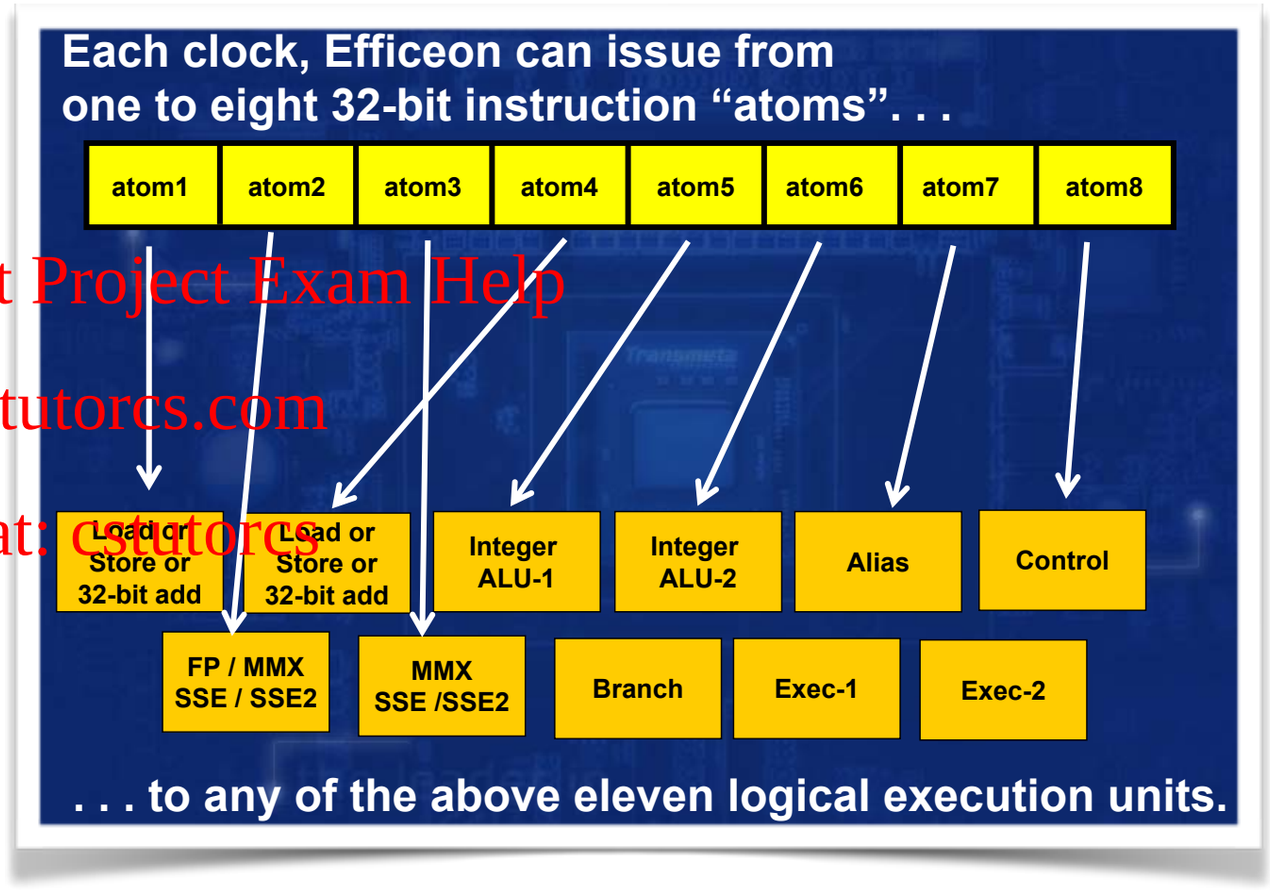


Figure 1. A molecule can contain up to four atoms, which are executed in parallel.

Efficion is VLIW

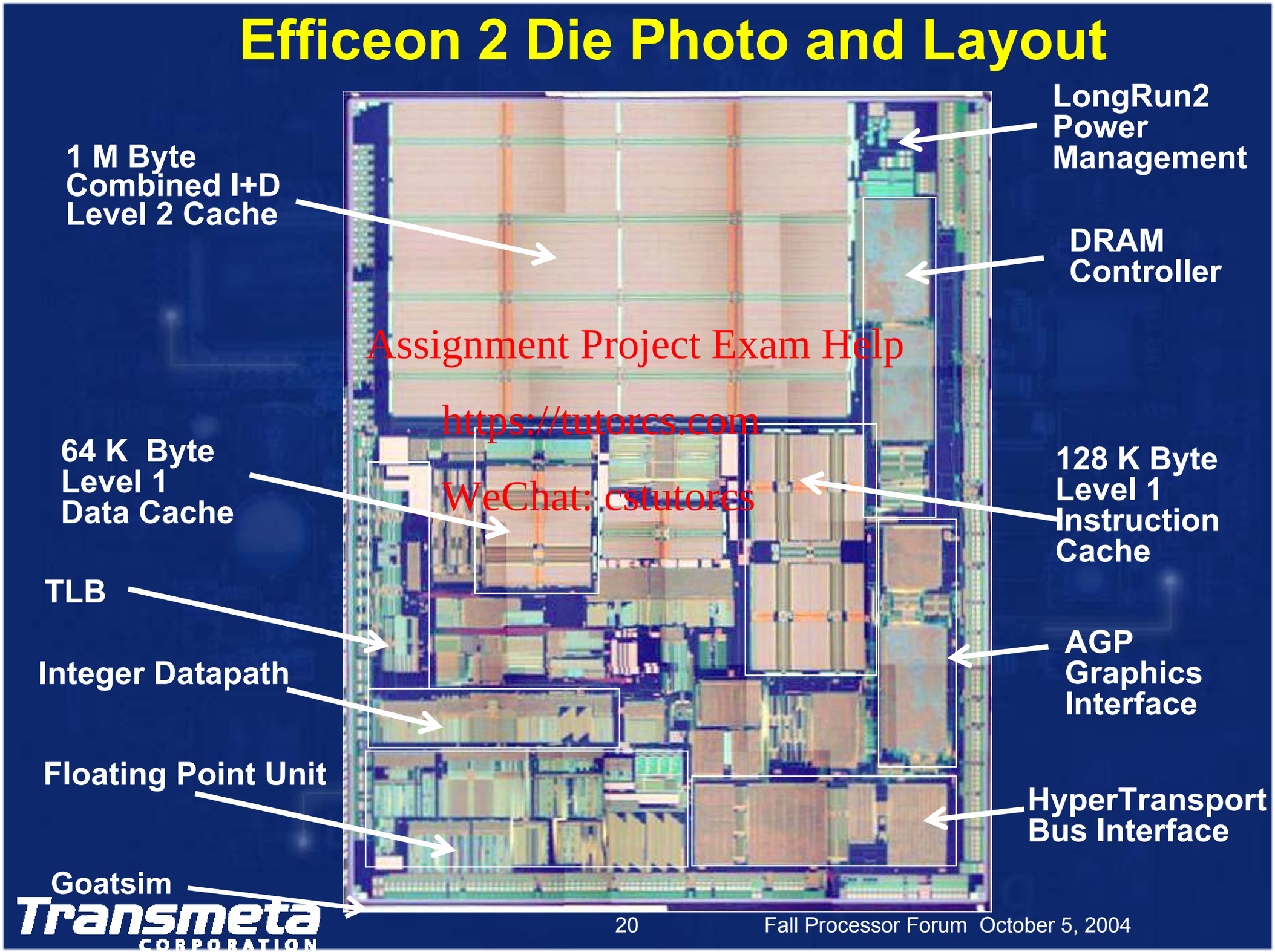
- Functional units
  - 2 FPUs
  - 2 ALUs
  - 2 load-store units
  - 2 “execute” units
  - 1 branch unit
  - 1 control unit
  - 1 alias unit
- 256b wide



David Ditzel, 2004 Fall Processor Forum

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Efficeon 2 Die Photo

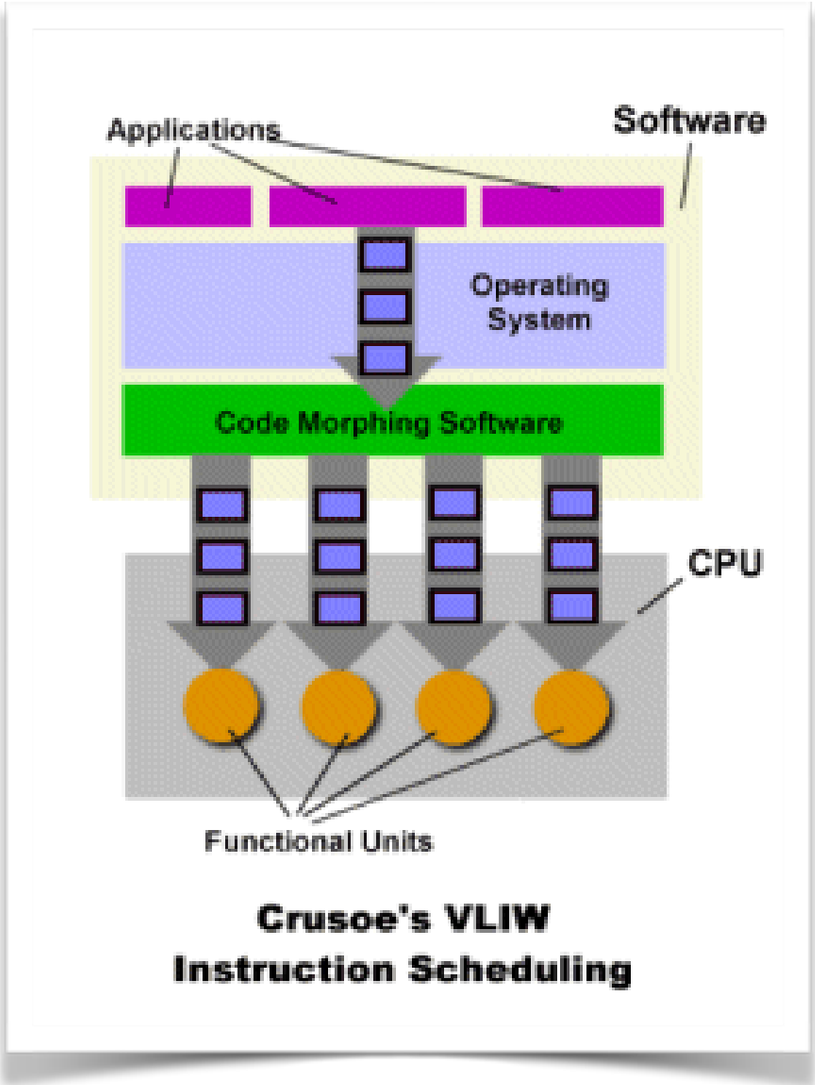


Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Code Morphing

- x86 fed to Code Morphing layer
- CM translates chunk of x86 to VLIW
  - Output stored in translation cache
- CM watches execution:
  - Frequently used chunks are more heavily optimized
  - Watches branches, can tailor speculation to history
- Some CM support in hardware

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs



Why Do Dynamic Scheduling?

- Why not just let the compiler schedule code?
- Not all stalls are predicable
  - e.g., cache misses
- Can't always schedule around branches
  - Branch outcome is dynamically determined
- Different implementations of an ISA have different latencies and hazards

Assignment Project Exam Help
https://tutores.com
WeChat: cstutorcs

Fallacies

- Pipelining is easy (!)
  - The basic idea is easy
  - The devil is in the details
    - e.g., detecting data hazards
- Pipelining is independent of technology
  - So why haven't we always done pipelining?
  - More transistors make more advanced techniques feasible
  - Pipeline-related ISA design needs to take account of technology trends
    - e.g., predicated instructions

Assignment Project Exam Help
https://tutores.com
WeChat: cstutores

Pitfalls

- Poor ISA design can make pipelining harder
  - e.g., complex instruction sets (VAX, IA-32)
    - Significant overhead to make pipelining work
    - IA-32 micro-op approach
  - e.g., complex addressing modes
    - Register update side effects, memory indirection
  - e.g., delayed branches
    - Advanced pipelines have long delay slots

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

Concluding Remarks

- ISA influences design of datapath and control
- Datapath and control influence design of ISA
- Pipelining improves instruction throughput using parallelism
  - More instructions completed per second
  - Latency for each instruction not reduced
- Hazards: structural, data, control
- Multiple issue and dynamic scheduling (ILP)
  - Dependencies limit achievable parallelism
  - Complexity leads to the power wall

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs