

Lecture 12:

Implementing a Processor 4/5

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

**Introduction to Computer Architecture
UC Davis EEC 170, Fall 2019**

Extracting More Performance

■ Two options:

- Increase the depth of the pipeline to increase the clock rate
—superpipelining
 - How does this help performance? (What does it impact in the performance equation?)
- Fetch (and execute) more than one instruction at one time
(expand every pipeline stage to accommodate multiple instructions)—multiple-issue
 - How does this help performance? (What does it impact in the performance equation?)

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

Why Do Processors Get Faster?

■ 3 reasons:

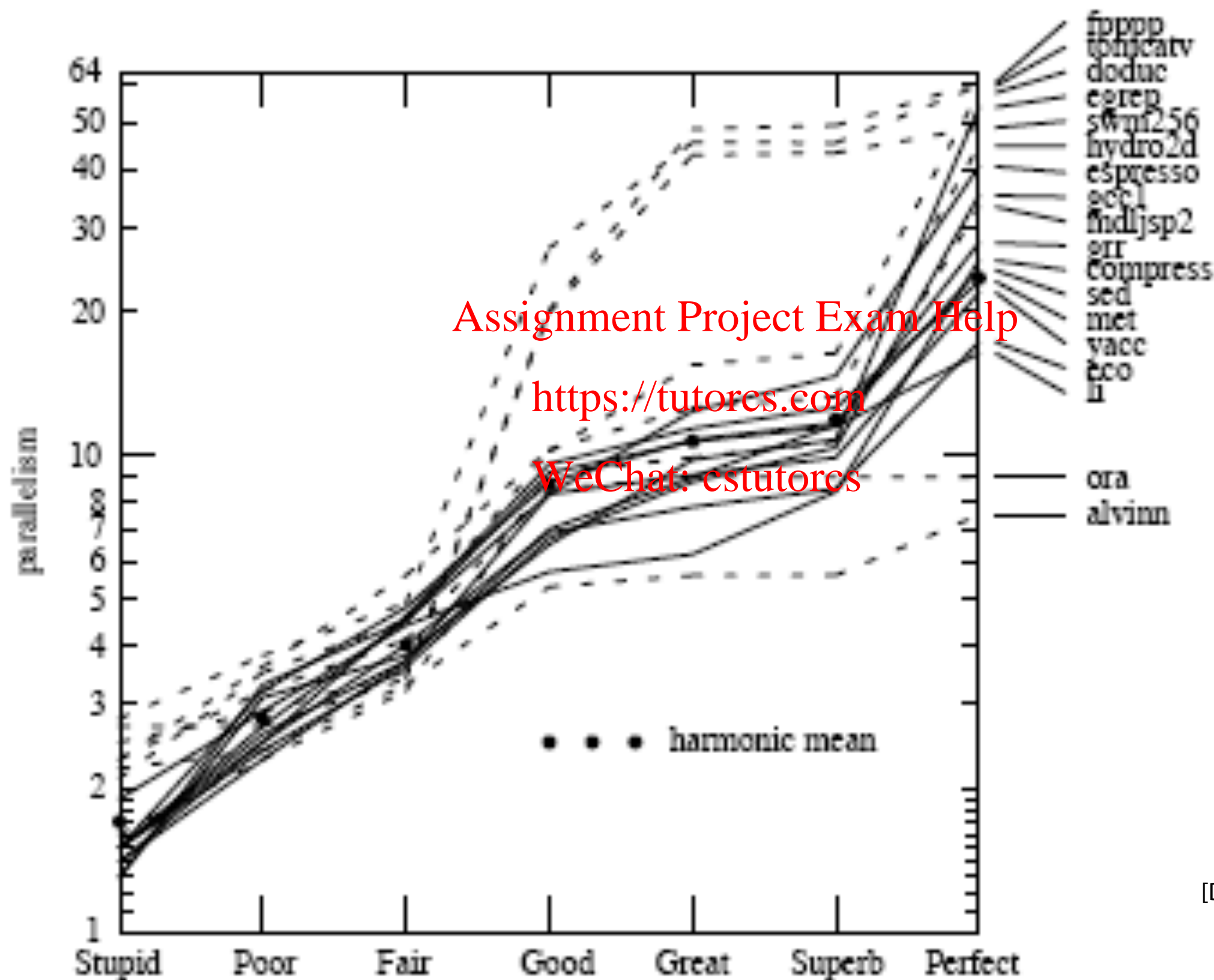
- **More parallelism (or more work per pipeline stage): fewer clocks/instruction [more instructions/cycle]**
 - **Get WIDER**
- **Deeper pipelines: fewer gates/clock**
 - **Get DEEPER**
- **Transistors get faster (Moore's Law): fewer ps/gate**
 - **Get FASTER**

Assignment Project Exam Help

<https://tutorcs.com>

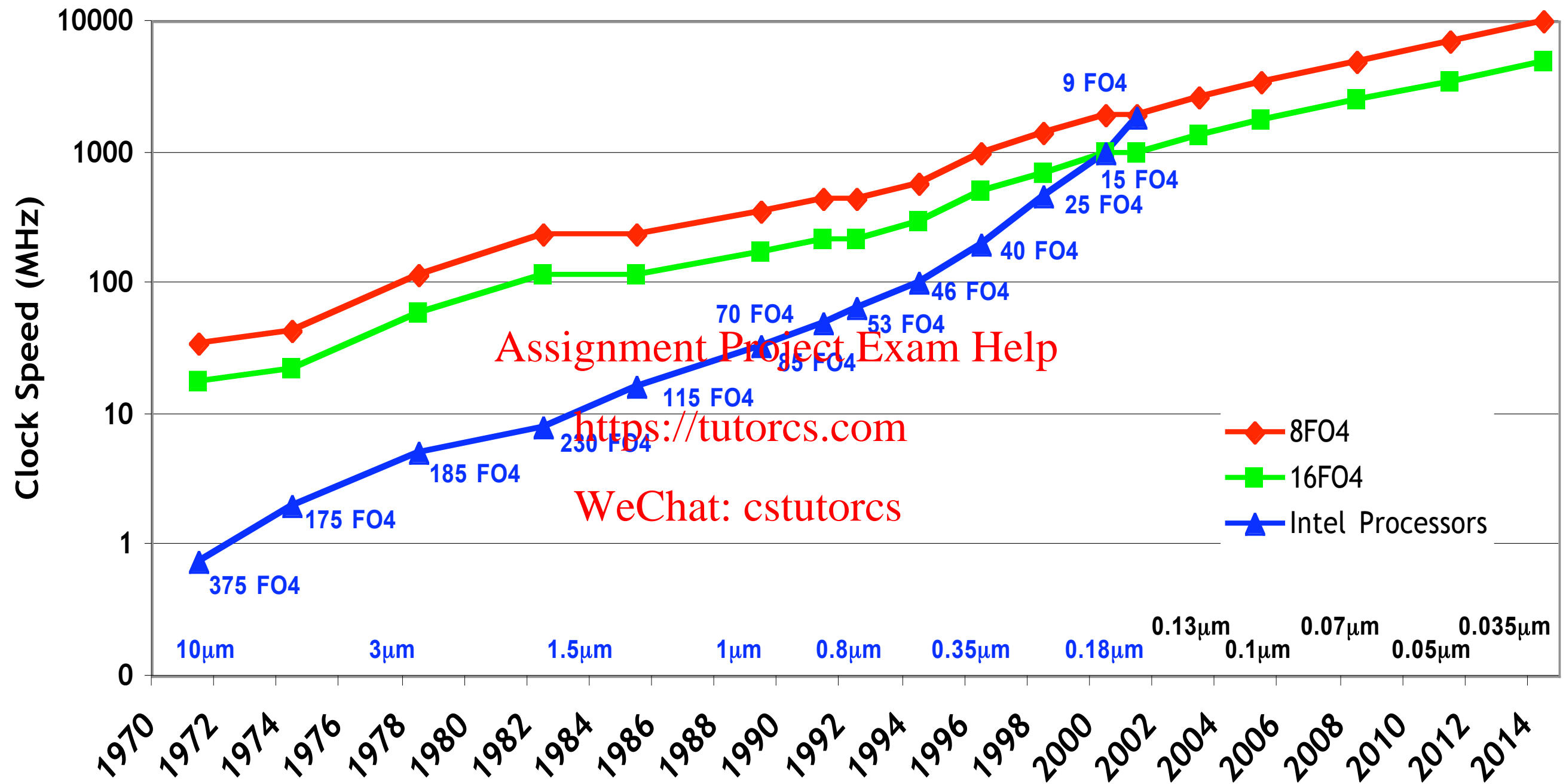
WeChat: cstutorcs

Limits to Instruction Level Parallelism



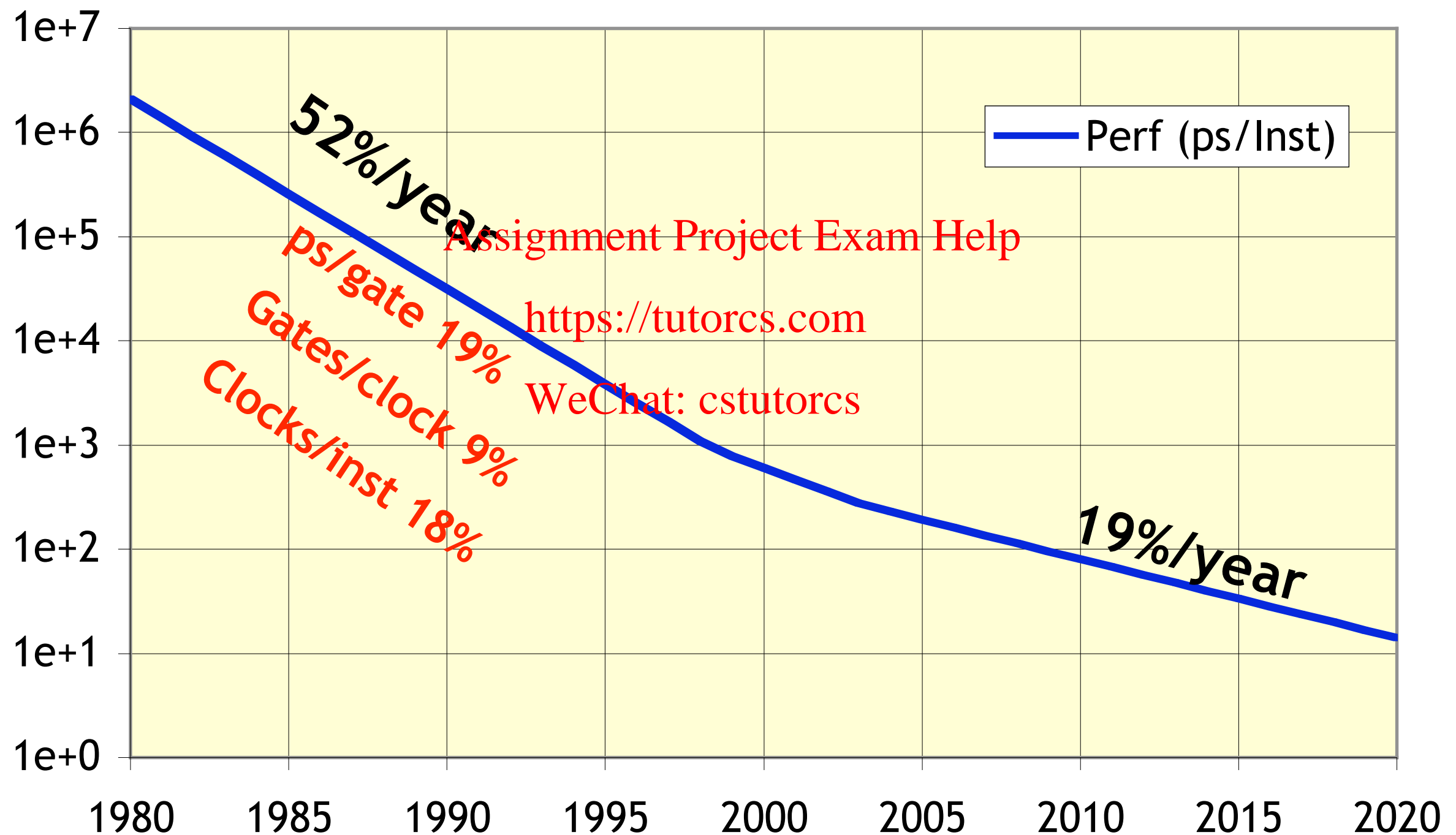
[David Wall, *Limits of Instruction-Level Parallelism*, WRL Research Report 93/6

Clock Scaling: Historical and Projected



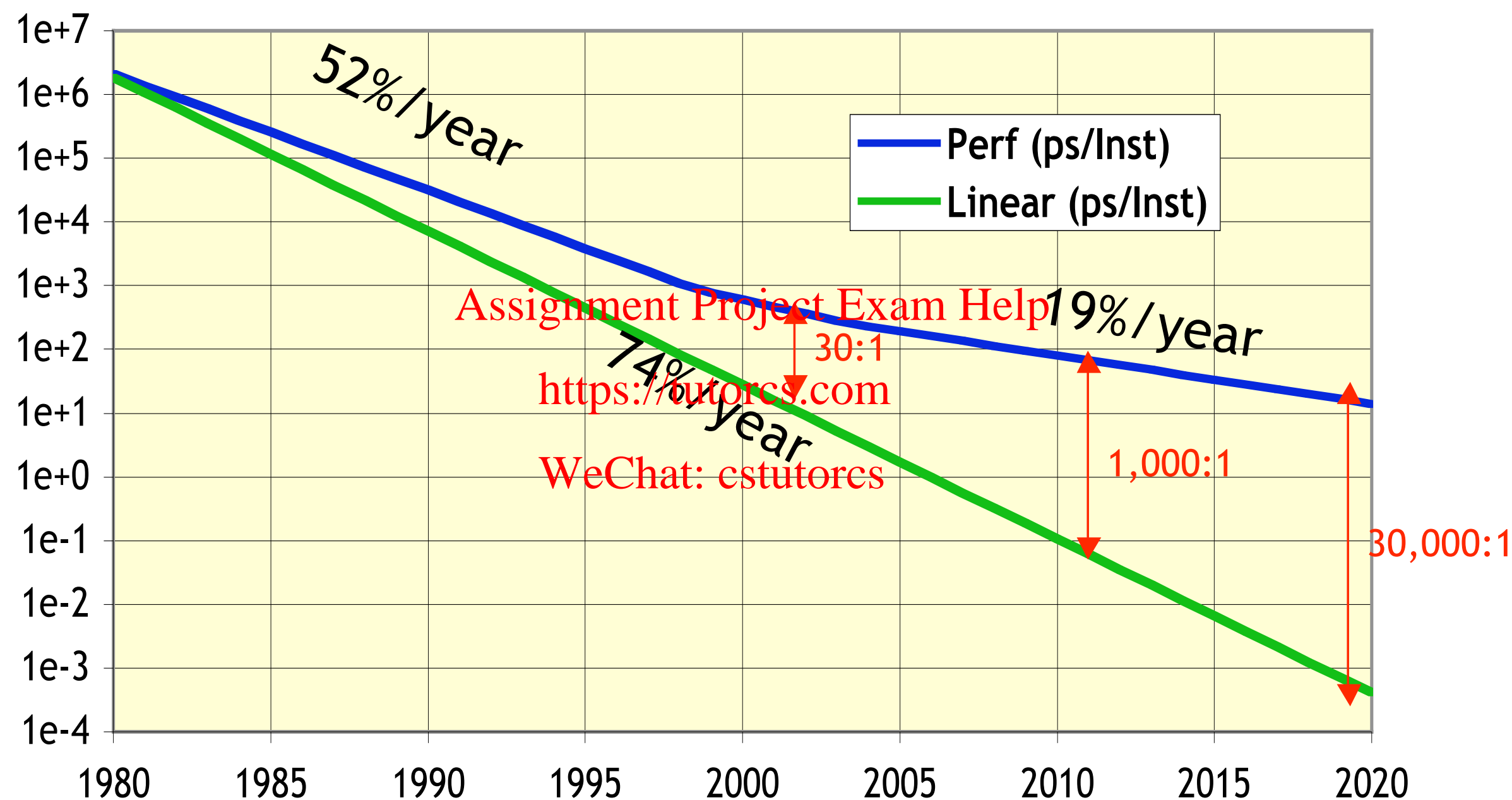
- “The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays” (ISCA02) [courtesy Steve Keckler]

Microprocessor Scaling Has Slowed



[courtesy of Bill Dally]

Future Potential is Large



[courtesy of Bill Dally]

Extracting More Performance with IPC

- Launching multiple instructions per stage allows the instruction execution rate, CPI, to be less than 1
 - So instead we use IPC: instructions per clock cycle
 - e.g., a 3 GHz, four-way multiple-issue processor can execute at a peak rate of 12 billion instructions per second with a best case CPI of 0.25 or a best case IPC of 4
 - If the datapath has a five stage pipeline, how many instructions are active in the pipeline at any given time?
 - How might this lead to difficulties?

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Superpipelined Processors

- **Increase the depth of the pipeline leading to shorter clock cycles (and more instructions “in flight” at one time)**
 - **The higher the degree of superpipelining, the more forwarding/hazard hardware needed, the more pipeline latch overhead (i.e., the pipeline latch accounts for a larger and larger percentage of the clock cycle time), and the bigger the clock skew issues (i.e., because of faster and faster clocks)**
 - **We know there are limits to this (6–8 F04 delays), from a few slides ago**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

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!
- <https://tutorcs.com>
WeChat: cstutorcs

Instruction vs. Machine Parallelism

- **Instruction-level parallelism (ILP) of a program—a measure of the average number of instructions in a *program* that, in theory, a processor might be able to execute at the same time**
 - **Mostly determined by the number of true (data) dependencies and procedural (control) dependencies in relation to the number of other instructions**
 - **ILP is traditionally “extracting parallelism from a single instruction stream working on a single stream of data”**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Instruction vs. Machine Parallelism

- Machine parallelism of a processor—a measure of the ability of the processor to take advantage of the ILP of the program
 - Determined by the number of instructions that can be fetched and executed at the same time
 - A perfect machine with infinite machine parallelism can achieve the ILP of a program
- To achieve high performance, need both ILP and machine parallelism

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Matrix Multiplication

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

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

Assembly for y0

- $y0 = m00 * x0 + m01 * x1 + m02 * x2 + m03 * x3$
- $t0 = m00 * x0$
 $t1 = m01 * x1$
 $t2 = m02 * x2$
 $t3 = m03 * x3$
 $t4 = t0 + t1$
 $t5 = t2 + t3$
 $y0 = t4 + t5$

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Multiple Issue

■ Static multiple issue

- Compiler groups instructions to be issued together
- Packages them into “issue slots”
- Compiler detects and avoids hazards
- We'll talk about this briefly today

Assignment Project Exam Help

<https://tutorcs.com>

■ Dynamic multiple issue

WeChat: cstutorcs

- CPU examines instruction stream and chooses instructions to issue each cycle
- Compiler can help by reordering instructions
- CPU resolves hazards using advanced techniques at runtime
- This is today's main topic

Static Multiple Issue

- **Compiler groups instructions into “issue packets”**
 - **Group of instructions that can be issued on a single cycle**
 - **Determined by pipeline resources required**
- **Think of an issue packet as a very long instruction**
 - **Specifies multiple concurrent operations**
 - **⇒ Very Long Instruction Word (VLIW)**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Scheduling Static Multiple Issue

- **Compiler must remove some/all hazards**
 - **Reorder instructions into issue packets**
 - **No dependencies within a packet**
 - **Possibly some dependencies between packets**
 - **Varies between ISAs; compiler must know!**
 - **Pad with nop if necessary**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

RISC-V with Static Dual Issue

- Two-issue packets
 - One ALU/branch instruction
 - One load/store instruction
 - 64-bit aligned
 - ALU/branch, then load/store
 - Pad an unused instruction with nop

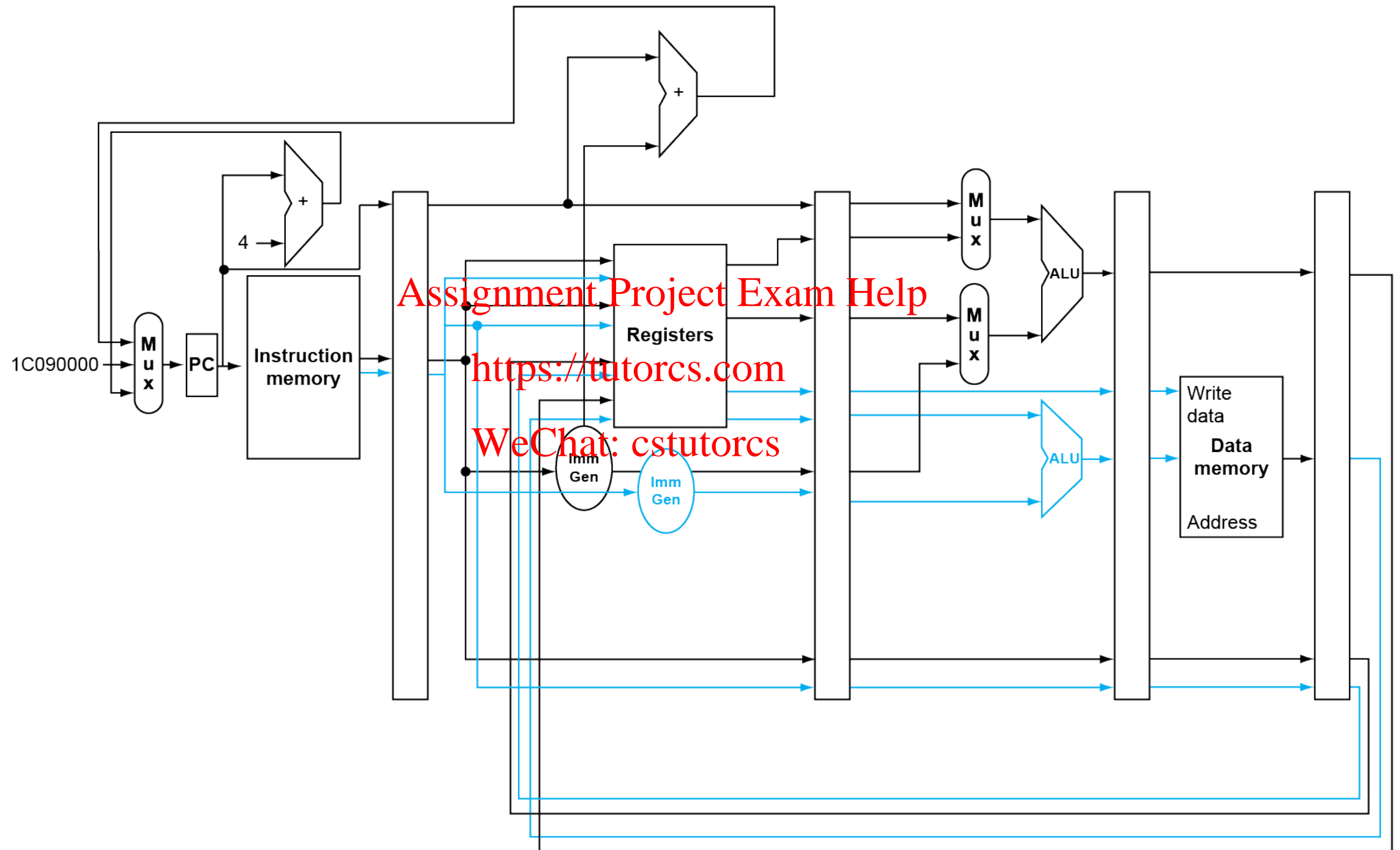
Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

RISC-V with Static Dual Issue



Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Hazards in the Dual-Issue RISC-V

- More instructions executing in parallel
- EX data hazard
 - Forwarding avoided stalls with single-issue
 - Now can't use ALU result in load/store in same packet
 - add x10, x0, x1
 - ld x2, 8(x10)
 - Split into two packets, effectively a stall
- Load-use hazard
 - Still one cycle use latency, but now two instructions
- More aggressive scheduling required

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: estutorcs

Scheduling Example

■ C code:

```
do {  
    *ptr-- += const;    // const: x21, ptr: x20  
} while (base < ptr);
```

Assignment Project Exam Help

<https://tutorcs.com>

■ Schedule this for dual-issue RISC-V

WhatsApp: @tutorcs

```
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
```

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
```

<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

■ $IPC = 5/4 = 1.25$ (c.f. peak $IPC = 2$)

Why Exploiting ILP Is Hard

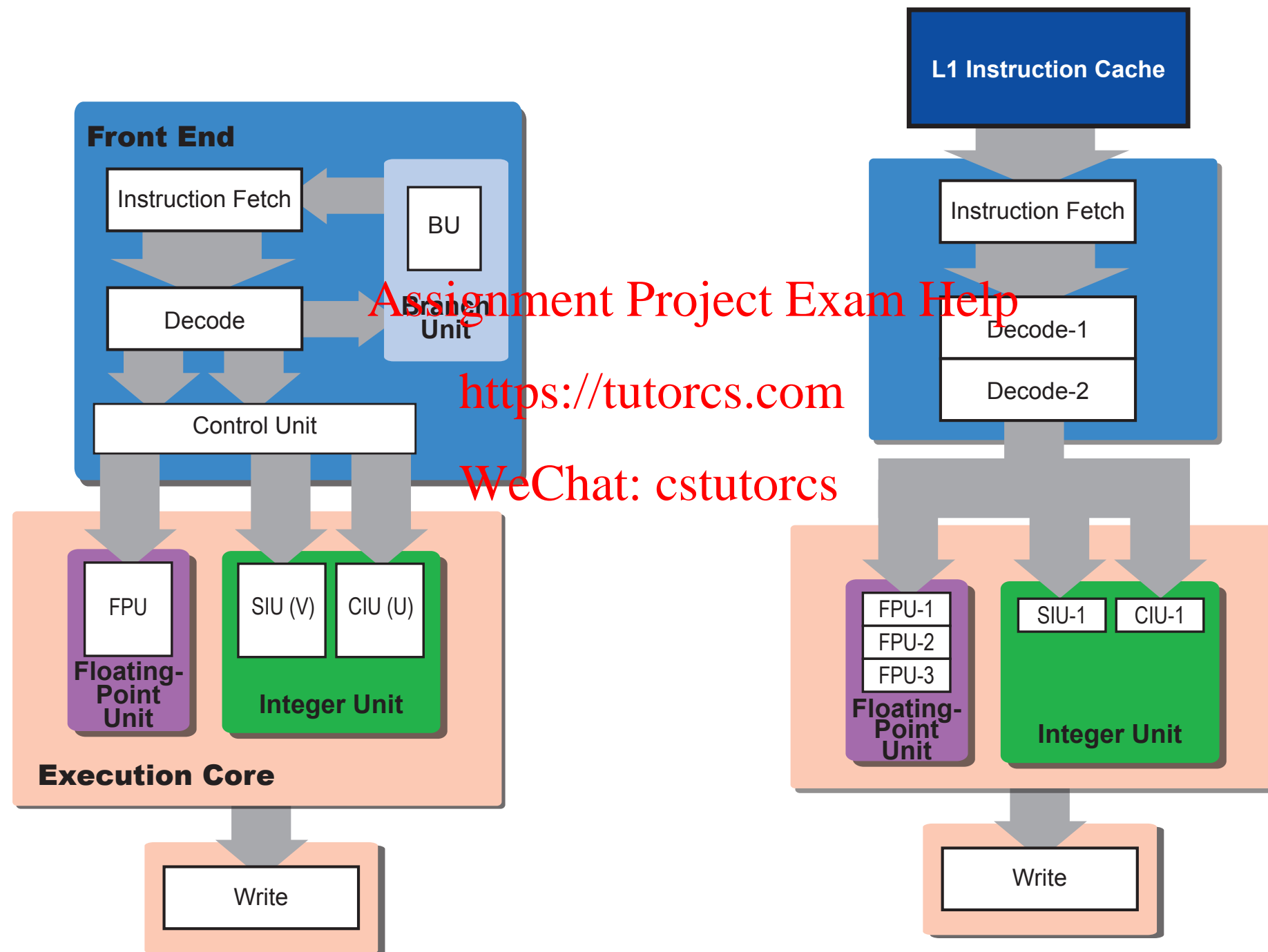
- Hazards reduce the amount of achievable machine parallelism and keep us from achieving all the ILP in the instruction stream.

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Pentium Microarchitecture



Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Pentium issue restrictions

- What restrictions would we need to place on the instructions in the U and V pipes to ensure correct execution?

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Additional restrictions

- **Some instructions are not pairable**
 - **some shift/rotate, long arith., extended, some fp, etc.**
- **Some instructions can only be issued to U**
 - **carry/borrow, prefix, shift w/ immediate, some fp**
- **Both instructions access same d-cache memory bank**
- **Multi-cycle instructions that write to memory must stall second pipe until last write**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Multiple-Issue Datapath Responsibilities

- Must handle, with a combination of hardware and software fixes, the fundamental limitations of
 - **Storage (data) dependencies**—aka data hazards
 - Most instruction streams do not have huge ILP so ...
 - ... this limits performance in a superscalar processor

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Multiple-Issue Datapath Responsibilities

- Must handle, with a combination of hardware and software fixes, the fundamental limitations of
 - **Procedural dependencies**—aka control hazards
 - Ditto, but even more severe
 - Use dynamic branch prediction (we talked about this!) to help resolve the ILP issue

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Multiple-Issue Datapath Responsibilities

- Must handle, with a combination of hardware and software fixes, the fundamental limitations of
 - **Resource conflicts**—aka structural hazards
 - A SS/VLIW processor has a much larger number of potential resource conflicts
 - Functional units may have to arbitrate for result buses and register-file write ports
 - Resource conflicts can be eliminated by duplicating the resource or by pipelining the resource

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Instruction Issue and Completion Policies

- **Instruction-issue—initiate execution**
 - **Instruction lookahead capability—fetch, decode and issue instructions beyond the current instruction**
 - **Instruction-completion—complete execution**
 - **Processor lookahead capability—complete issued instructions beyond the current instruction**
 - **Instruction-commit—write back results to the RegFile or D\$ (i.e., change the machine state)**
- In-order issue with in-order completion**
- In-order issue with out-of-order completion**
- Out-of-order issue with out-of-order completion and in-order commit**
- ~~**Out-of-order issue with out-of-order completion**~~

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

In-Order Issue with In-Order Completion

- **Simplest policy is to issue instructions in exact program order and to complete them in the same order they were fetched (i.e., in program order)**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

In-Order Issue with In-Order Completion (Ex.)

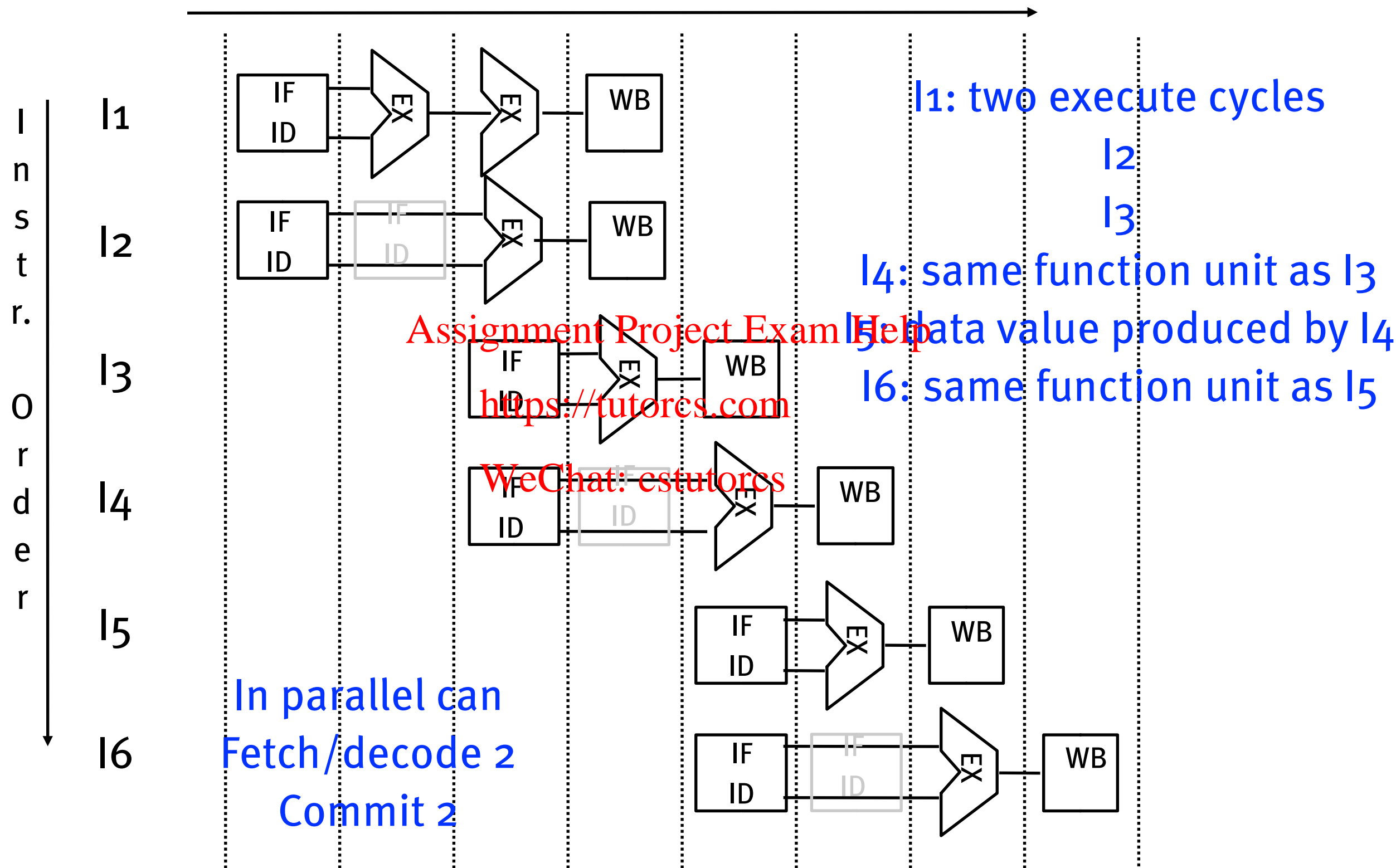
- Assume a pipelined processor that can fetch and decode two instructions per cycle, that has three functional units (a single cycle adder, a single cycle shifter, and a two cycle multiplier), and that can complete (and commit/write back) two results per cycle
- Instruction sequence:
 - I1 – needs two execute cycles (a multiply)
 - I2
 - I3
 - I4 – needs the same function unit as I3
 - I5 – needs data value produced by I4
 - I6 – needs the same function unit as I5

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

In-Order Issue, In-Order Completion Example



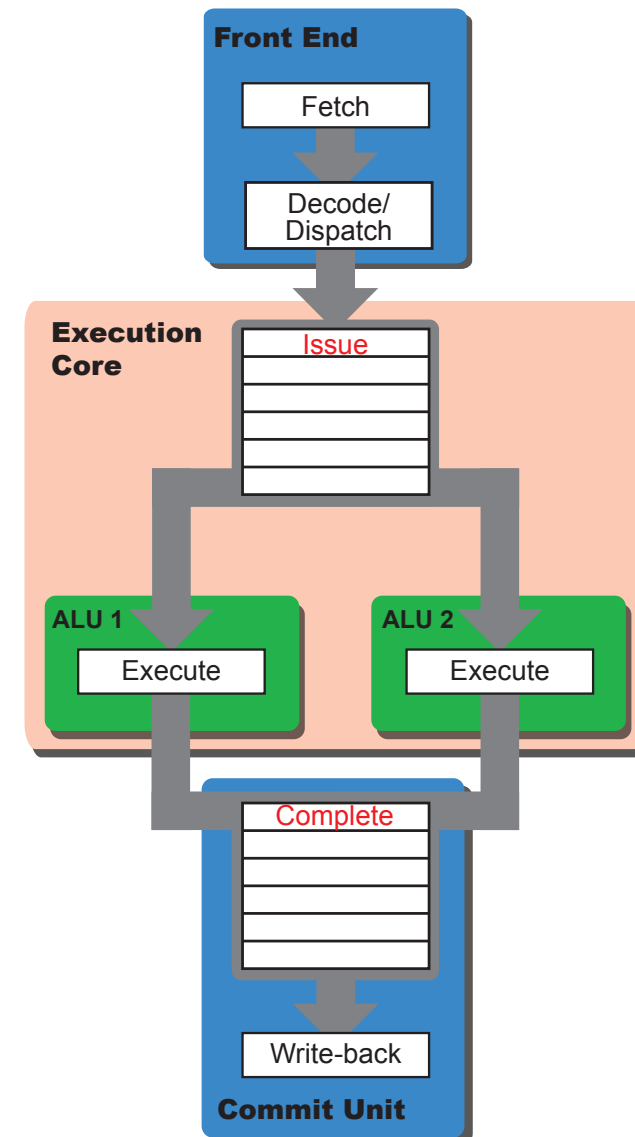
Pentium Retrospective

- Limited in performance by “front end”
 - Has to support variable-length instrs and segments
- Supporting all x86 features tough!
 - 30% of transistors are for legacy support
 - Up to 40% in Pentium Pro!
 - Down to 10% in P4
 - Microcode ROM is huge

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs



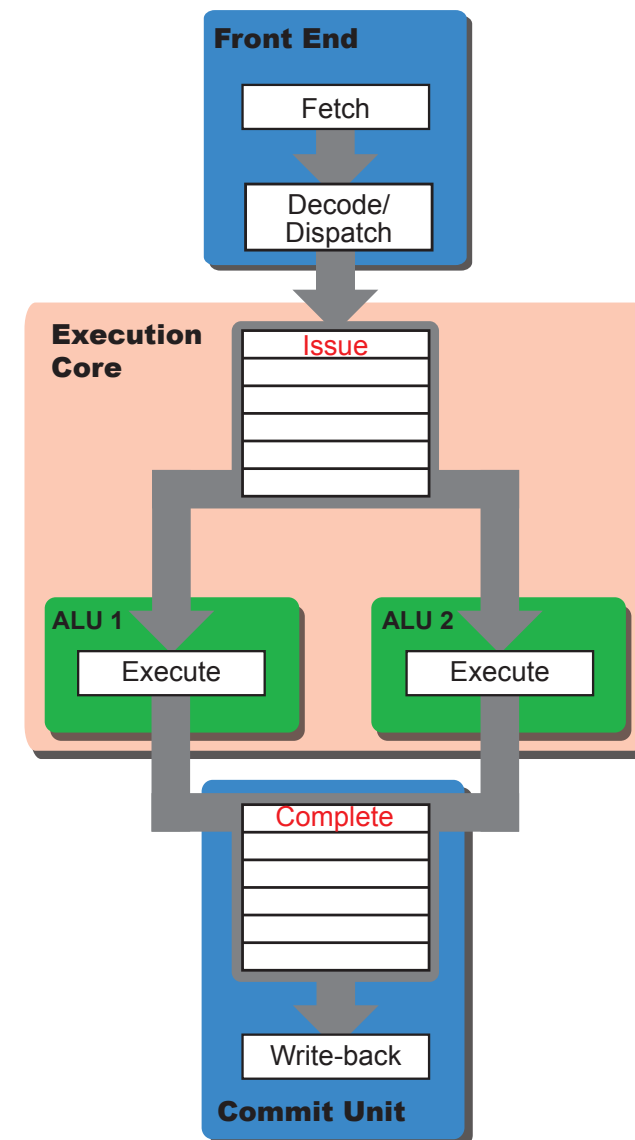
Pentium Retrospective

- Pentium is in-order issue, in-order complete
- “Static scheduling” by the dispatch logic:
 - Fetch/dispatch/execute/retire: all in order
- Drawbacks:
 - Adapts poorly to dynamic code stream
 - Adapts poorly to future hardware
 - What if we had 3 pipes not 2?

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cs_tutorcs



In-Order Issue with Out-of-Order Completion

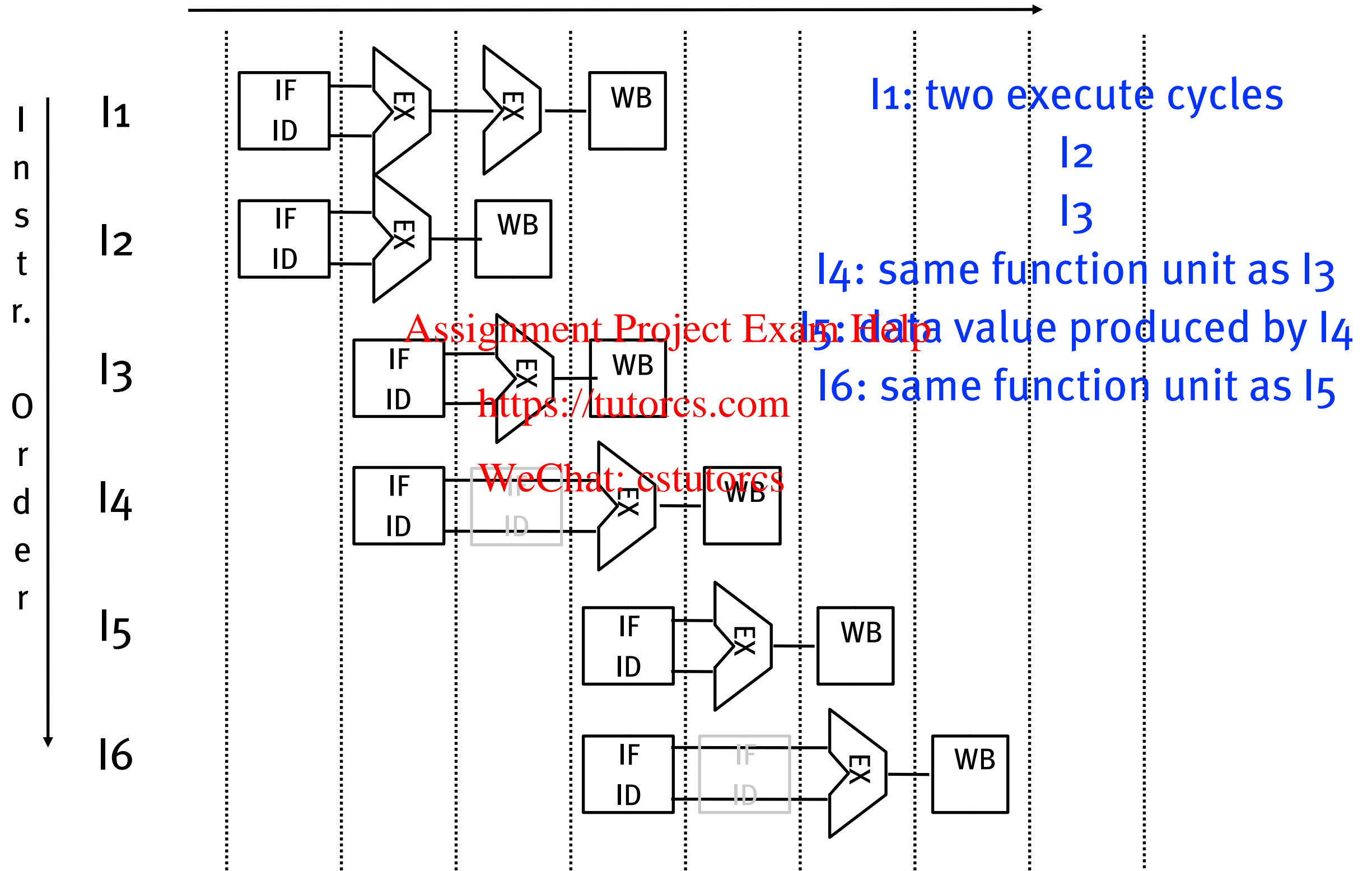
- With out-of-order completion, a later instruction may complete **before** a previous instruction
 - Out-of-order completion is used in single-issue pipelined processors to improve the performance of long-latency operations such as divide
- When using out-of-order completion instruction issue is **stalled** when there is a resource conflict (e.g., for a functional unit) or when the instructions ready to issue need a result that has not yet been computed

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

IOI-00C Example



Handling Output Dependencies

- There is one more situation that stalls instruction issuing with IOI-00C, assume
 - I1 – writes to R3
 - I2 – writes to R3
 - I5 – reads R3
- If the I1 write occurs after the I2 write, then I5 reads an incorrect value for R3
- I2 has an output dependency on I1—write before write
- The issuing of I2 would have to be stalled if its result might later be overwritten by an previous instruction (i.e., I1) that takes longer to complete—the stall happens before instruction issue
- While IOI-00C yields higher performance, it requires more dependency checking hardware (both write-after-read and write-after-write)

Assignment Project Exam Help

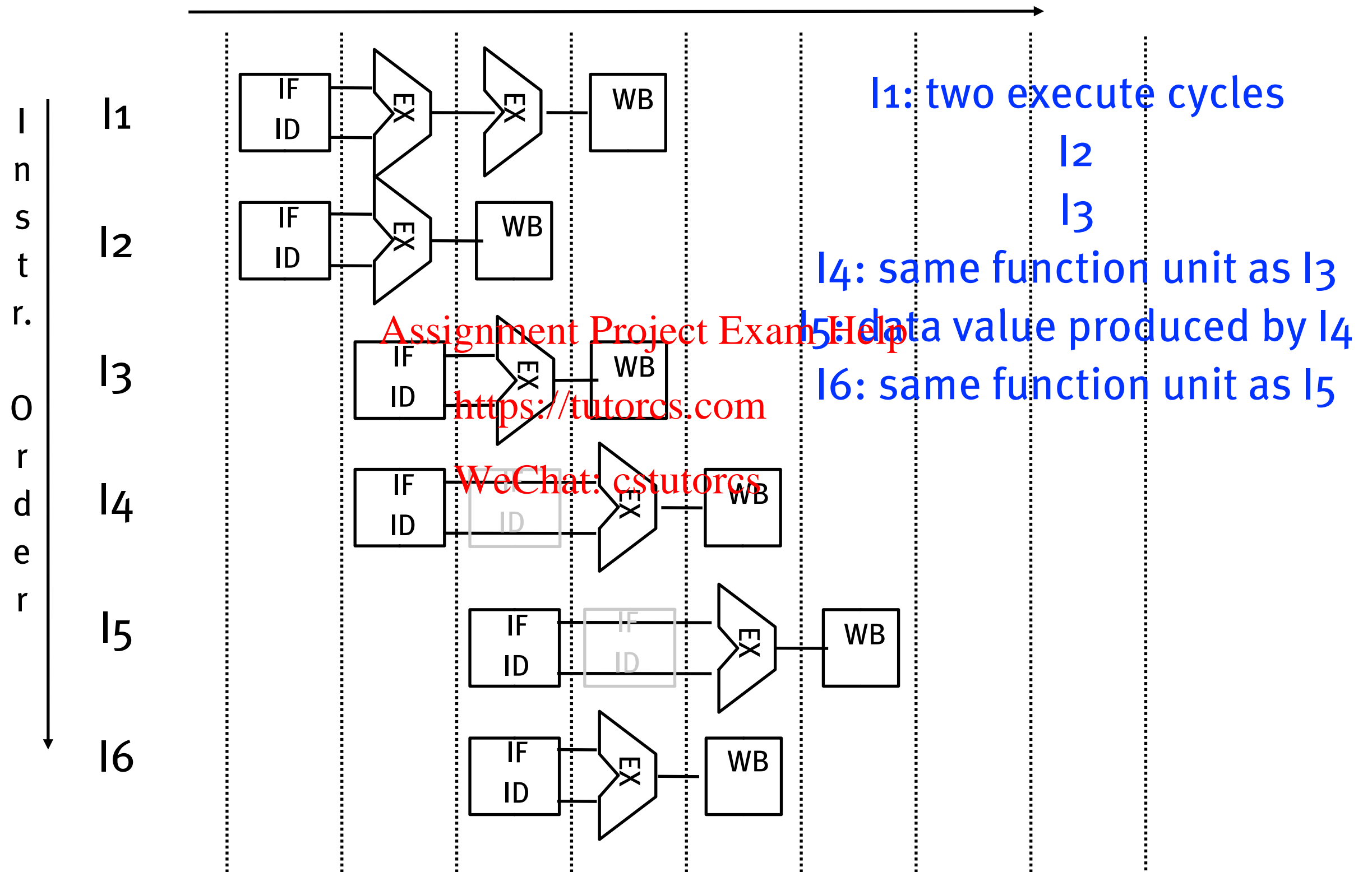
<https://tutorcs.com>

WeChat: cstutorcs

Out-of-Order Issue with Out-of-Order Completion

- With in-order issue the processor stops decoding instructions whenever a decoded instruction has a resource conflict or a data dependency on an issued, but uncompleted, instruction
 - The processor is not able to look beyond the conflicted instruction even though more downstream instructions might have no conflicts and thus be issueable
- Fetch and decode instructions beyond the conflicted one ("instruction window": Tetris), store them in an instruction buffer (as long as there's room), and flag those instructions in the buffer that don't have resource conflicts or data dependencies
- Flagged instructions are then issued from the buffer without regard to their program order

001-00C Example



Dependency Examples

■ $R3 := R3 * R5$

$R4 := R3 + 1$

$R3 := R5 + 1$

True data dependency (RAW)

Output dependency (WAW)

Antidependency (WAR)

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Antidependencies (WAR)

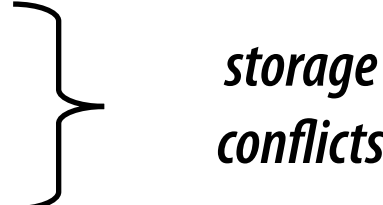
- With OOI also have to deal with data antidependencies – when a later instruction (that completes earlier) produces a data value that destroys a data value used as a source in an earlier instruction (that issues later)
- The constraint is similar to that of true data dependencies, except reversed
 - Instead of the later instruction using a value (not yet) produced by an earlier instruction (read before write), the later instruction produces a value that destroys a value that the earlier instruction (has not yet) used (write before read)

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Dependencies Review

- Each of the three data dependencies ...
 - True data dependencies (read before write)
 - Antidependencies (write before read)
 - Output dependencies (write before write)

storage conflicts
- ... manifests itself through the use of registers (or other storage locations)

<https://tutorcs.com>
- True dependencies represent the flow of data and information through a program

WeChat: estutorcs
- Anti- and output dependencies arise because the limited number of registers mean that programmers reuse registers for different computations
- When instructions are issued out-of-order, the correspondence between registers and values breaks down and the values conflict for registers

Conflict example

- (1) $R3 := R3 * R5$ $// i = j * k$
 - (2) $R4 := R3 + 1$ $// m = i + 1$
 - (3) $R3 := R5 + 1$ $// n = k + 1$
- (3) must ensure that (1) and (2) read R3 before (3) writes it
 - But the value in (3)'s R3 has nothing to do with the value in (1) and (2)'s R3! Shouldn't we be able to issue that instruction?

WeChat: cstutorcs

Storage Conflicts and Register Renaming

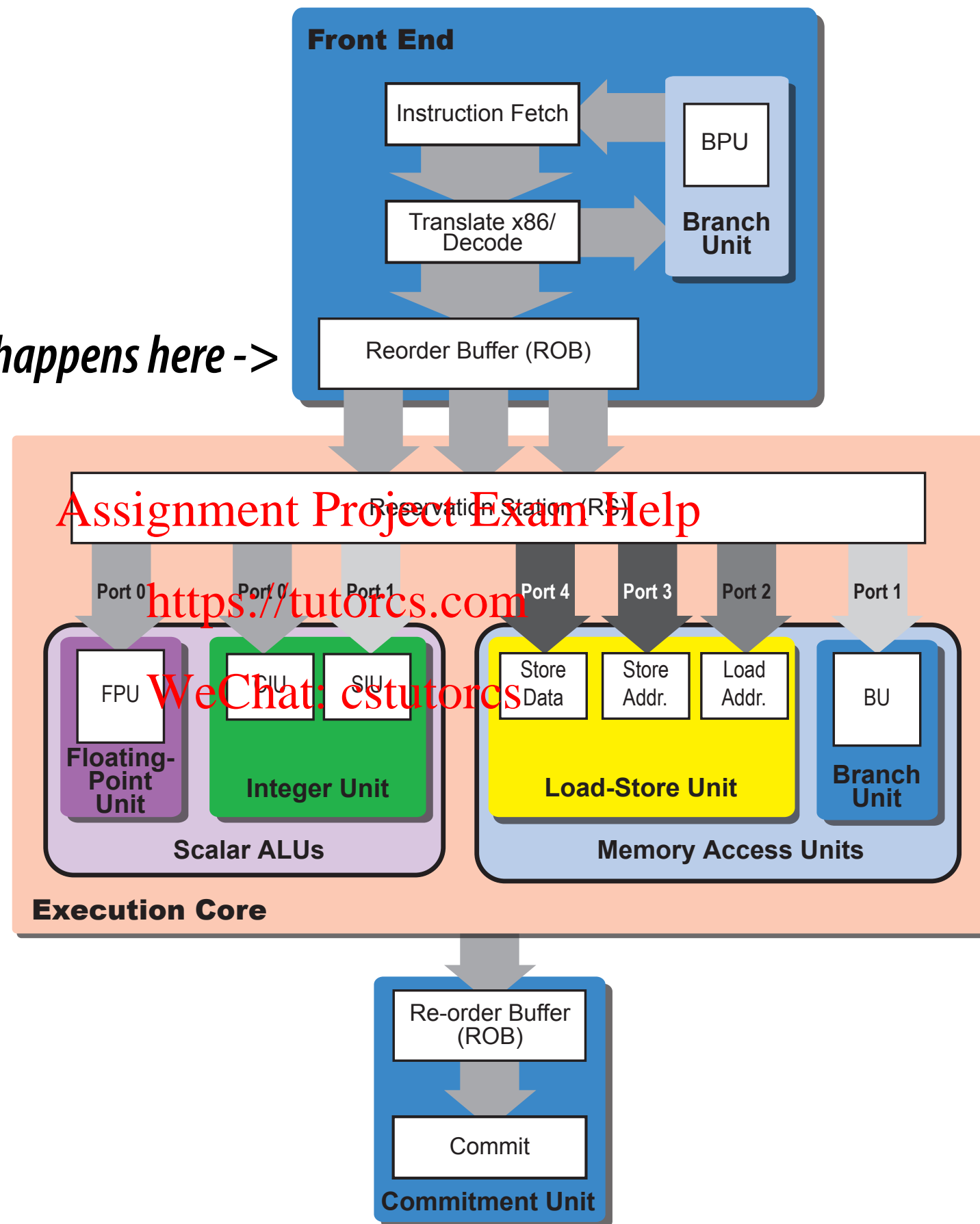
- Storage conflicts can be reduced (or eliminated) by increasing or duplicating the troublesome resource
 - Provide additional registers that are used to reestablish the correspondence between registers and values
 - Allocated dynamically by the hardware in SS processors
- Register renaming — the processor renames the original register identifier in the instruction to a new register (one not in the visible register set)

- $R3 := R3 * R5$ $R3b := R3a * R5a$
 $R4 := R3 + 1$ $R4a := R3b + 1$
 $R3 := R5 + 1$ $R3c := R5a + 1$

- The hardware that does renaming assigns a “replacement” register from a pool of free registers and releases it back to the pool when its value is superseded and there are no outstanding references to it

Pentium Pro

register renaming happens here ->



Pentium Pro

1. Fetch In order

2. Decode/dispatch In order

3. Issue Reorder

4. Execute Out of order

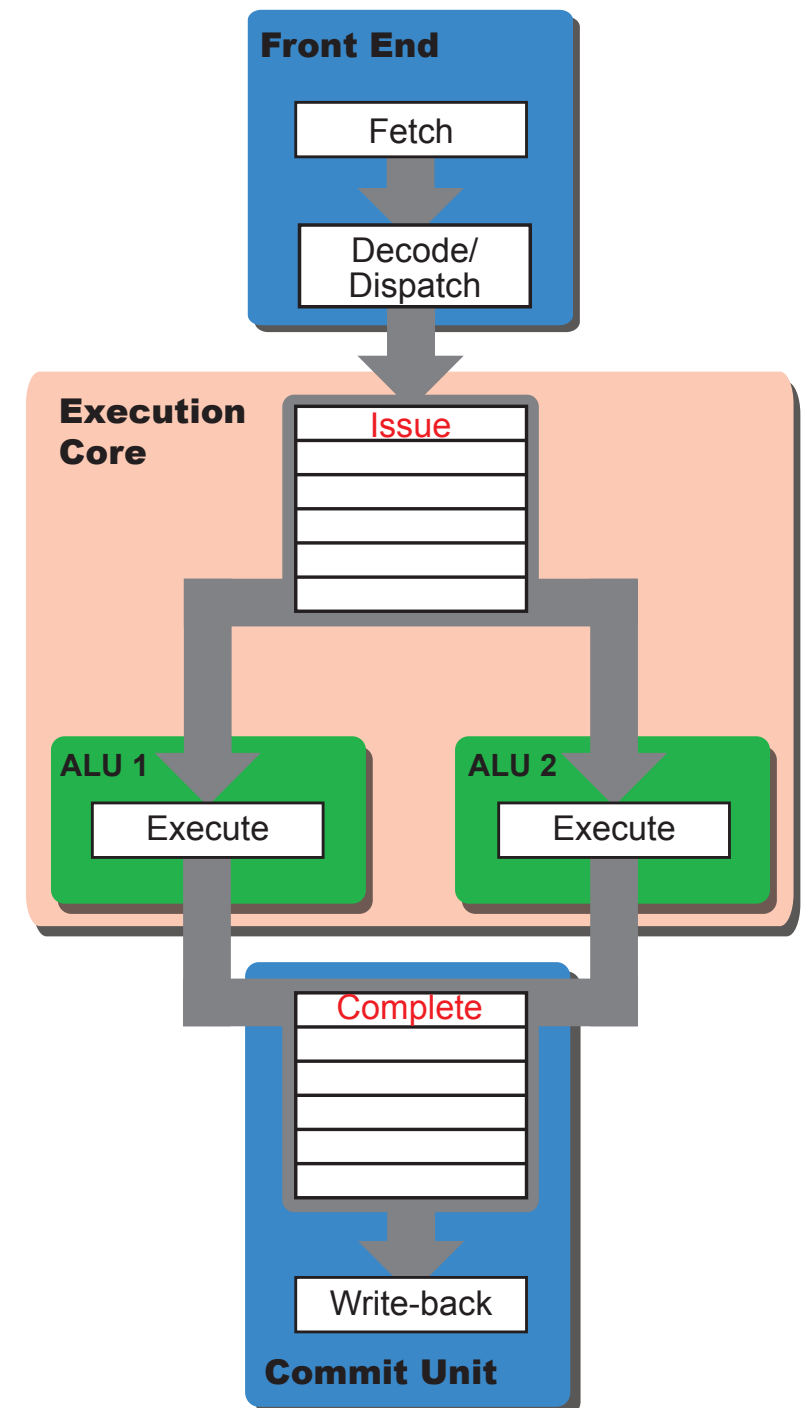
5. Complete Reorder

6. Writeback (commit) In order

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cs_tutorcs



P6 Pipeline

- **Instruction fetch, BTB access (3.5 stages)**
 - 2 cycles for instruction fetch
- **Decode, x86->uops (2.5 stages)**
- **Register rename (1 stage)**
- **Write to reservation station (1 stage)**
- **Read from reservation station (1 stage)**
- **Execute (1+ stages)**
- **Commit (2 stages)**

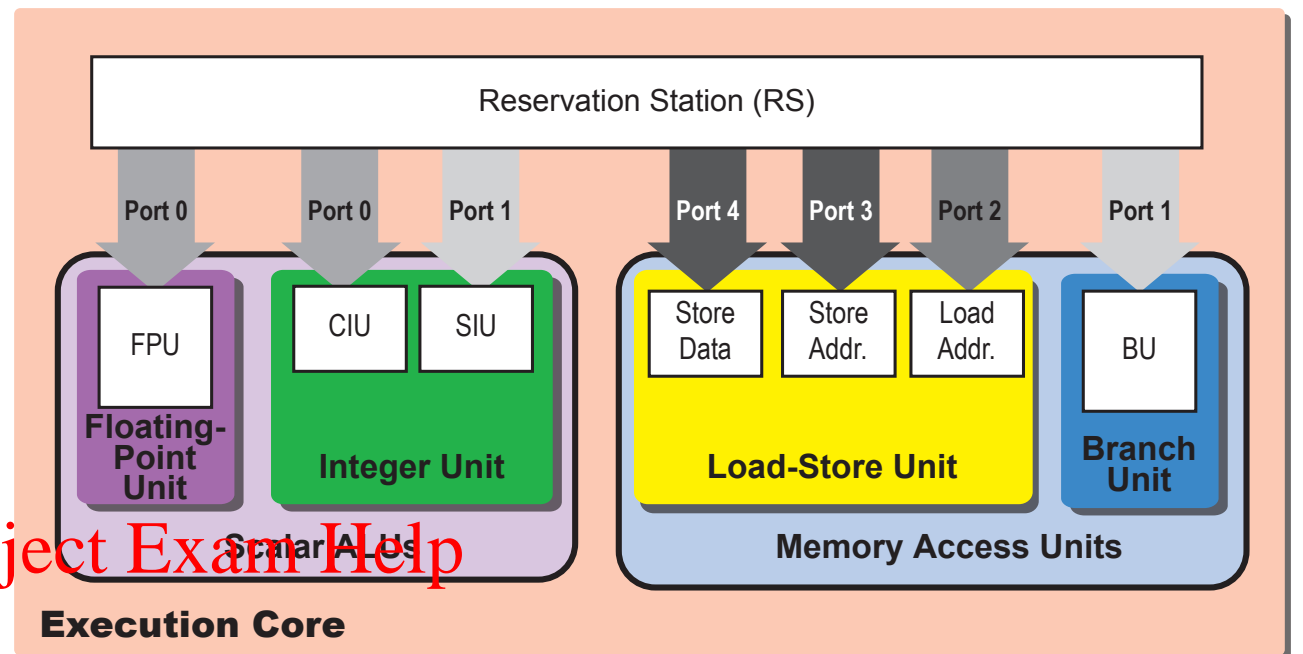
Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Pentium Pro backends

■ Pentium Pro

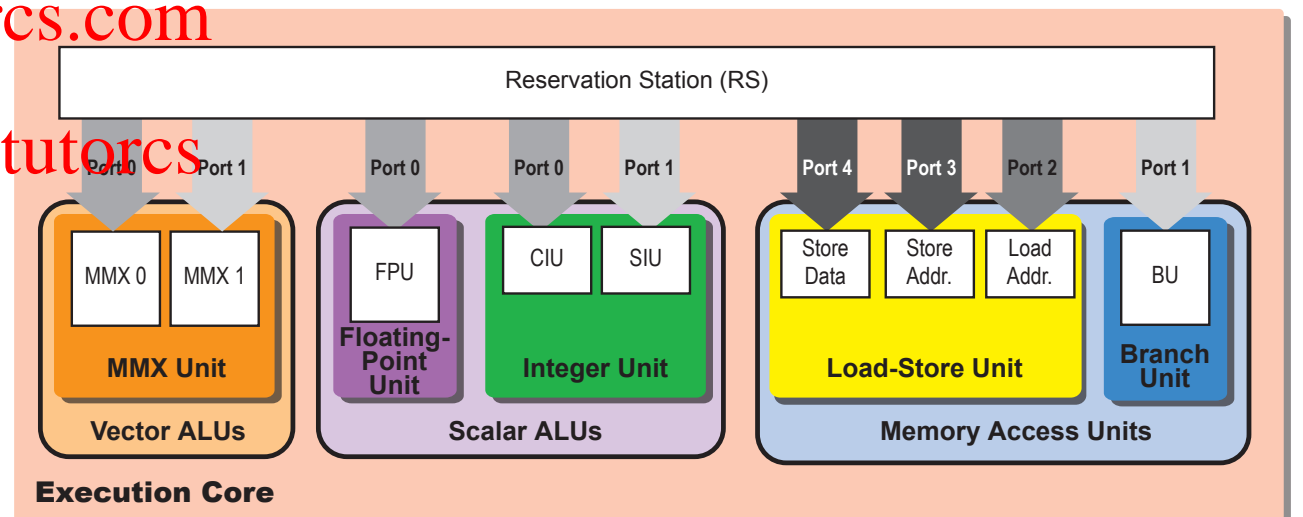


Assignment Project Exam Help

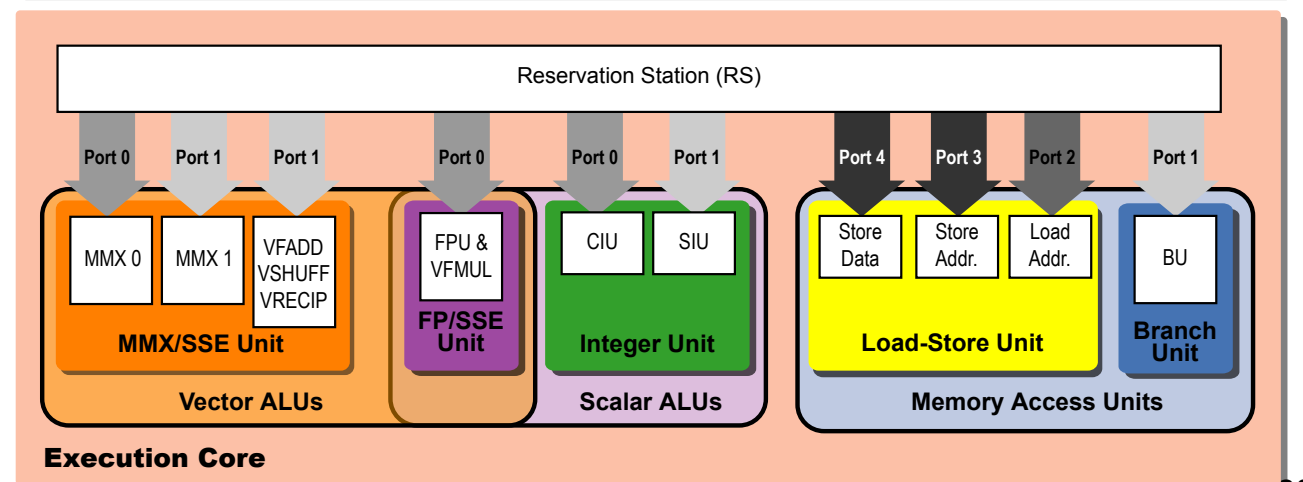
<https://tutorcs.com>

WeChat: cstutorcs

■ Pentium 2

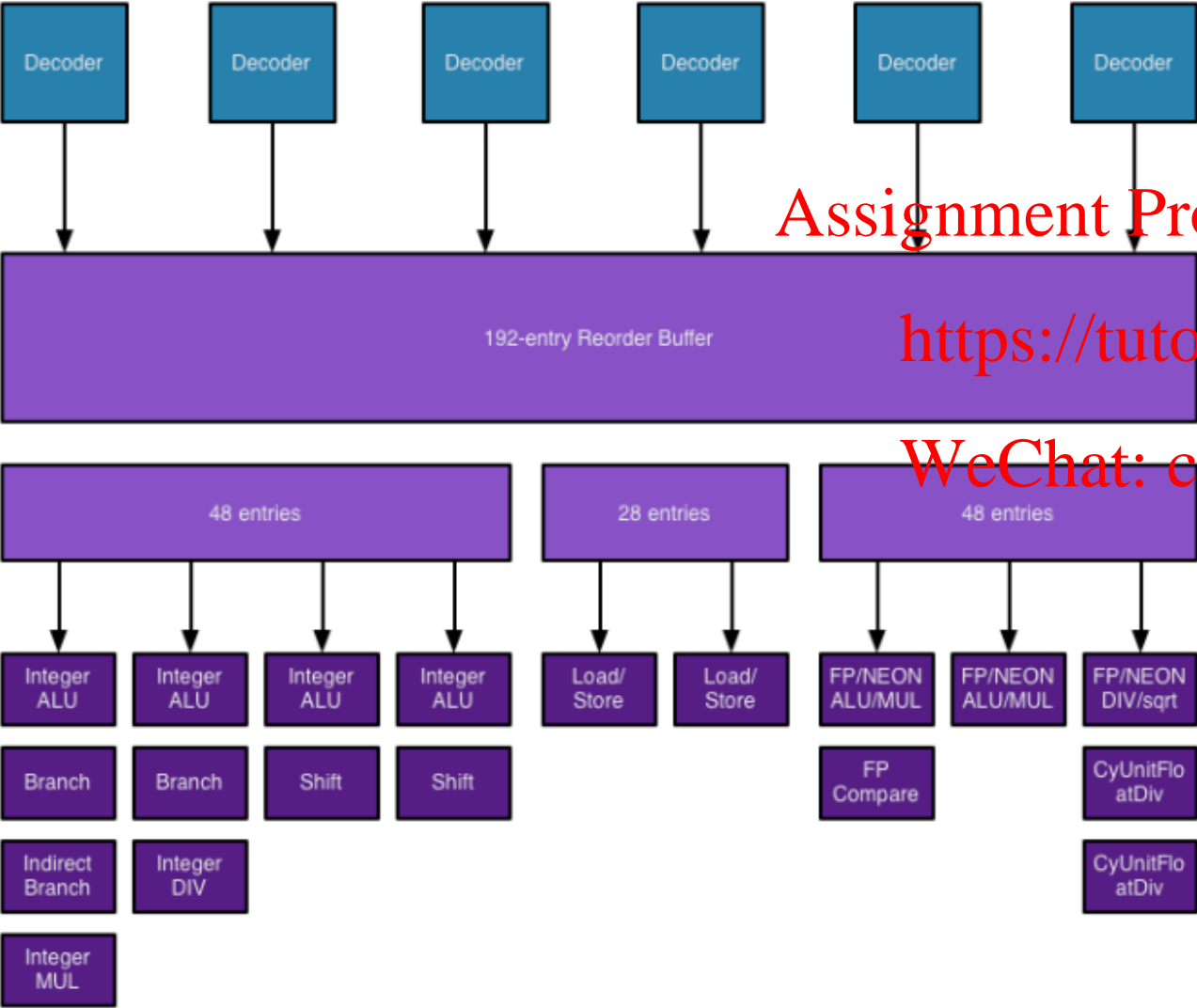


■ Pentium 3



Apple “Cyclone” A7 (2014)

Apple Cyclone



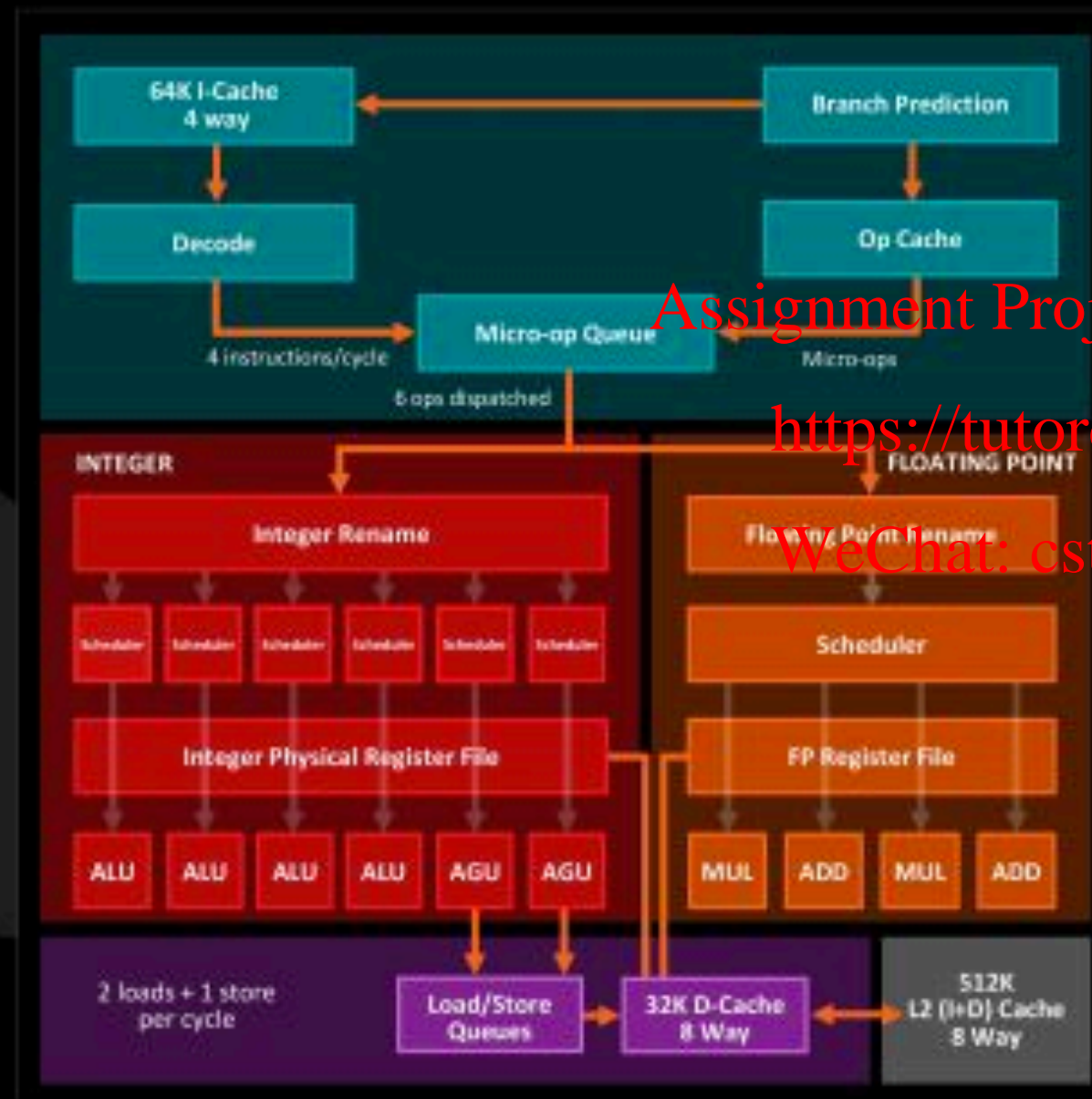
Apple Custom CPU Core Comparison		
	Apple A6	Apple A7
CPU Codename	Swift	Cyclone
ARM ISA	ARMv7-A (32-bit)	ARMv8-A (32/64-bit)
Issue Width	3 micro-ops	6 micro-ops
Reorder Buffer Size	45 micro-ops	192 micro-ops
Branch Mispredict Penalty	14 cycles	16 cycles (14 - 19)
Integer ALUs	2	4
Load/Store Units	1	2
Load Latency	3 cycles	4 cycles
Branch Units	1	2
Indirect Branch Units	0	1
FP/NEON ALUs	?	3
L1 Cache	32KB I\$ + 32KB D\$	64KB I\$ + 64KB D\$
L2 Cache	1MB	1MB
L3 Cache	-	4MB

AMD Zen (2016)

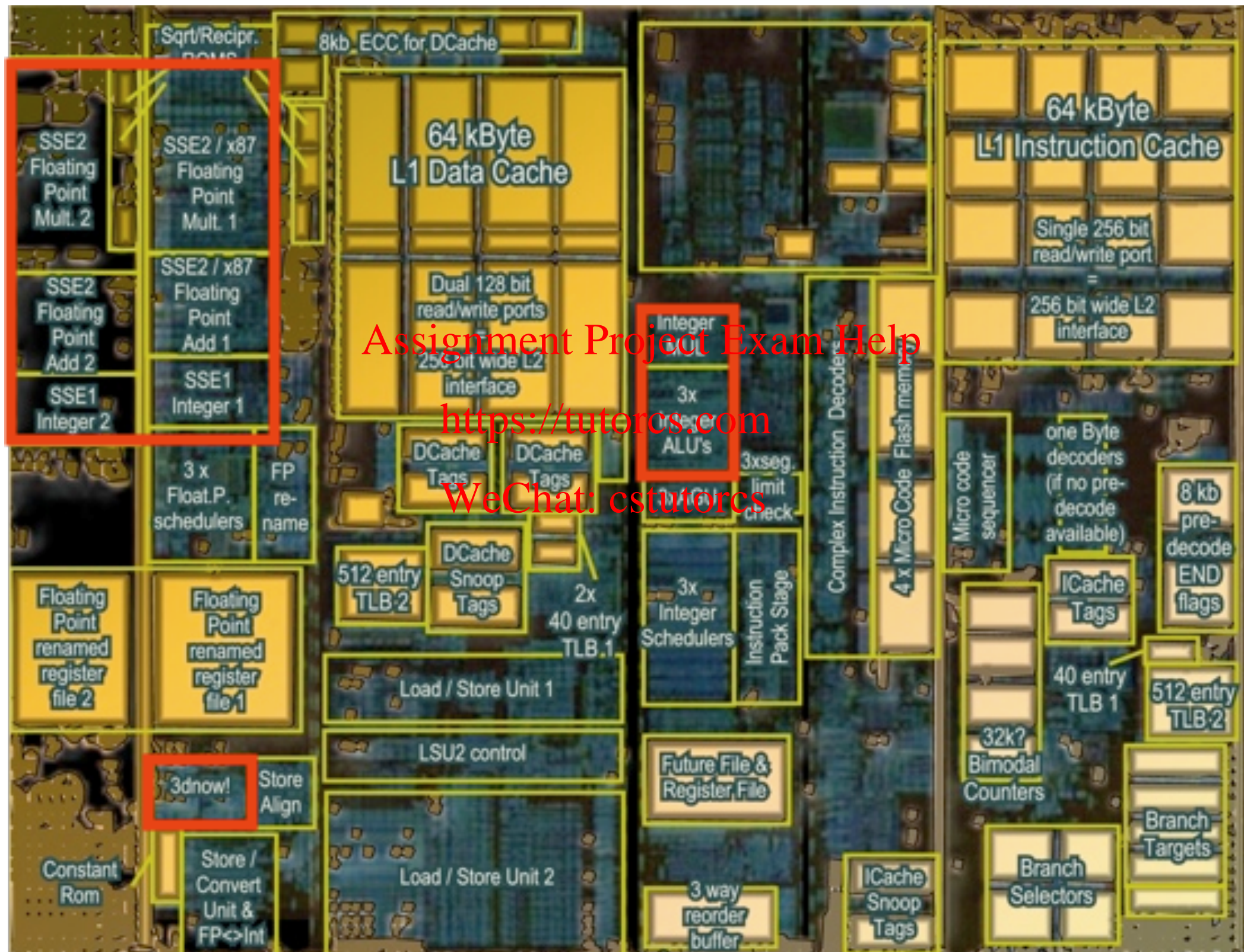


ZEN MICROARCHITECTURE

- Fetch Four x86 instructions
- Op Cache instructions
- 4 Integer units
 - Large rename space – 168 Registers
 - 192 instructions in flight/8 wide retire
- 2 Load/Store units
 - 72 Out-of-Order Loads supported
- 2 Floating Point units x 128 FMACs
 - built as 4 pipes, 2 Fadd, 2 Fmul
- I-Cache 64K, 4-way
- D-Cache 32K, 8-way
- L2 Cache 512K, 8-way
- Large shared L3 cache
- 2 threads per core



AMD K7 "Deerhound"



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