MONASH University
Information Technology

FIT3159
Computer Architecture

CPU Organisation – Superscalar and Multi-Processing

Dr Carlo Kopp, SMIEEE, AFAIAA, PEng
Carlo.Kopp@monash.edu
Clayton School of Information Technology,
Monash University
Australia

Why Study Superscalar Processing?

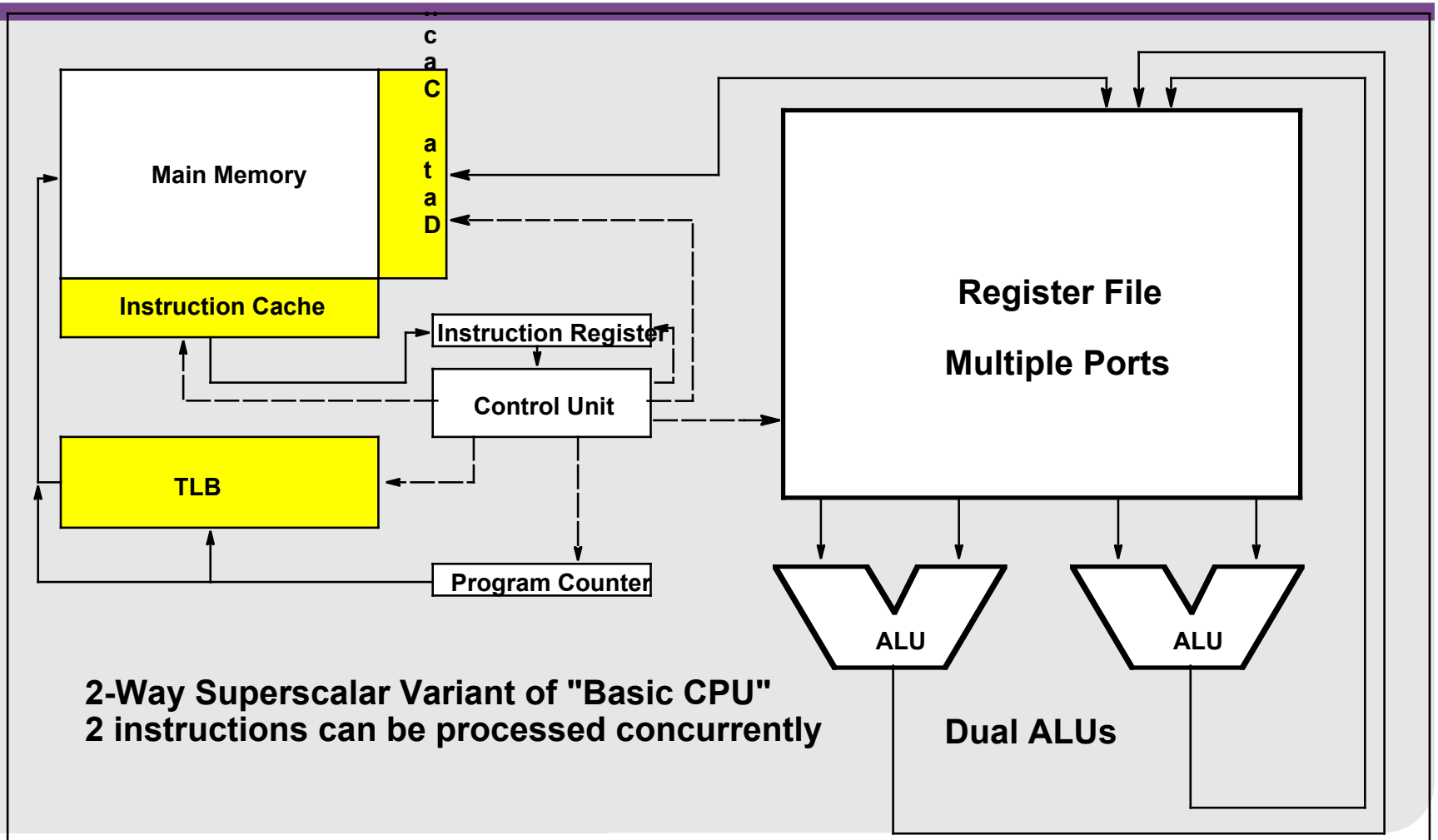
- ***CPU organisation and design strongly impacts performance and functionality of all modern computers.***
- **Foundation knowledge: Superscalar design critical to system performance and a major area in contemporary machine design – most current CPUs are superscalar designs.**
- **Practical skills: Ability to optimise application code design to best take advantage of superscalar architecture and thus maximise application performance.**
- **Practical skills: Ability to recognise and differentiate machine performance by understanding superscalar design limitations.**
- **Practical skills: Ability to anticipate likely application performance problems by relating application design to superscalar organisation in target platforms for the application.**

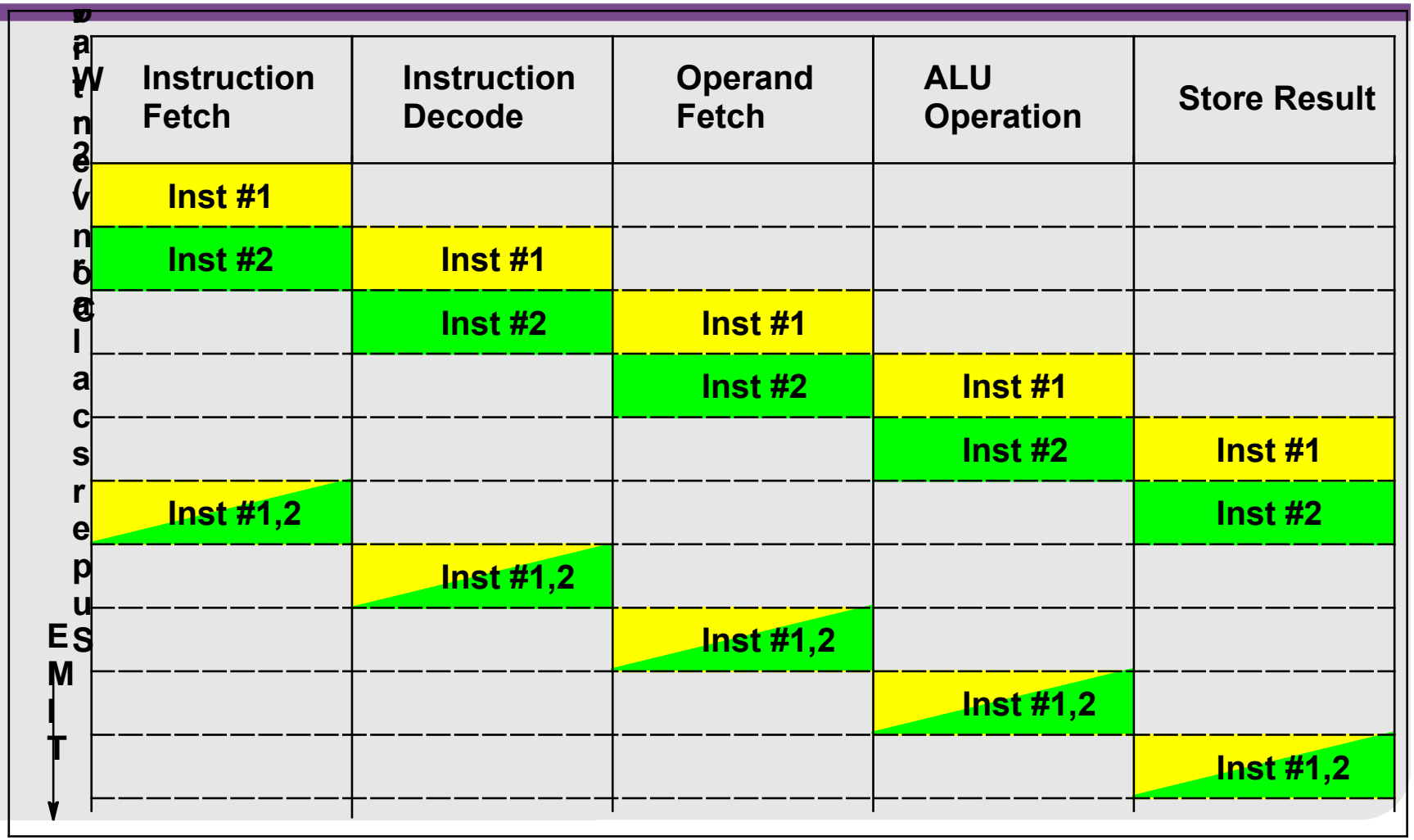
What is Instruction Level Parallelism?

- Where multiple instructions do not depend upon the results of each other, these instructions can be executed in parallel, i.e. concurrently.
- The property of not being mutually dependent upon each other's results is termed ILP.
- The level of ILP in any piece of code depends on the algorithm being executed, and how the compiler organises the instructions.
- *How can we exploit ILP?*

Superscalar Processing

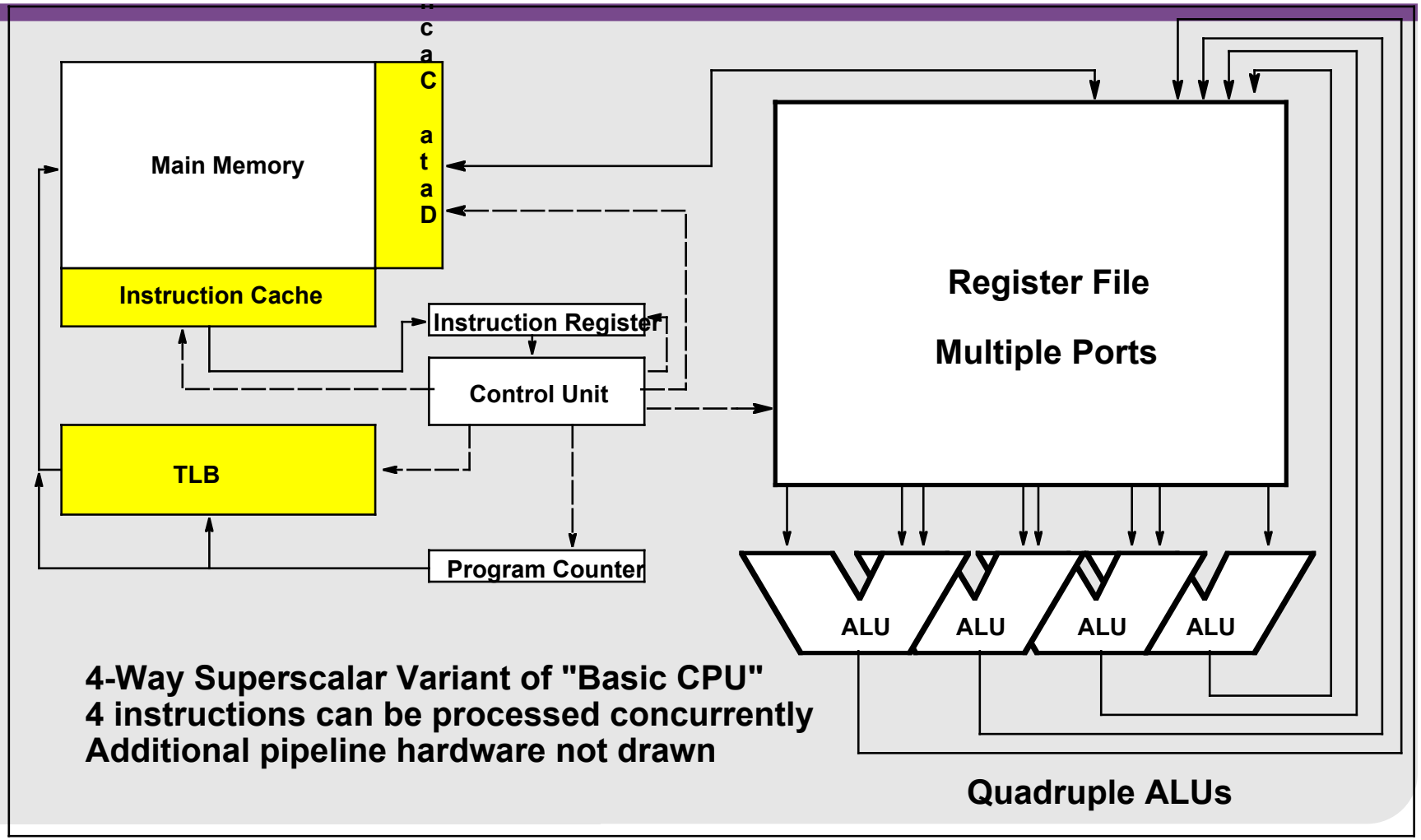
- One technique for improving CPU performance by exploiting ILP is “superscalar processing”.
- Superscalar techniques first appeared in mainframes and supercomputers during the 1960s, now they are used in most modern CPUs.
- A superscalar CPU employs more than one pipeline/ ALU, so it can execute more than one instruction concurrently, should the opportunity arise to do so.

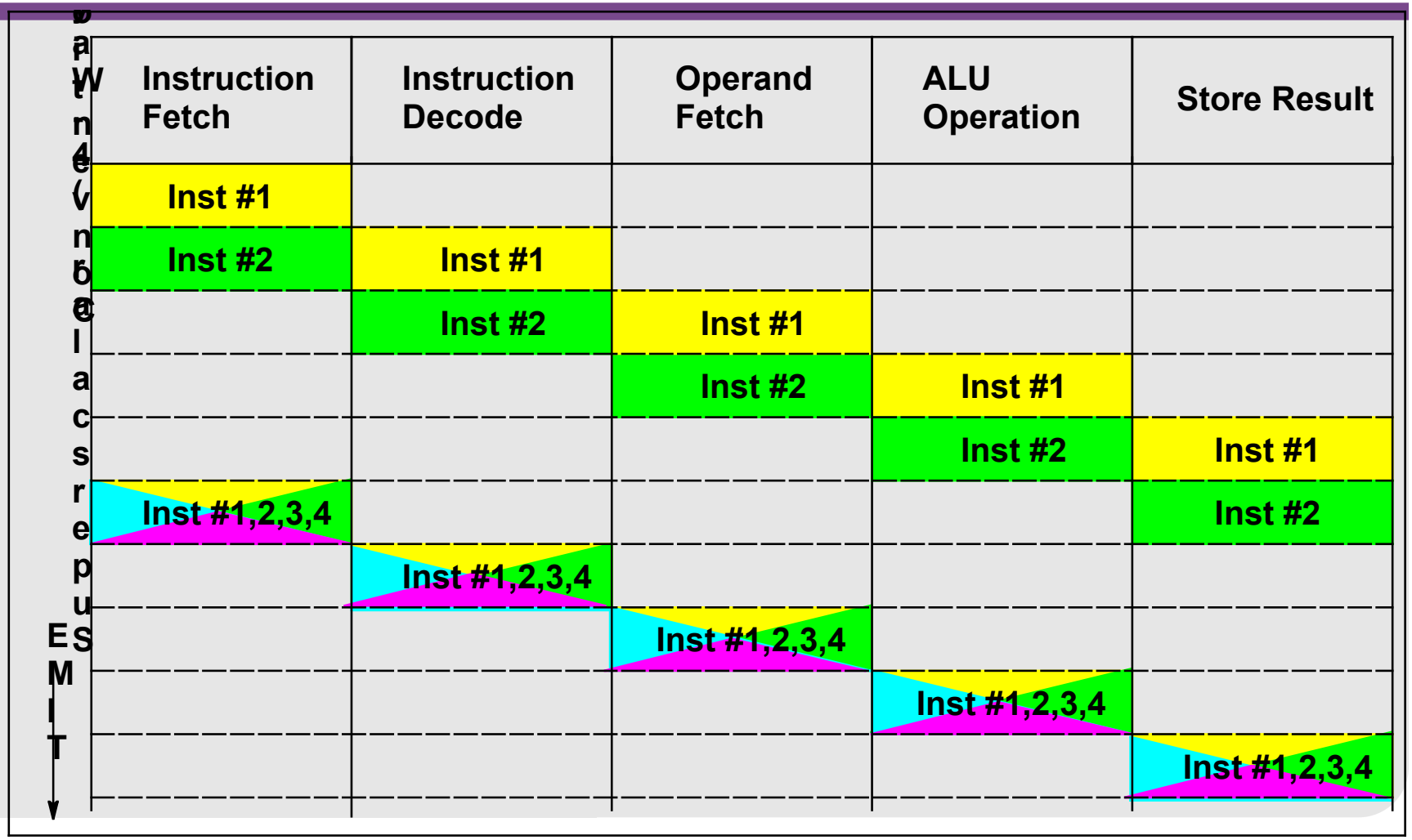

Main Memory
Data Cache
Instruction Cache
Instruction Register
Control Unit
TLB
Program Counter
Register File
Multiple Ports
ALU
ALU
Dual ALUs
2-Way Superscalar Variant of "Basic CPU"
2 instructions can be processed concurrently

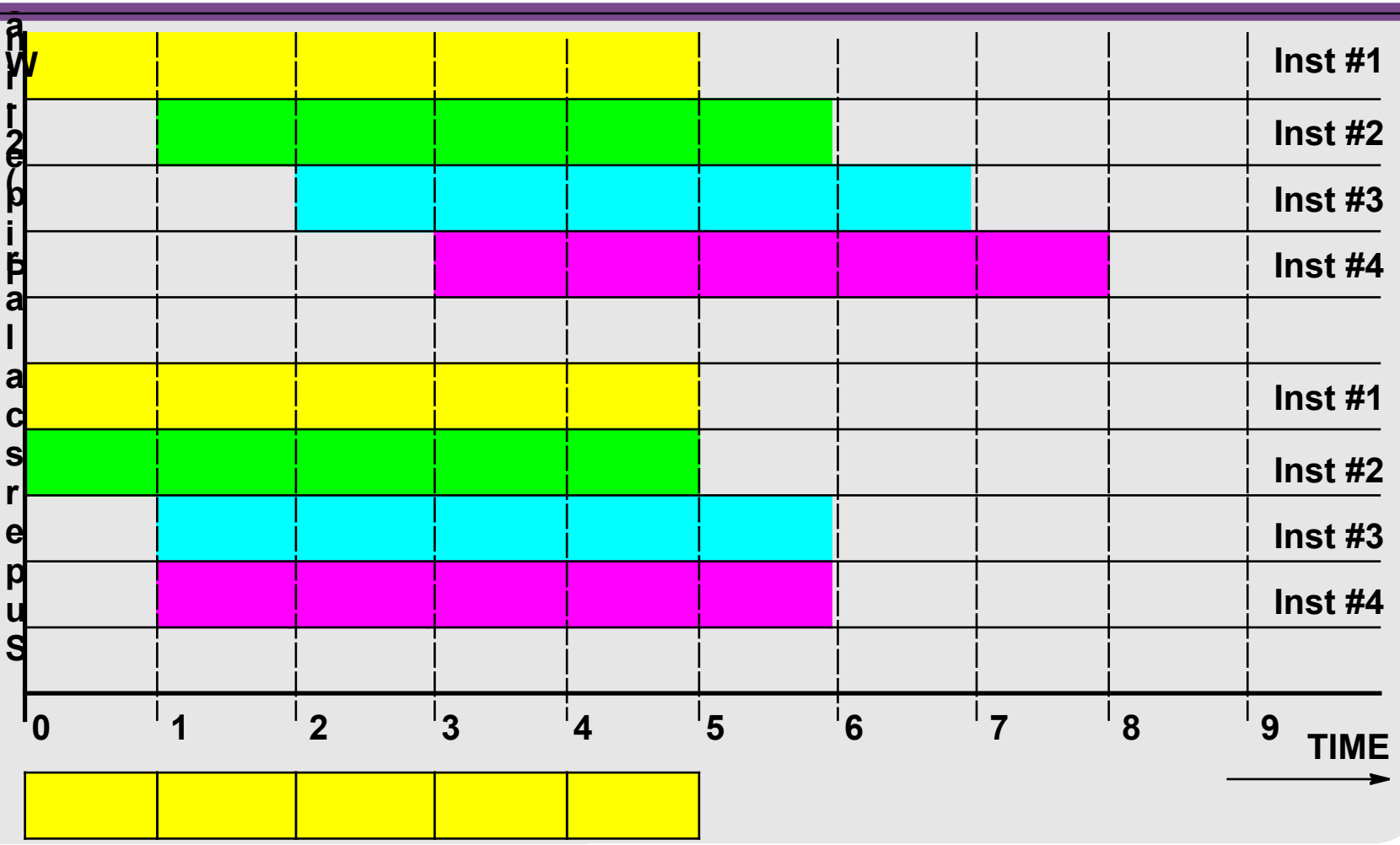

Instruction Fetch
Instruction Decode
Operand Fetch
ALU Operation
Store Result
Inst #1
Inst #2
Inst #1
Inst #2
Inst #1
Inst #2
Inst #1
Inst #2
Inst #1
Inst #1,2
Inst #2
Inst #1,2
Inst #1,2
Inst #1,2
Inst #1,2
Wait (VMEM access)
TIME

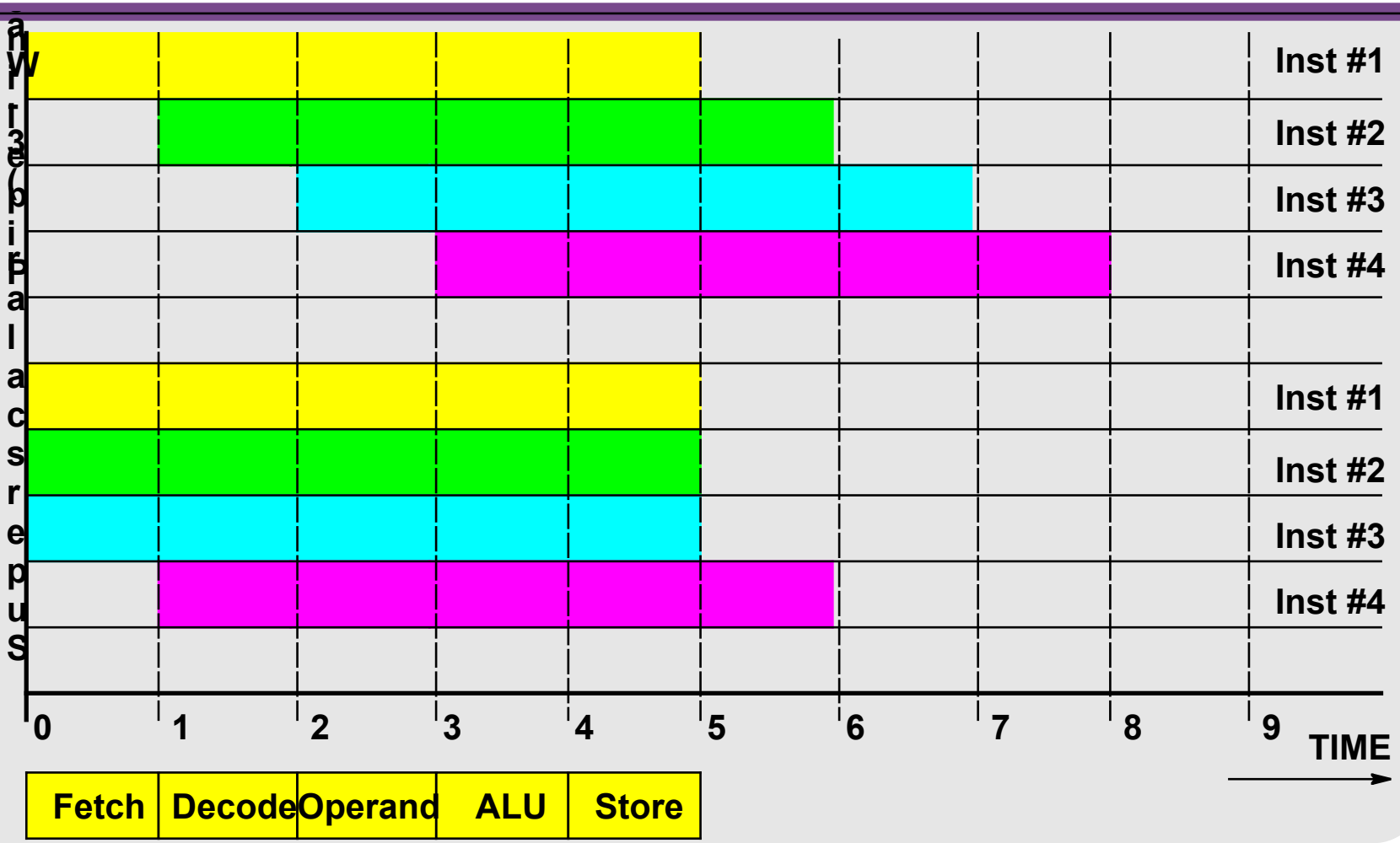
Superscalar Processing

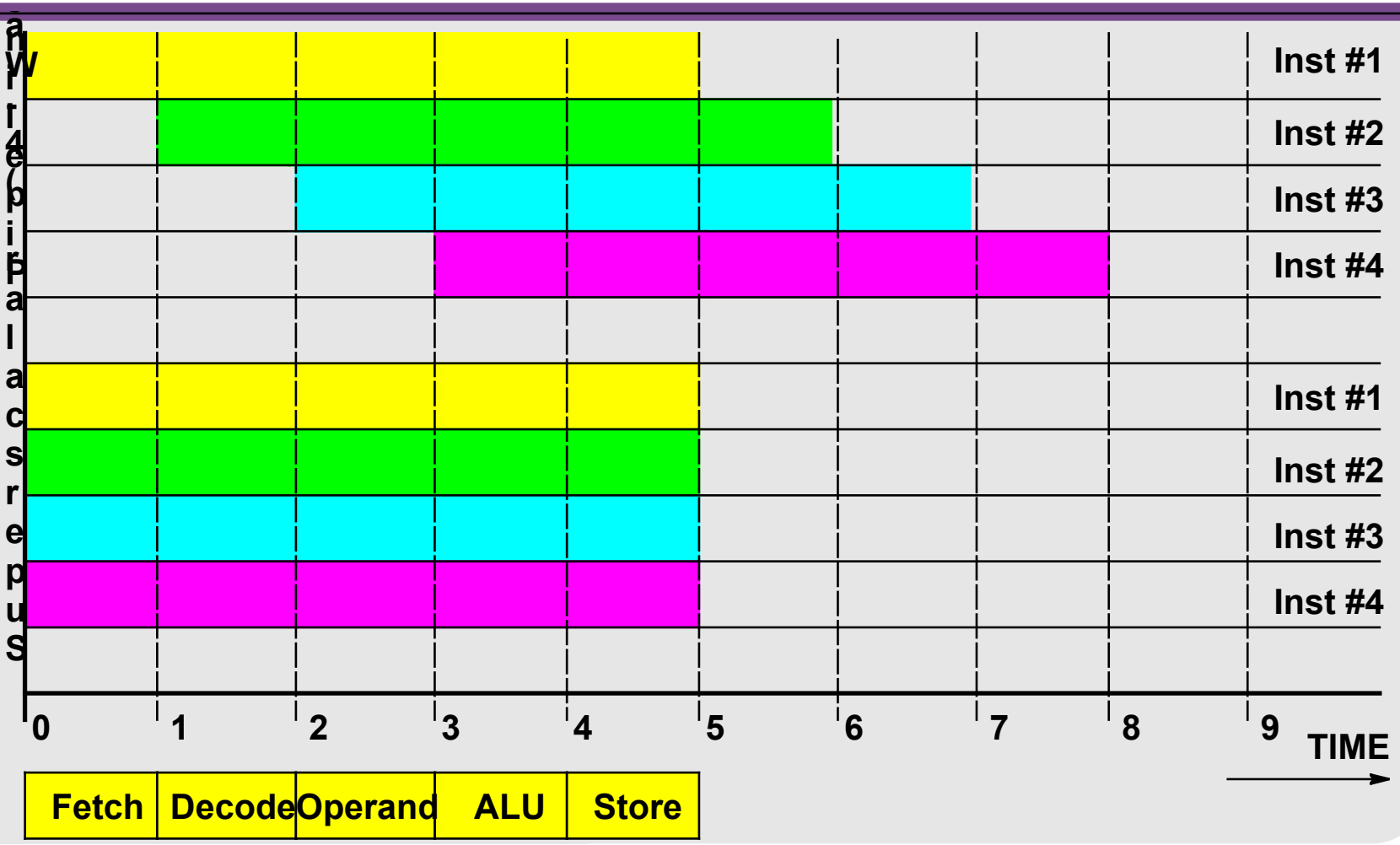
- An "N-Way Superscalar" CPU has N pipelines/ALUs and can execute N instructions in parallel, should ILP permit so.
- Therefore an "N-Way Superscalar" CPU is potentially N-times faster than its equivalent with a single ALU.
- *The full performance potential of a superscalar CPU is only realised when the code it is executing has a high level of ILP in it. If the code has poor ILP, performance may not differ from a conventional CPU.*


Main Memory
Instruction Cache
Instruction Register
Control Unit
TLB
Program Counter
Register File
Multiple Ports
ALU
ALU
ALU
ALU
4-Way Superscalar Variant of "Basic CPU"
4 instructions can be processed concurrently
Additional pipeline hardware not drawn
Quadruple ALUs

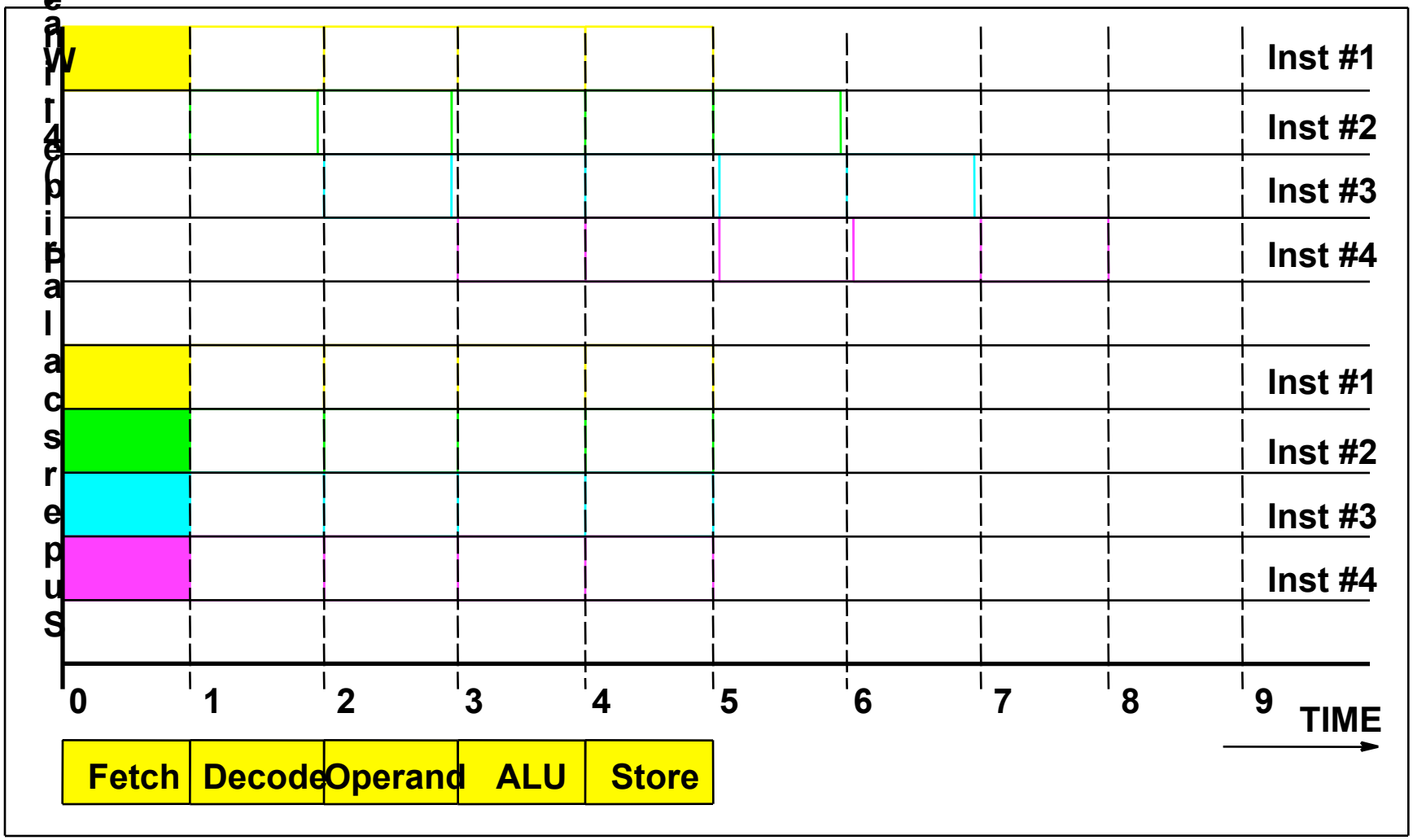

Instruction Fetch
Instruction Decode
Operand Fetch
ALU Operation
Store Result
Inst #1
Inst #2
Inst #1
Inst #2
Inst #1
Inst #2
Inst #1
Inst #2
Inst #1
Inst #1,2,3,4
Inst #1,2,3,4
Inst #1,2,3,4
Inst #1,2,3,4
Inst #1,2,3,4
Time (Instructions executed)


Inst #1
Inst #2
Inst #3
Inst #4
Inst #1
Inst #2
Inst #3
Inst #4
0
1
2
3
4
5
6
7
8
9
TIME

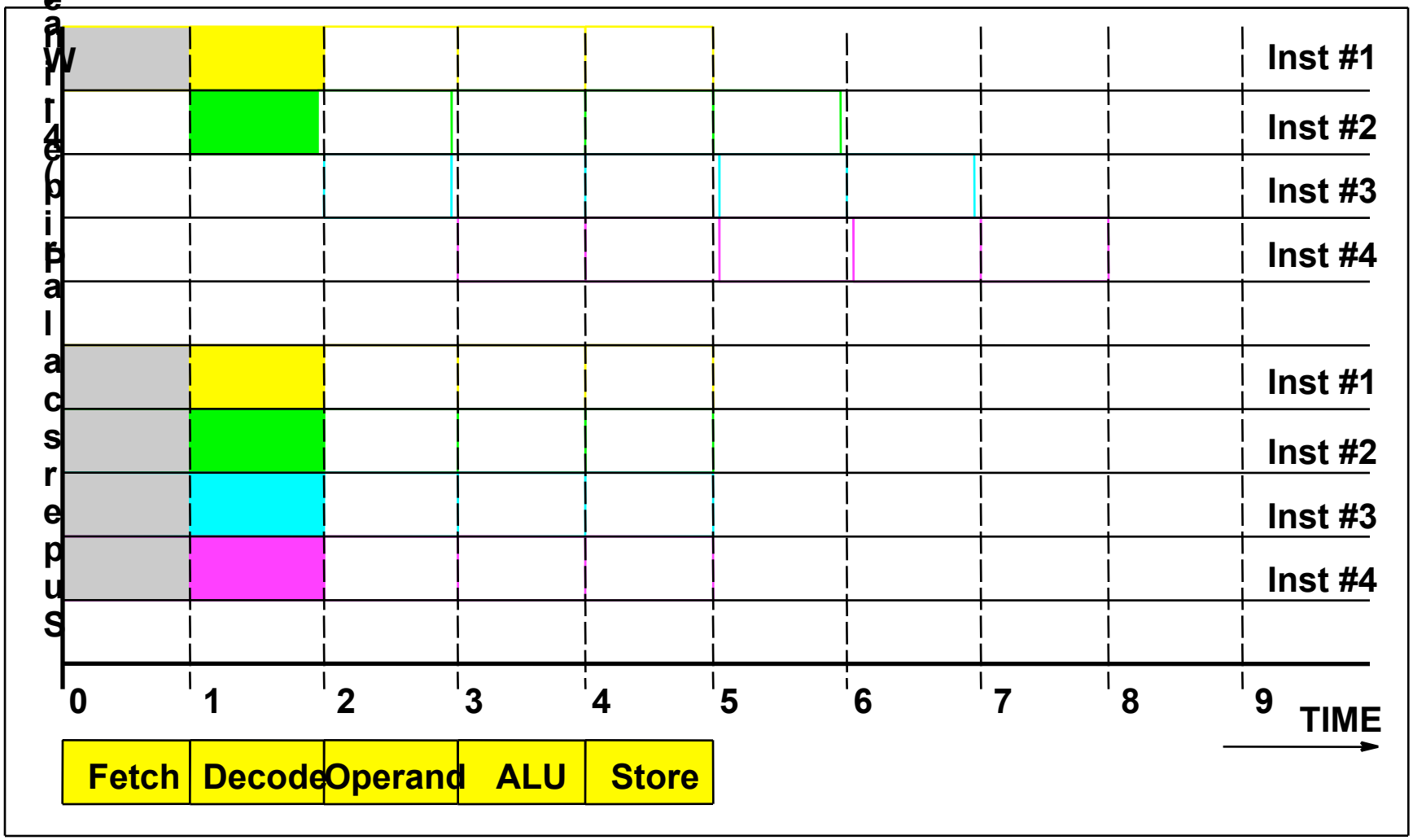

Inst #1
Inst #2
Inst #3
Inst #4
Inst #1
Inst #2
Inst #3
Inst #4
0
1
2
3
4
5
6
7
8
9
TIME
Fetch
Decode
Operand
ALU
Store


Inst #1
Inst #2
Inst #3
Inst #4
Inst #1
Inst #2
Inst #3
Inst #4
0
1
2
3
4
5
6
7
8
9
TIME
Fetch
Decode
Operand
ALU
Store

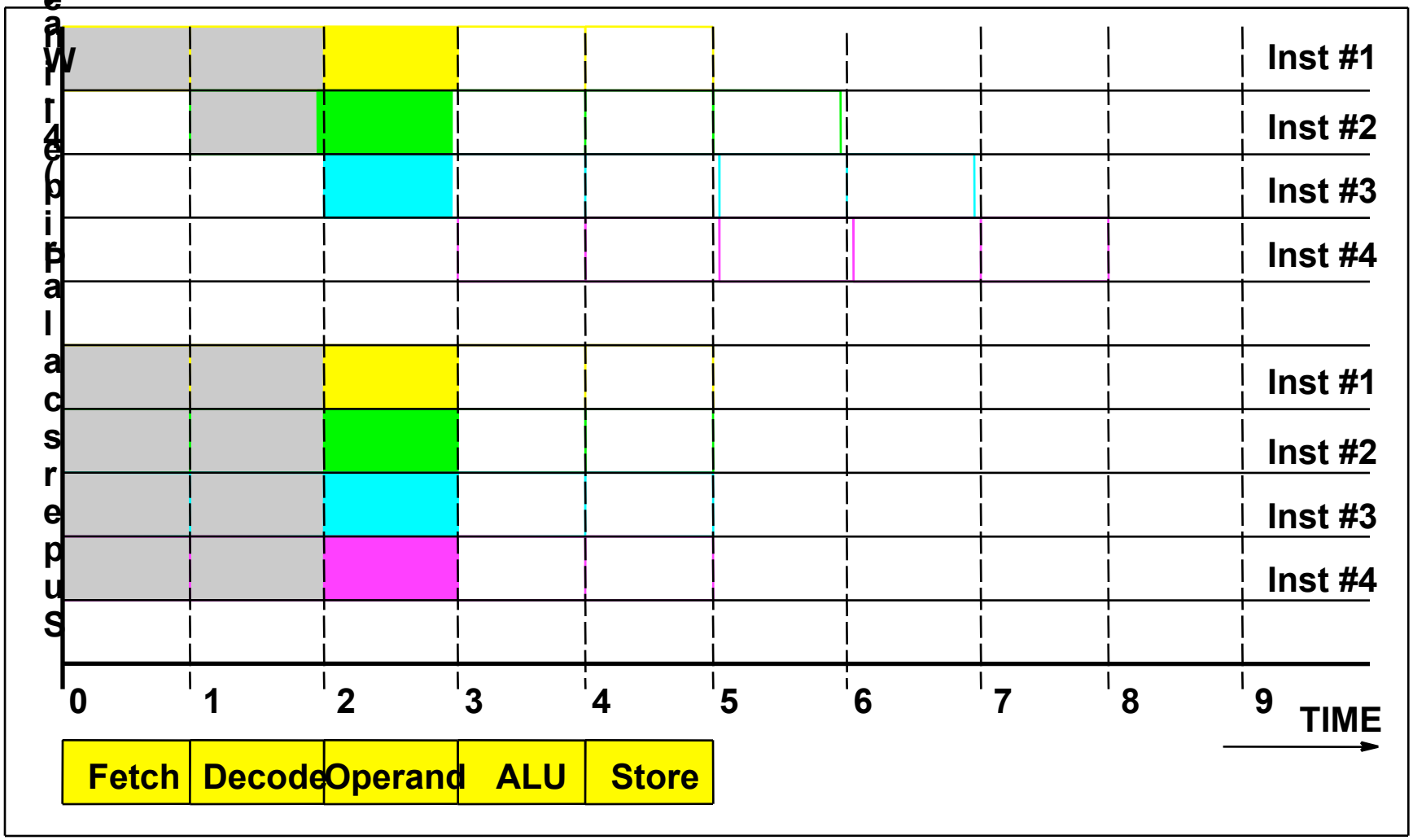
Time Slot 1


Inst #1
Inst #2
Inst #3
Inst #4
Inst #1
Inst #2
Inst #3
Inst #4
0
1
2
3
4
5
6
7
8
9
TIME
Fetch
Decode
Operand
ALU
Store

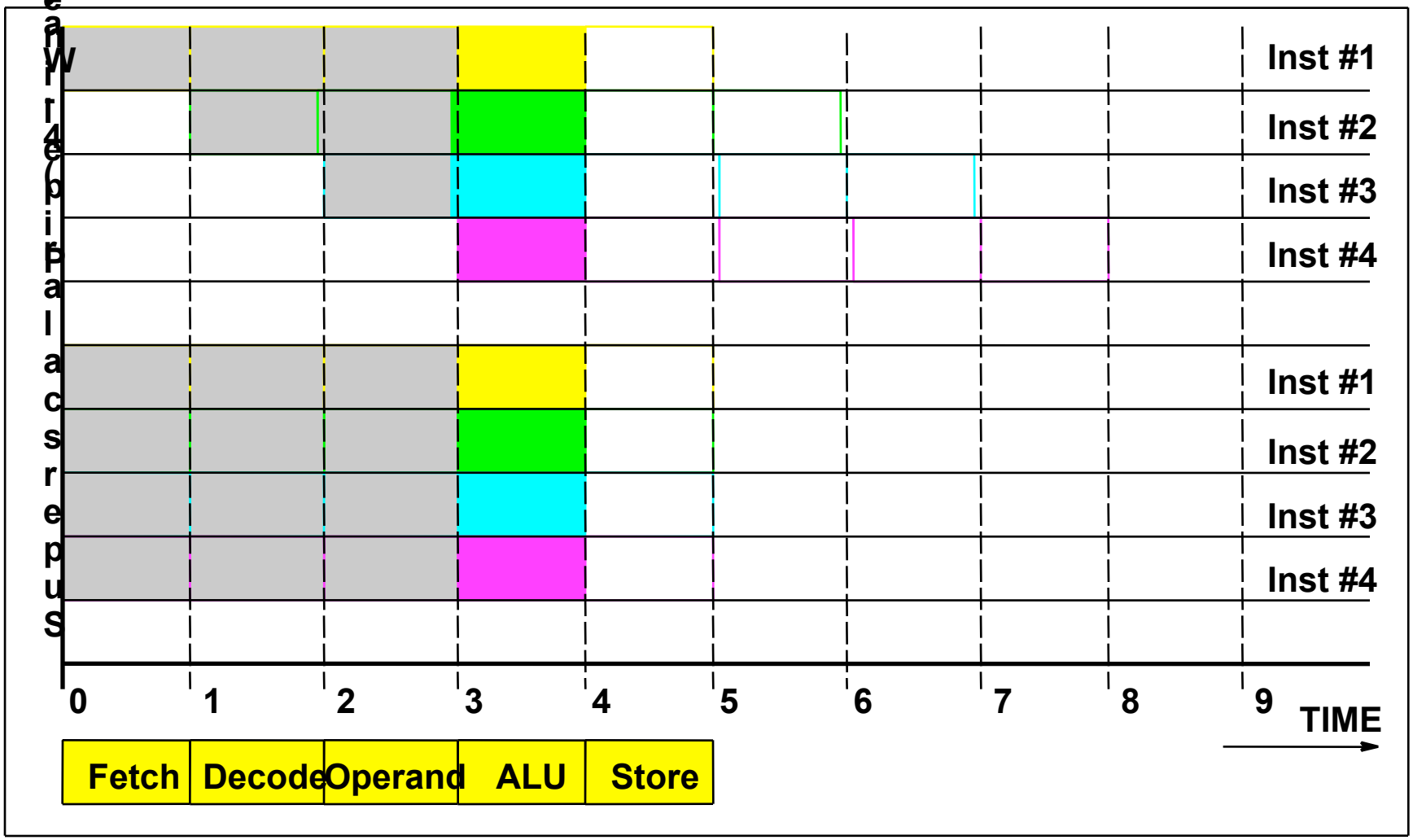
Time Slot 2


Inst #1
Inst #2
Inst #3
Inst #4
Inst #1
Inst #2
Inst #3
Inst #4
0
1
2
3
4
5
6
7
8
9
TIME
Fetch
Decode
Operand
ALU
Store

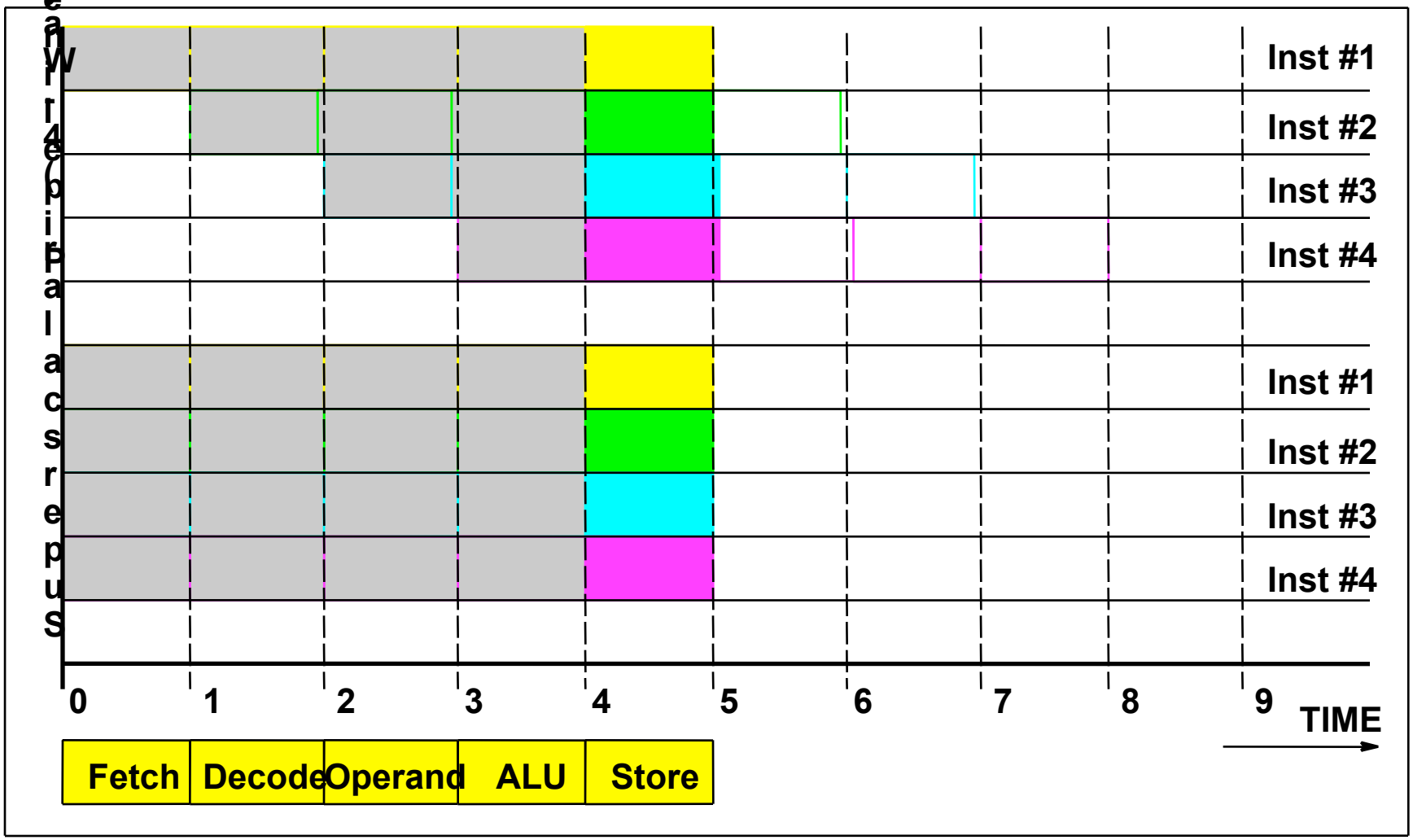
Time Slot 3


Inst #1
Inst #2
Inst #3
Inst #4
Inst #1
Inst #2
Inst #3
Inst #4
0
1
2
3
4
5
6
7
8
9
TIME
Fetch
Decode
Operand
ALU
Store

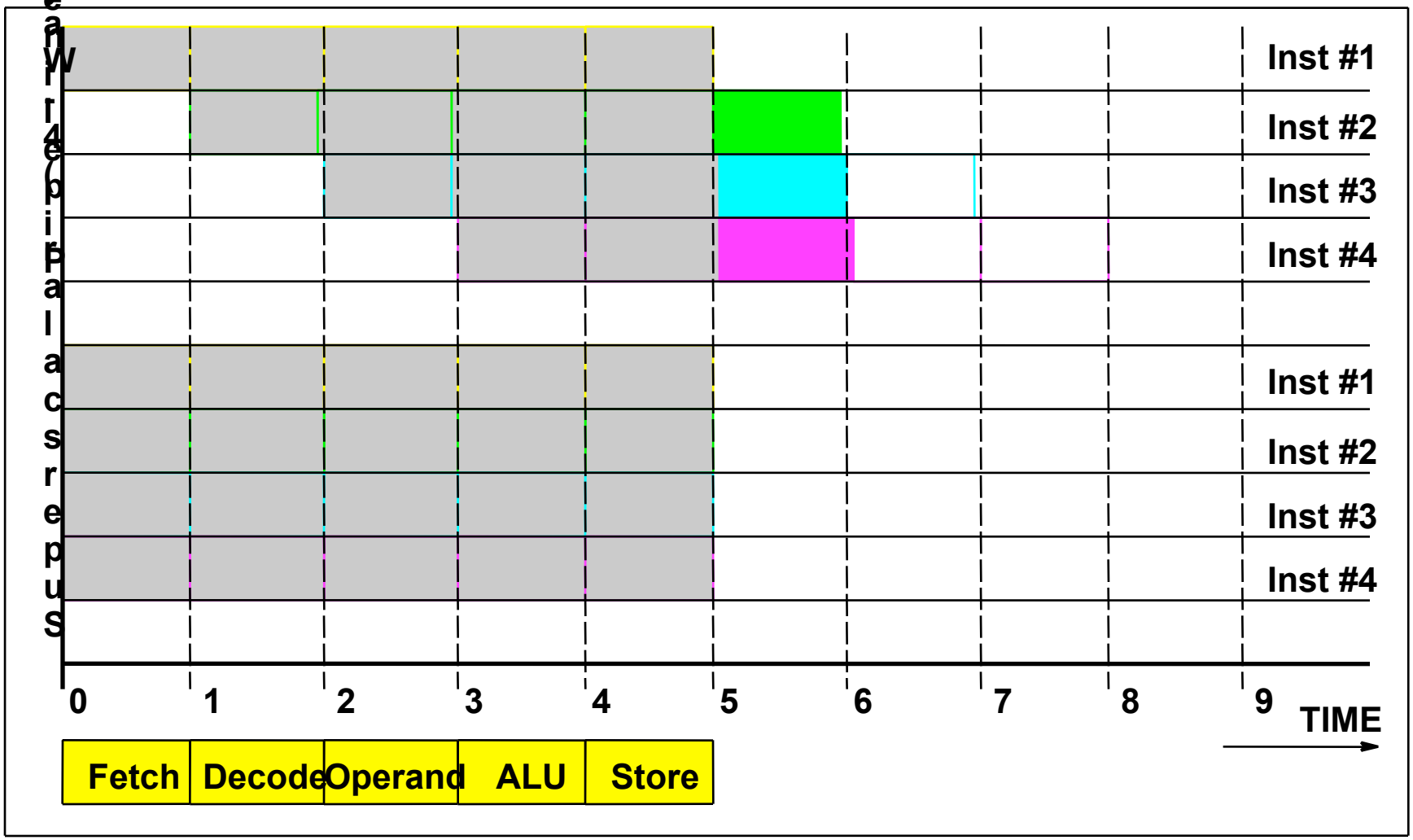
Time Slot 4



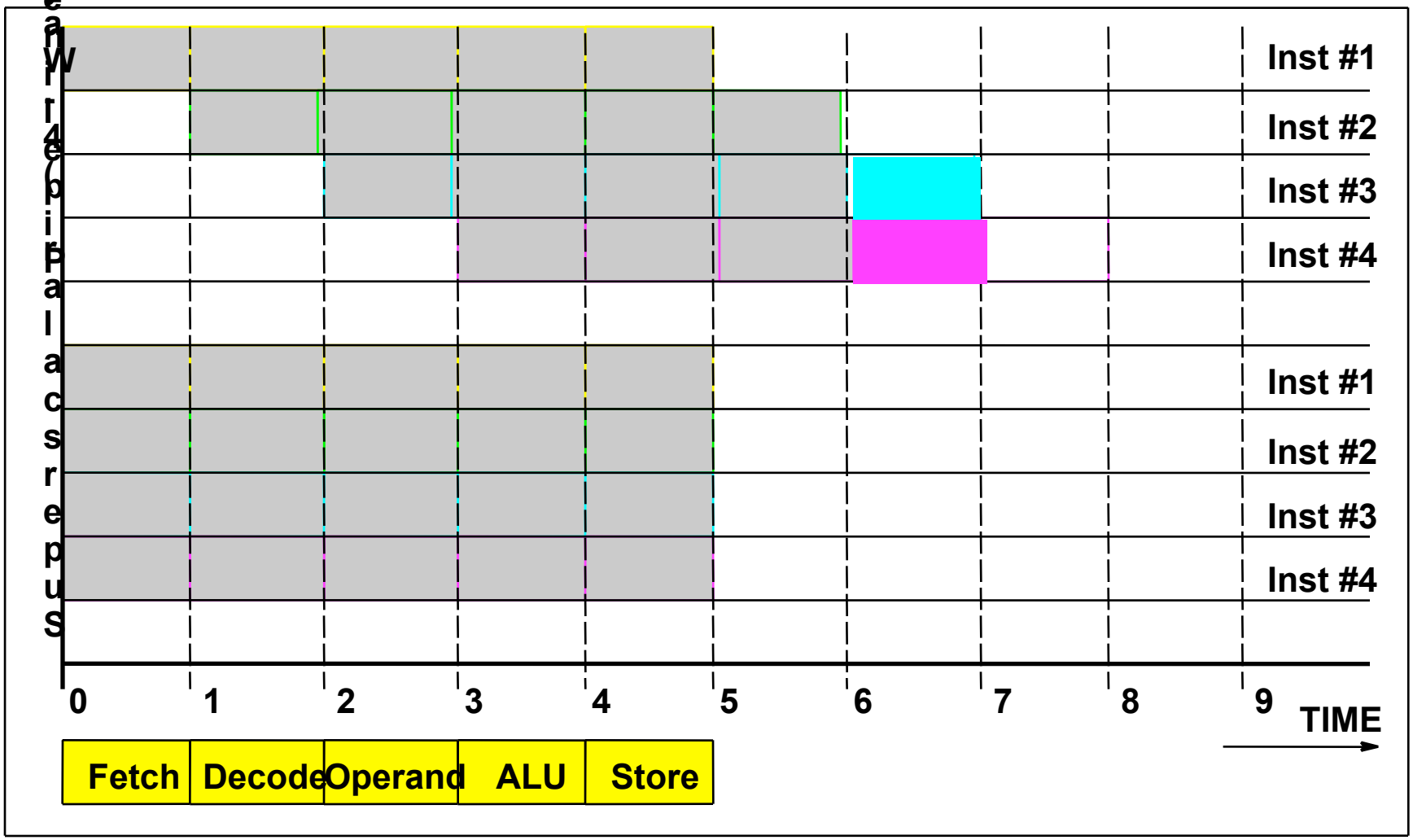
Time Slot 5



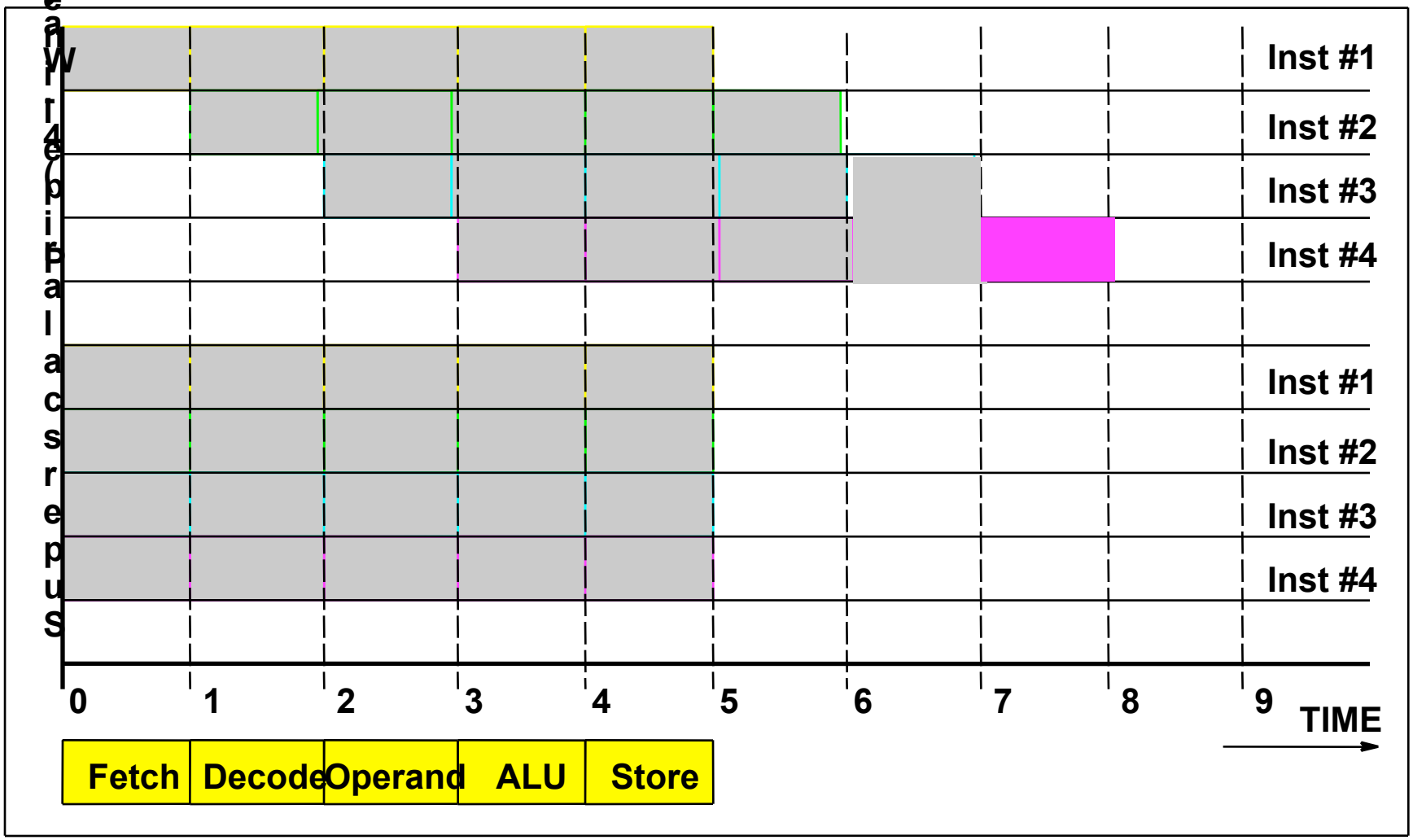
Time Slot 6



Time Slot 7


Inst #1
Inst #2
Inst #3
Inst #4
Inst #1
Inst #2
Inst #3
Inst #4
0
1
2
3
4
5
6
7
8
9
TIME
Fetch
Decode
Operand
ALU
Store

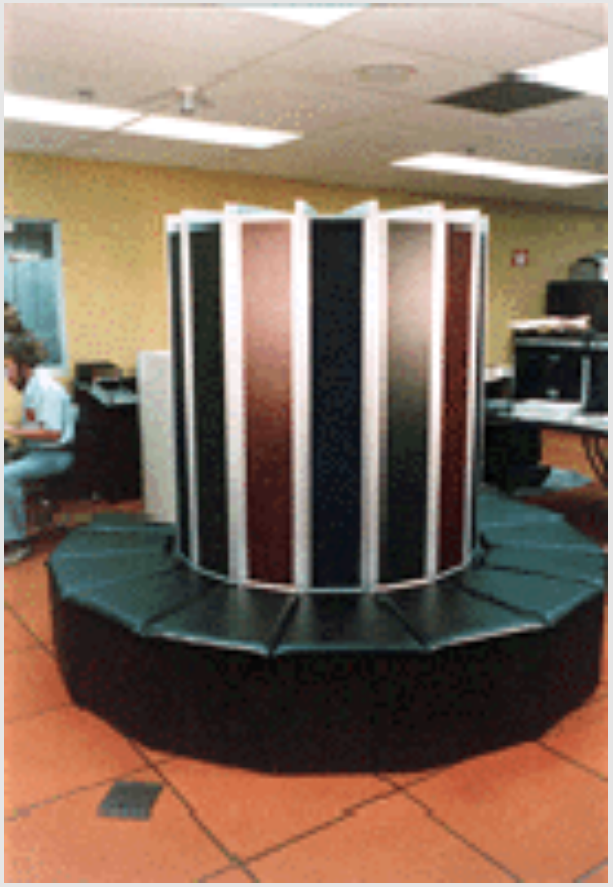
Time Slot 8

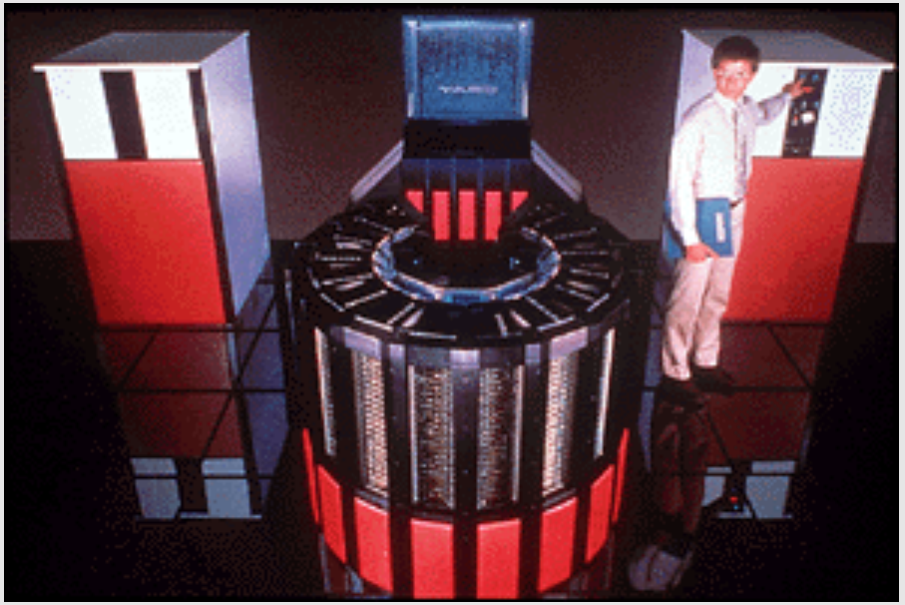

Inst #1
Inst #2
Inst #3
Inst #4
Inst #1
Inst #2
Inst #3
Inst #4
0
1
2
3
4
5
6
7
8
9
TIME
Fetch
Decode
Operand
ALU
Store

Early Examples

- **CDC 6600 supercomputer - 10 (4)-way superscalar (1963)**
- **IBM 360/91 mainframe - 3-way superscalar (1967).**
- **Cray-1 supercomputer - 13-way superscalar (1975)**
  - NB: due to the high cost of adding additional pipeline and ALU hardware, superscalar techniques were only used in the largest and most expensive machines of the period.

Cray 1 & 2 Supercomputers

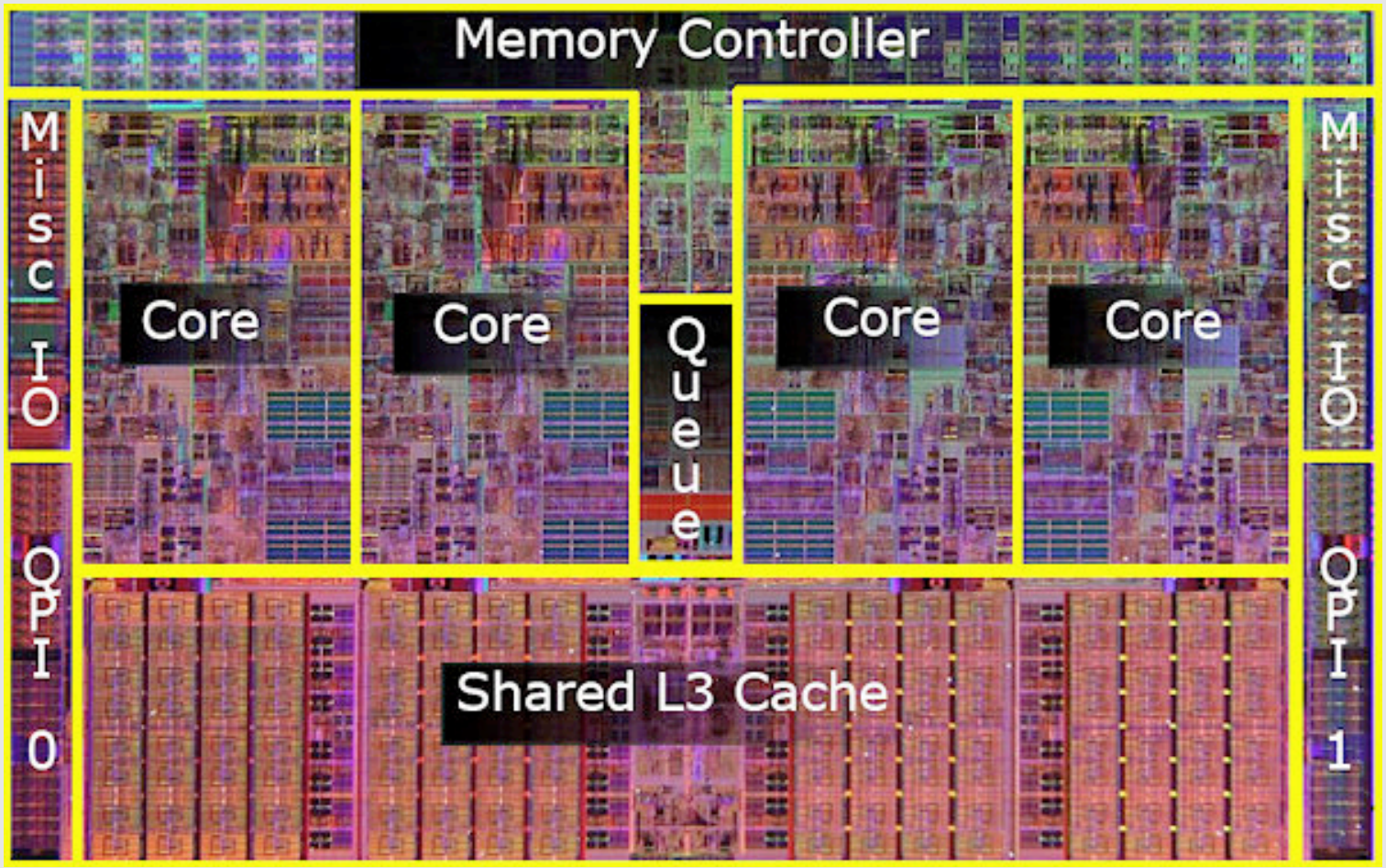




Recent Examples

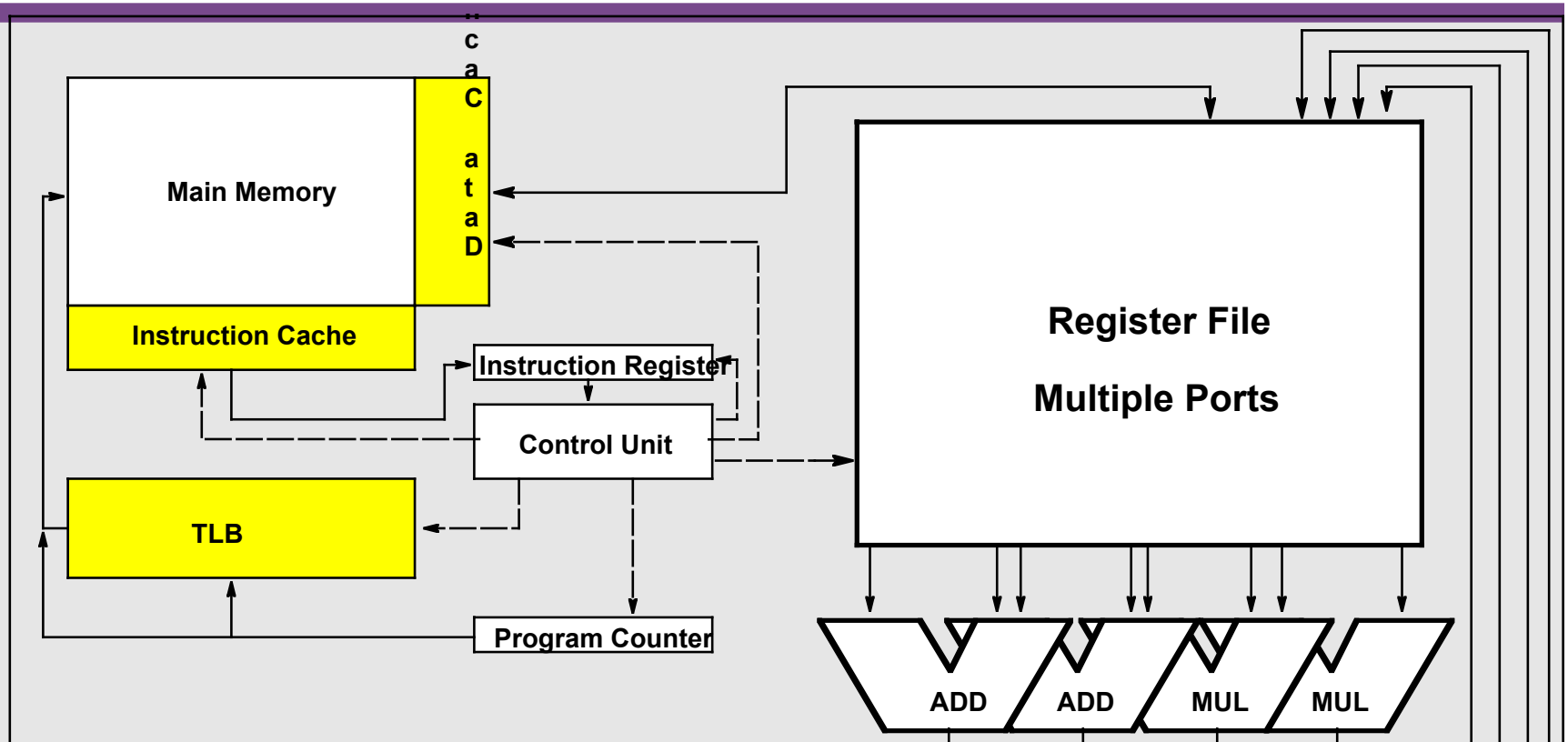
- Intel Pentium-I - 2-way superscalar (1993).
- Sun SuperSPARC/Viking - 3-way superscalar (1993).
- Intel Pentium-Pro/Pentium-II/III - 5 way superscalar (1996-1999).
- AMD Athlon/K7 - 9-way superscalar (1999).
  - NB: the increasing number of transistors on microprocessor dies allowed by the early nineties the incorporation of superscalar techniques.
  - 1960s supercomputer ~ 1990s desktop.
- Intel Core i7 Extreme Edition 32 nm
  - 12-way superscalar (2010 – 2011)

Intel Core i7 CPU



Superscalar Processing

- In a superscalar CPU it is customary to call the ALU hardware an Execution Unit (EU).
- There is no requirement for the EUs to be identical, in most superscalar CPUs there are several types of EU, e.g. Integer/Logical, Address Arithmetic and Floating Point.
- The performance gain over a CPU with a single EU can as little as 50%, in practice a 200% to 500% improvement is common.



Assymetric 4-Way Superscalar Variant of "Basic CPU"
2 x ADD and 2 x MUL can be processed concurrently
Additional pipeline hardware not drawn

Superscalar Processing

- To achieve maximum performance gain it is necessary to efficiently detect ILP in the executable code.
- Detecting ILP requires that the CPU analyse the instruction stream to find opportunities to execute instructions in parallel.
- Up to 25-30% of the hardware in a contemporary superscalar CPU may be used for this purpose.

Performance Limitations in SSP

- **Several factors can impair the performance of a superscalar CPU:**

Data (Output) Dependency.

Procedural Dependency.

Resource Conflicts.

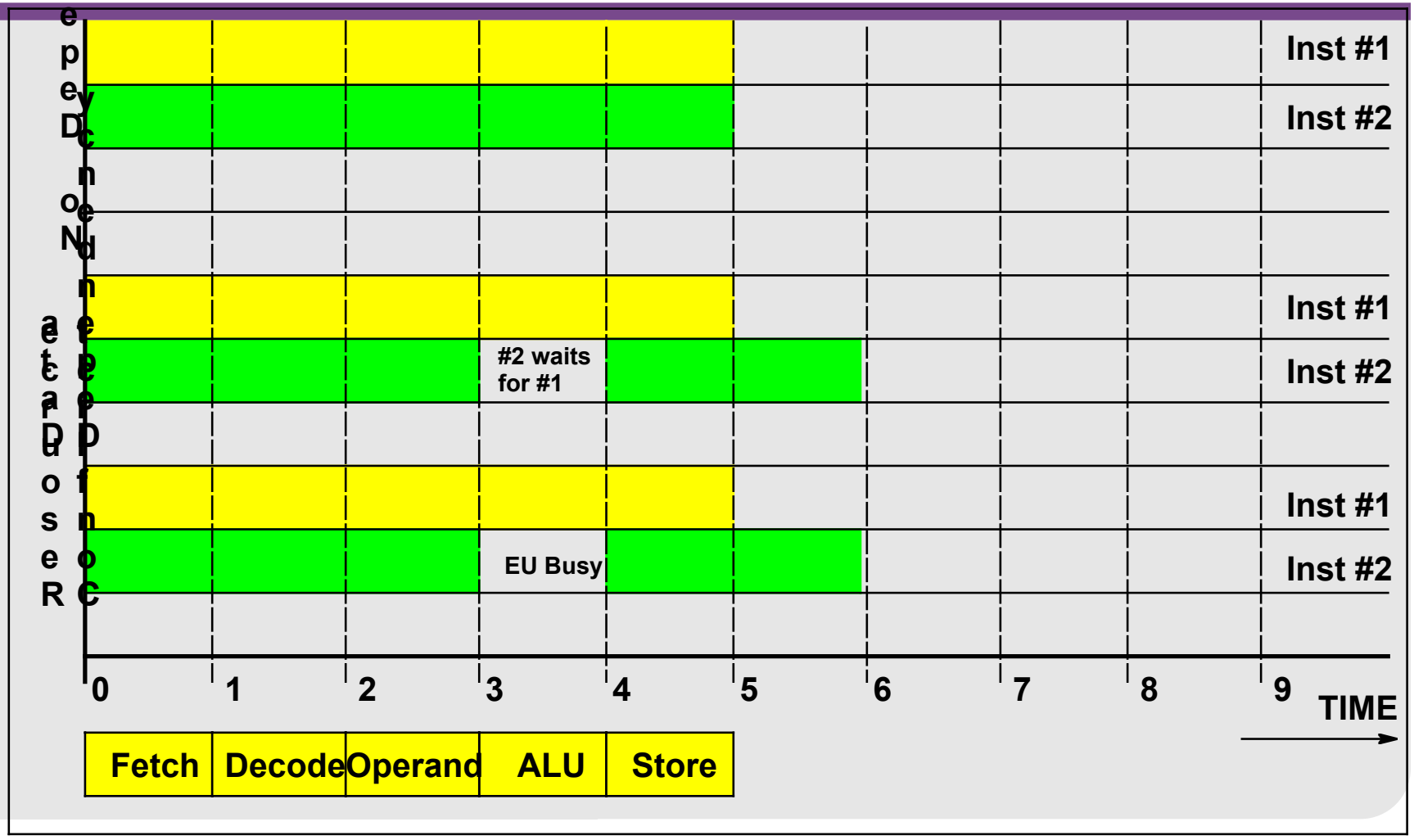
Anti-dependency.

Data Dependency

- Essentially the same type of problem as a “data hazard” seen in a pipeline.
- The result from instruction "N" may not be ready when instruction "N + 1" needs to use it.
- Handled by stalling the pipeline until the result is available, or by result forwarding where possible.
- Many data dependencies are inherent in the program being executed and cannot be avoided.

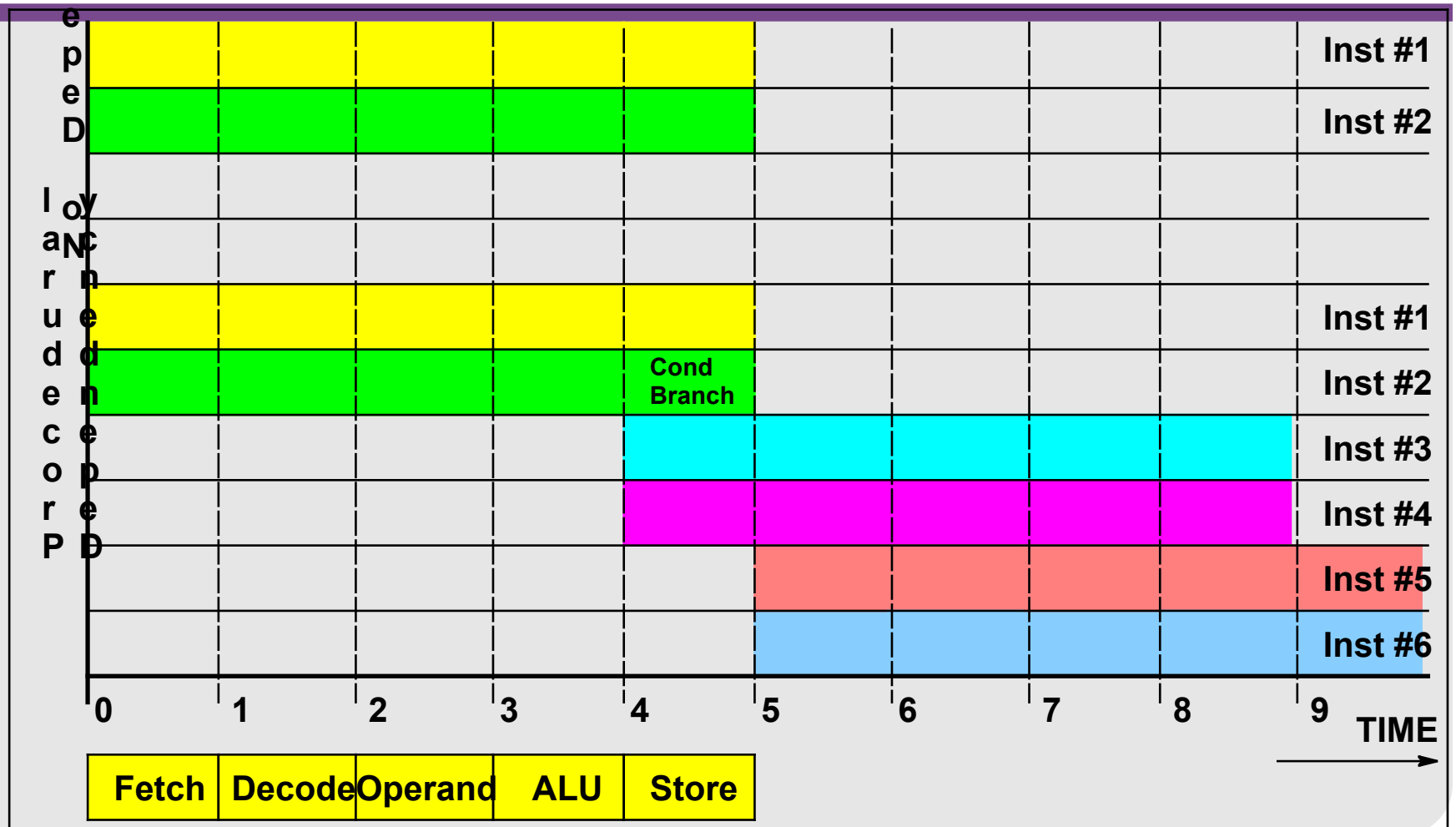
Resource Conflicts

- Resource conflicts arise when more than one instruction needs to use an EU, register or bus.
- This is essentially a “deadlock” situation since one or more instructions must wait for the resource to complete its work.
- Resource conflicts can be removed by adding more registers, busses and EUs into the CPU.
- This incurs much additional cost in hardware.


Pipelined Execution
Non-pipelined Execution
Inst #1
Inst #2
Inst #1
Inst #2
#2 waits for #1
Inst #1
Inst #2
EU Busy
0
1
2
3
4
5
6
7
8
9
TIME
Fetch
Decode
Operand
ALU
Store

Procedural Dependency

- The procedural dependency problem is essentially the same problem seen in a pipeline due to a conditional branch.
- Until the result of the branch is known, the pipelines are stalled.
- In a superscalar CPU, many more instructions are delayed, in comparison with a single pipeline, therefore the performance loss is much greater.


Pipelined Processor
Non-pipelined
Inst #1
Inst #2
Inst #1
Inst #2
Cond Branch
Inst #3
Inst #4
Inst #5
Inst #6
0
1
2
3
4
5
6
7
8
9
TIME
Fetch
Decode
Operand
ALU
Store

Speculative Execution

- Procedural dependency can be handled by "speculative execution", where the CPU calculates both possible instruction streams following the branch and then throws one away, once the outcome of the branch is known.
- Speculative execution is very expensive in hardware, since two threads of execution must be followed to get one thread of results.
- Example: Intel Pentium Pro/II/III CPU.

Anti-Dependency

- Anti-dependency arises when an instruction overwrites the source operand for a preceding instruction.
- This is not a problem in a pipelined CPU since the instructions are executed in their proper order.
- In a superscalar CPU, which may use “out of order” execution, anti-dependency prevents the execution of the later instruction until the source operand has been used.

Out of Order Execution (OOE)

- If no dependency exists between instructions, there is no requirement for these instructions to be executed in the CPU in the same order as they appear in memory.
- OOE is frequently used since it allows instructions with no dependencies to be executed when "convenient" opportunities arise between instructions with dependencies.
- OOE allows optimal use of a superscalar CPU.
- Instruction results are usually written in order.

RISC vs CISC

- Complex Instruction Set Computer (CISC).
- CISC is characterised by using a large number of very complex instructions, intended to minimise memory use.
- Typical CISC instructions incorporate complex addressing modes for operands.
- CISC machines usually have small or modest register file sizes.
- CISC machines were mostly built as microcoded, but more recently have also been implemented as hardwired.

RISC vs CISC

- Reduced Instruction Set Computer (RISC).
- RISC is characterised by using a small number of very simple instruction types.
- RISC typically uses only a Load and Store instruction for memory access.
- RISC machines frequently have very large register file sizes.
- RISC machines have almost always been hardwired.
- RISC machines need a high cache hit ratio.

Why RISC ?

- By the early 1980s most machines used CISC architectures, built using microcoded control units.
- The complicated CISC instruction sets required complicated control units, which proved to be difficult to design and debug.
- To achieve higher clock speeds, hardwired control units were preferable.
- Compilers for CISC architectures seldom exploited the vast instruction set to an advantage.

RISC Objectives

- **Use small and simple instructions to allow for a very high CPU speed, using a hardwired control unit.**
- **Use simple Load/Store to move operands into large register file, all operations to work on fast registers.**
- **Rely on a sophisticated compiler to optimise the code.**
- ***Achieve performance by using fast registers, high clock speeds, and compiler optimisation.***

Examples

- RISC: MIPS, SPARC, Alpha, HP-PA, PowerPC, i860, i960.
- CISC: VAX, PDP-11, Intel i86, Motorola 68K.
- CISC with RISC-like internal macro/micro-instructions: Pentium, AMD Athlon.

What After RISC ?

- Contemporary RISC and CISC CPUs are mostly superscalar with hardwired control.
- The economics of finding and exploiting ILP incur large overheads in CPU hardware, and significantly complicate the internal design.
- To discover more ILP, the CPU must look further ahead into the instruction stream, with every branch in the code the number of possible streams to be speculatively executed will double.
- Similar problem to “chess game” lookahead.

VLIW Architectures

- Very Large Instruction Word (VLIW) – uses many execution units in parallel, large VLIW instructions allow for concurrent operations on multiple Eus, with multiple operands.
- Each VLIW instruction contains multiple operations and operands.
- VLIW shifts the burden of ILP discovery and exploitation to software, thereby allowing for a simpler CPU with many more EUs.
- VLIW performance is critically dependent on the performance of the compiler/translator.

VLIW and CISC/RISC Emulation

- VLIW CPUs can be used to emulate CISC and RISC architectures ie backward compatibility.
- This is performed using “dynamic translation” methods, where multiple CISC or RISC instructions are translated into a single VLIW instruction, in software, during execution.
- A VLIW CPU will usually cache the CISC/RISC to VLIW translations to improve performance.
- *Translation cache hit ratio is very important.*
- Examples: Intel IA-64 Itanium, Transmeta Crusoe.

Multi Processing

- An alternative technique used for parallel processing is the use of multiple CPUs, which share a common main bus, main memory and I/O hardware;
- A multiprocessor allows completely separate programs to be run on each of the CPUs;
- Asymmetrical MP is where the OS is run on one CPU, and user programs on the others;
- Symmetrical multiprocessing is where both the OS and user programs run on any CPU;
- Early multiprocessors had dedicated cards, each with a single CPU chip;
- Multicore CPU chips, initially dual-core, appeared ~2005, followed by quad-core, and more recently many cores;

Issues in Multiprocessing - Cache Coherency

- ***Bus throughput*** **can limit the number of CPUs in the system. If the bus cannot handle the volume of accesses to memory, CPUs will stall and performance is then lost;**
- ***Cache coherency*** **with memory is vital to ensure that CPUs do not modify data, unbeknownst to other CPUs;**
- **If two CPUs each hold copies of the same location in memory within their caches, a mechanism must exist to prevent them from overwriting each others' results;**
- **Numerous techniques exist for maintaining cache coherency;**
- ***Snooping techniques*** **use intelligent caches which broadcast any writes to all other caches in the system. Each cache then updates itself to reflect the change.;**

Performance Gains in Multiprocessing

- A system with N CPUs can do more useful work than a system with a single CPU, but only where the application program being executed can be divided across all N CPUs;
- In general, a program can be functionally divided into computations which are 'serial' and computations which are 'parallel';
- 'Serial' computations have some data dependencies and cannot be parallelised, unlike the 'parallel' computations;
- Assume a program has no computations with mutual dependencies. If this is true, then running the program on a machine with N CPUs results in an N-fold gain in performance as the workload can be evenly divided across all N CPUs.
- This N-fold gain in performance is termed `linear speedup'.
- ***What happens if the program comprises only computations with mutual dependencies?***

Performance Gains in Multiprocessing (2)

- **Assume a program only has computations with mutual dependencies.**
- **Then the program cannot be usefully divided across a pool of N CPUs, and no performance gain can be realised.**
- **Many computational programs fall into this category and are not suitable for multiprocessing systems.**
- ***What happens if a program contains a mix of computations, some of which are `serial' and and some of which are `parallel'?***

Amdahl's Law

- In 1967 Gene Amdahl formulated `Amdahl's Law' which describes the achievable speedup in a multiprocessing system which is executing a program with a mix of `serial' and `parallel' computations.
- (Amdahl, G.M. *Validity of the single-processor approach to achieving large scale computing capabilities.* In AFIPS Conference Proceedings vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., 1967, pp. 483-485).

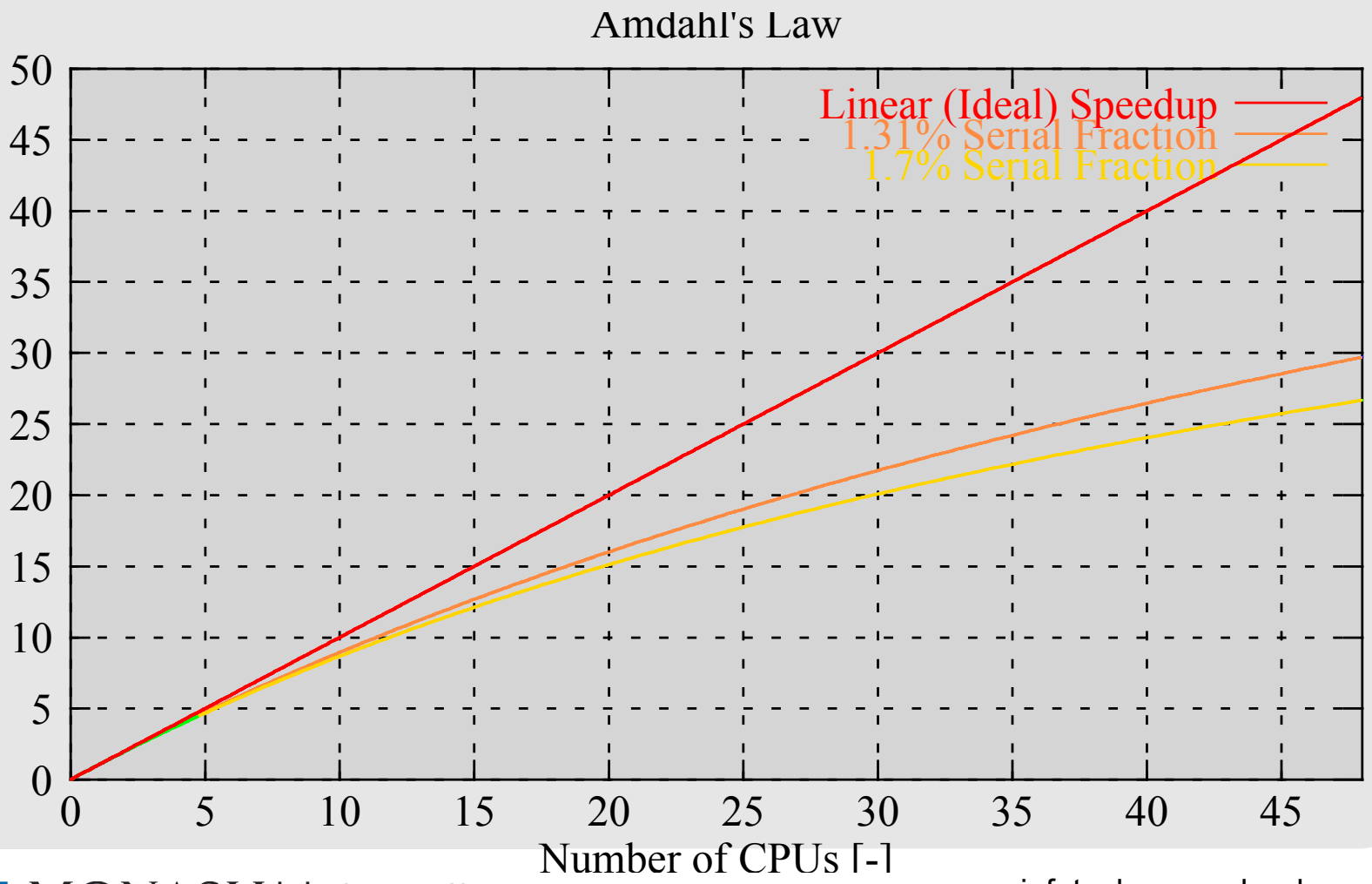
Amdahl's Law

- **Assume N is the number of processors, *s* is the time spent on serial components of a program, *p* is the time spent on parallel components of a program, where *(s+p)=1*.**
- **Then:**

$$Speedup = (s+p)/(s+p/N) = 1/(s+p/N)$$

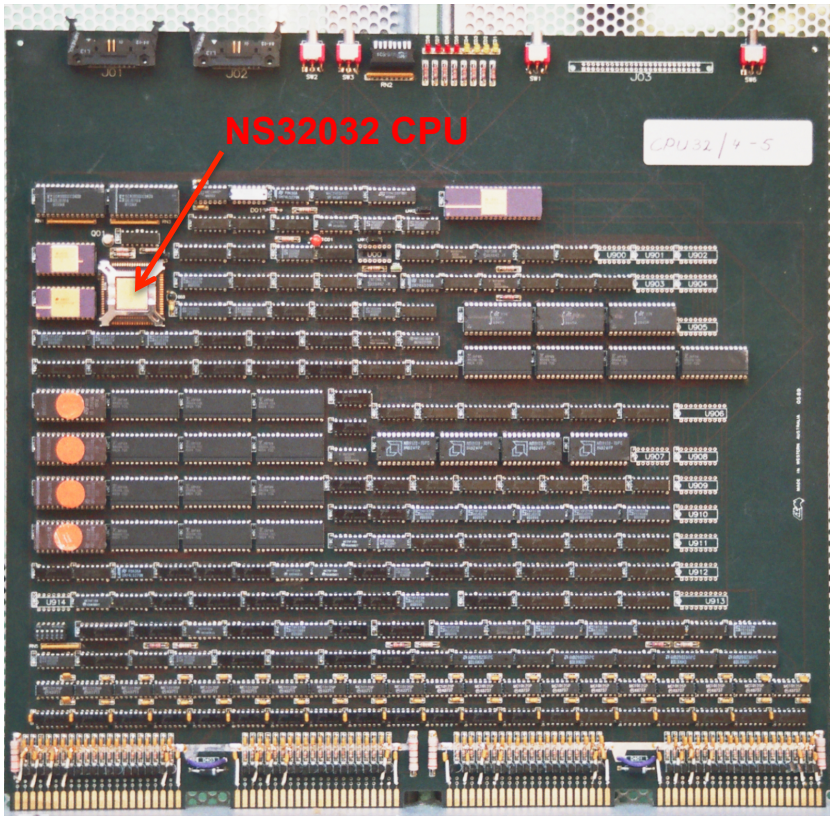
- **This expression is known as Amdahl's Law, and it shows that the performance gain in a multiple processor system depends strongly upon the behaviour of the program. Adding CPUs only makes sense when the program has a large `parallel' component.**

Amdahl's Law Example – Unix Server 1996



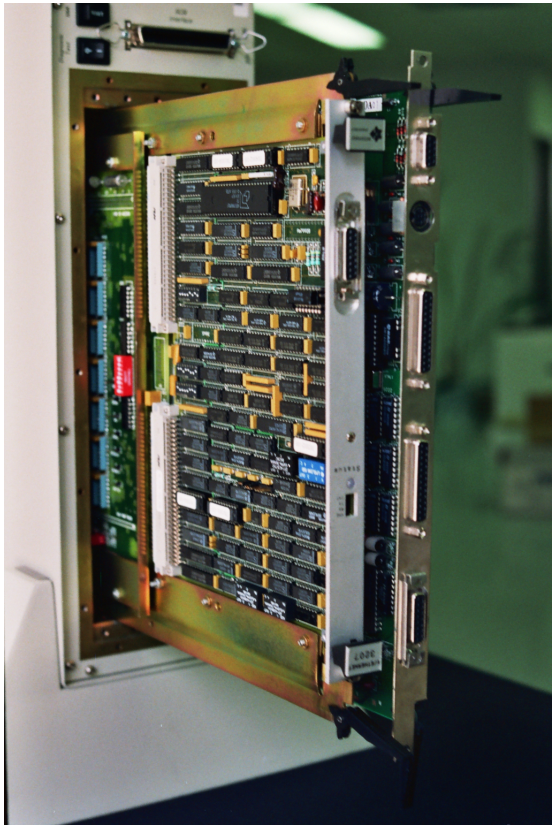
Multi Processing Examples (1989-1990)

Chris Wallace Multi 32-bit CPU Board



1 x National Semiconductor NS32032
Dept of Computer Science, Monash Uni, 1989

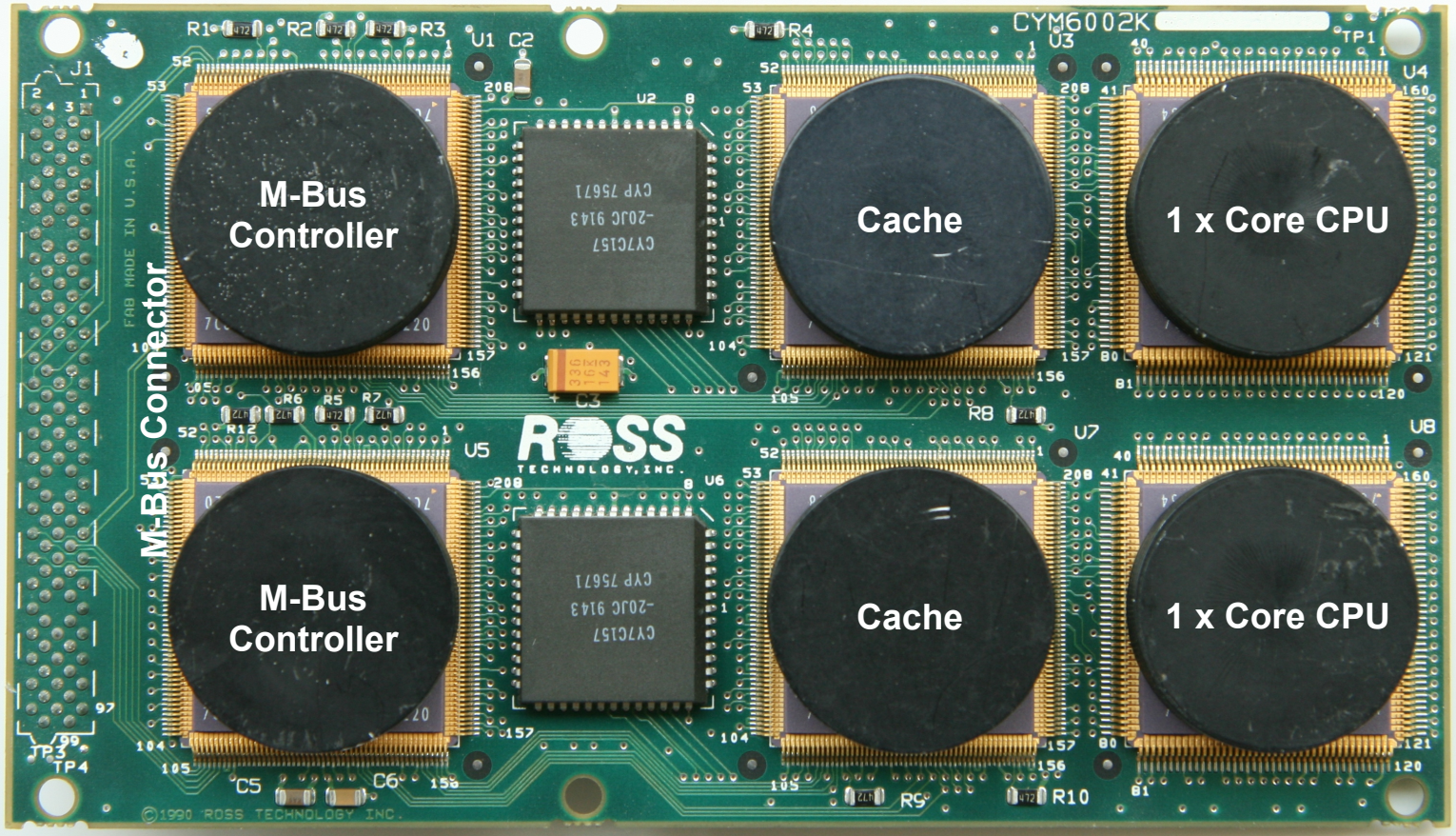
Solbourne 5/502 K-Bus Dual CPU System



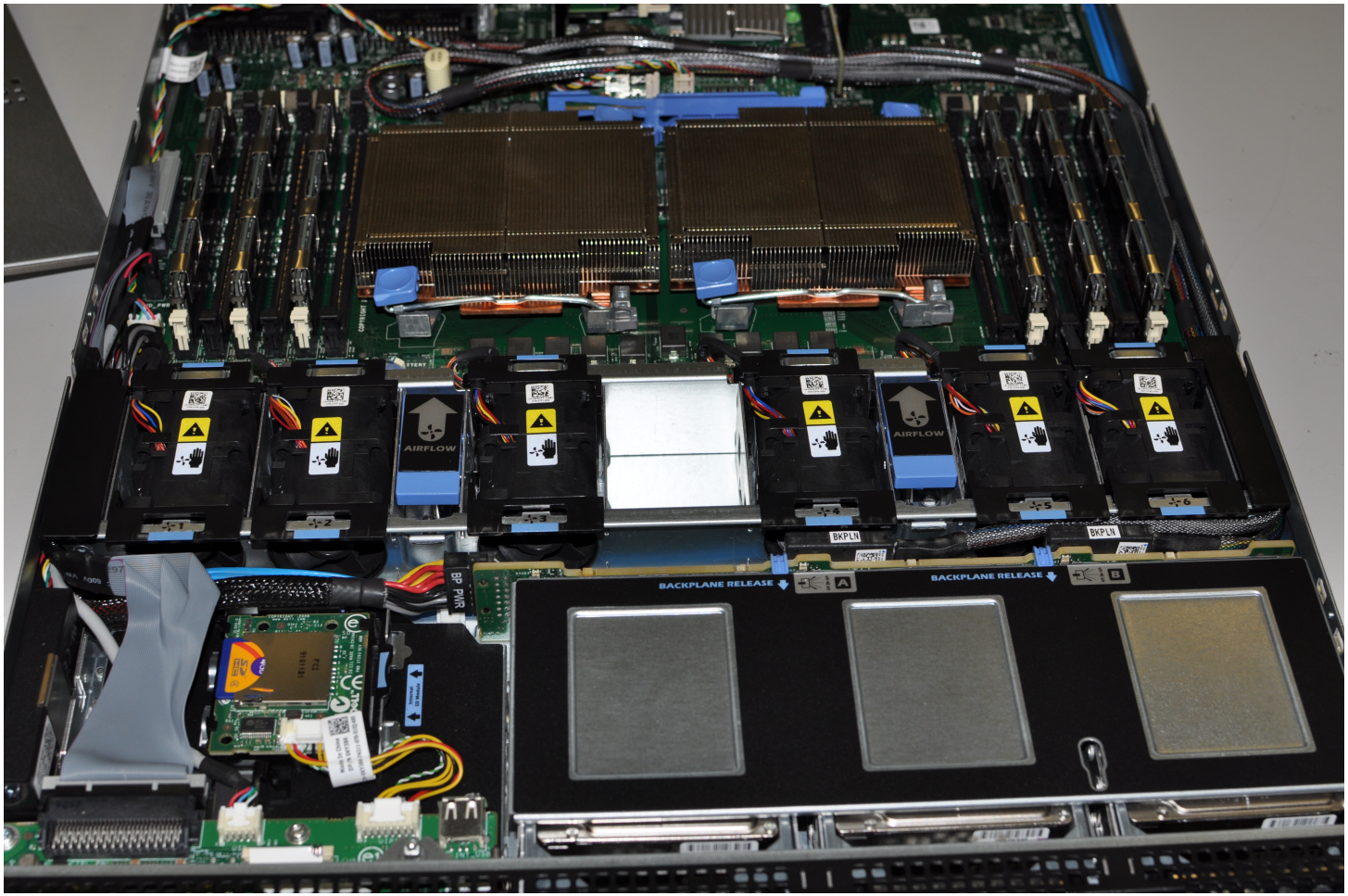
1-2 x 33 MHz Cypress CY7C601
Solbourne Computer / Matsushita 1990
www.infotech.monash.edu

2 x Single Core CPUs (1993)

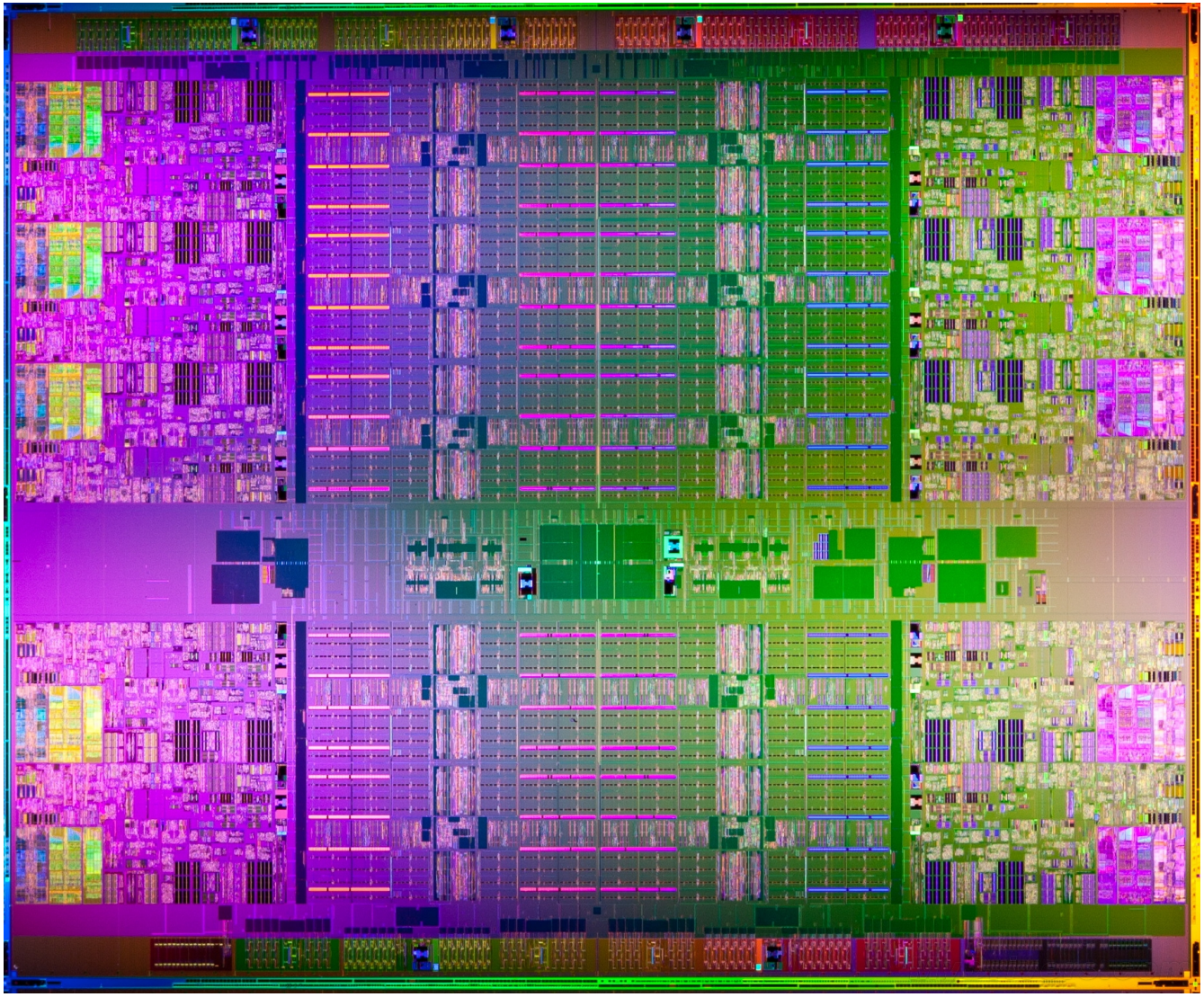
Ross-Cypress CYM6002K M-Bus Dual SPARC CPU Module



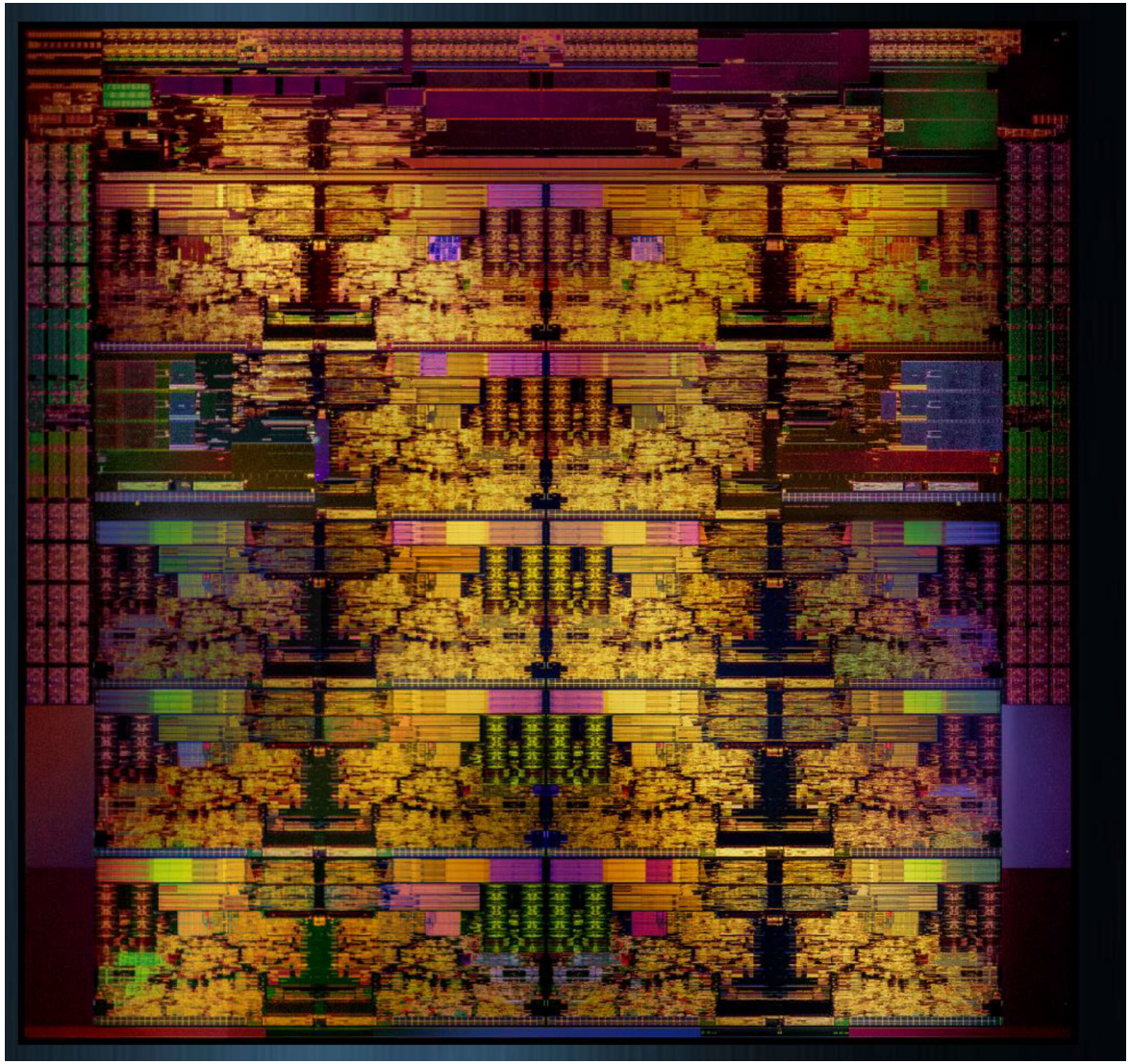
16 x i86 Cores / 4 CPU Chips - Dell Blade (2011)



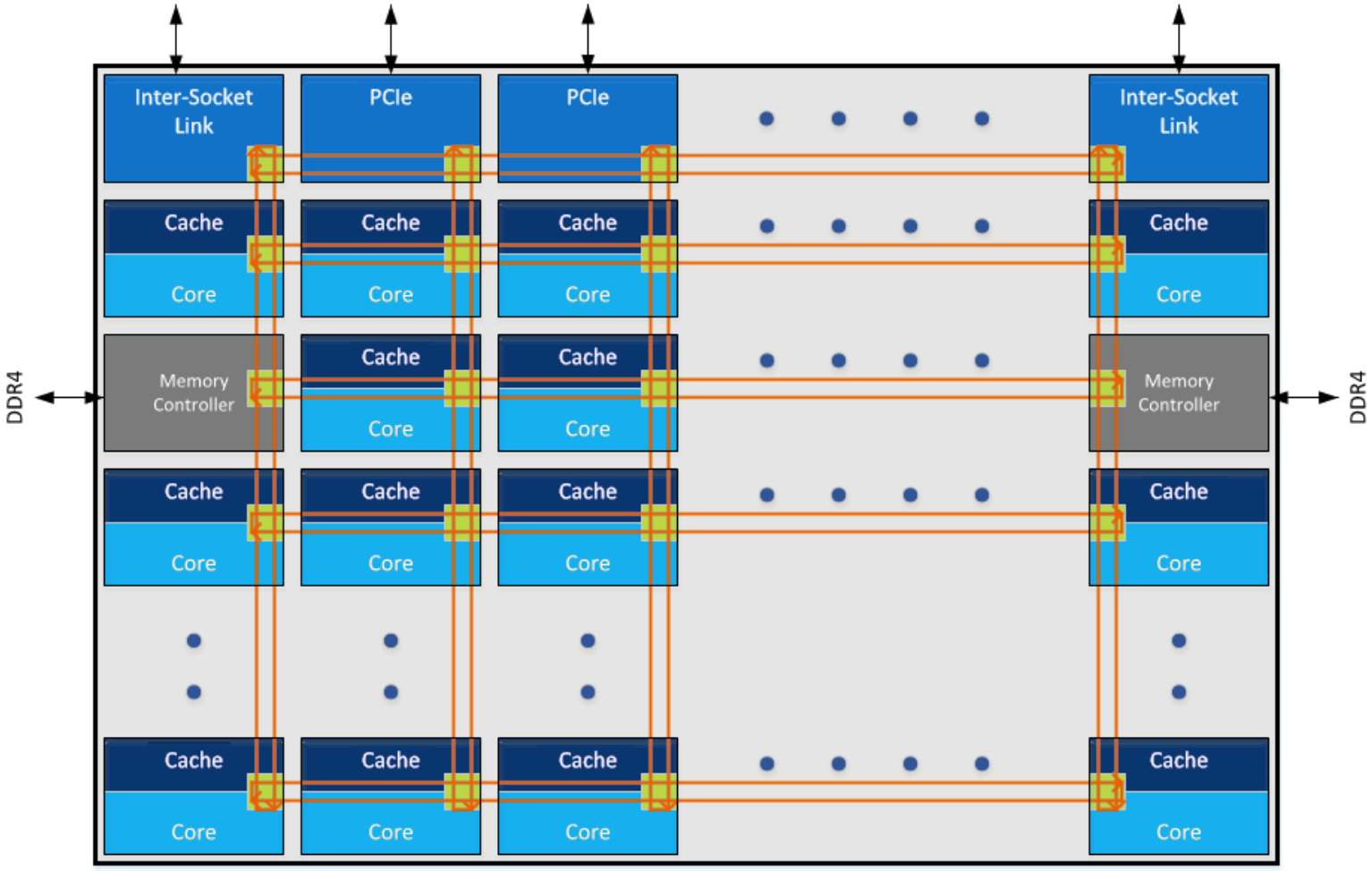
10 x Cores – Intel Xeon E7 (2014)



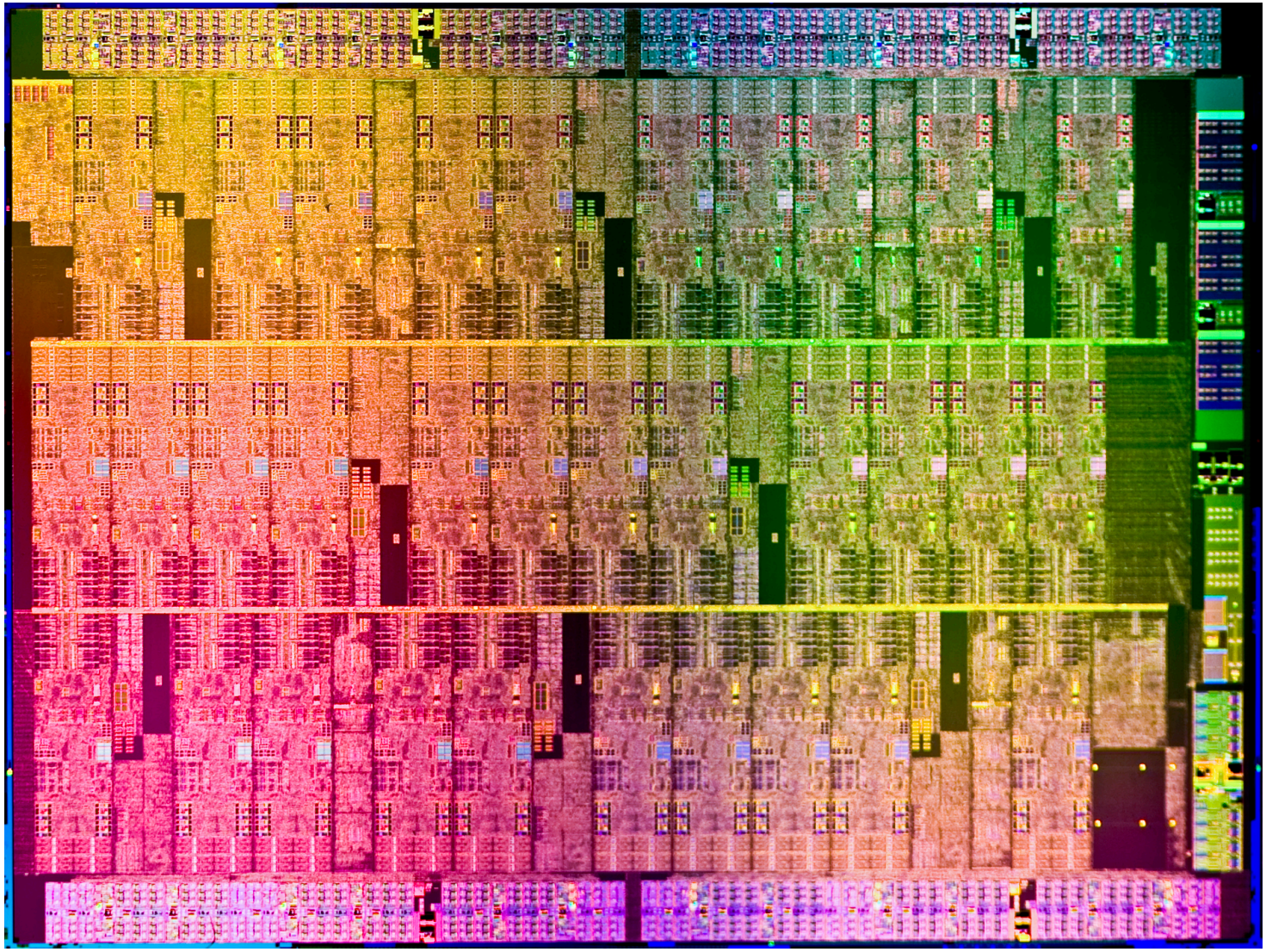
18 (20) x Cores – Intel Skylake X (2017)



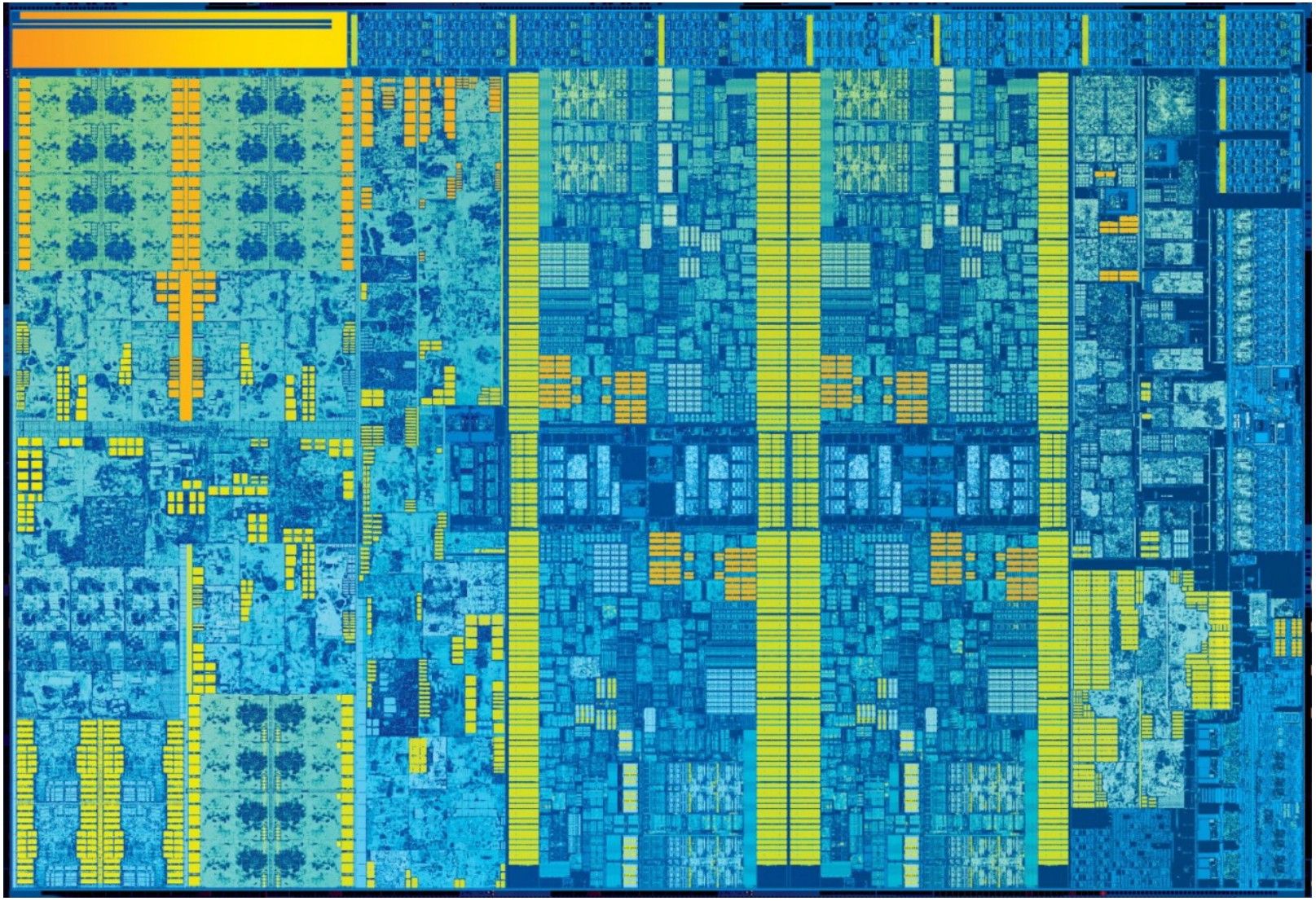
18 (20) x Cores – Intel Skylake X Mesh Bus (2017)



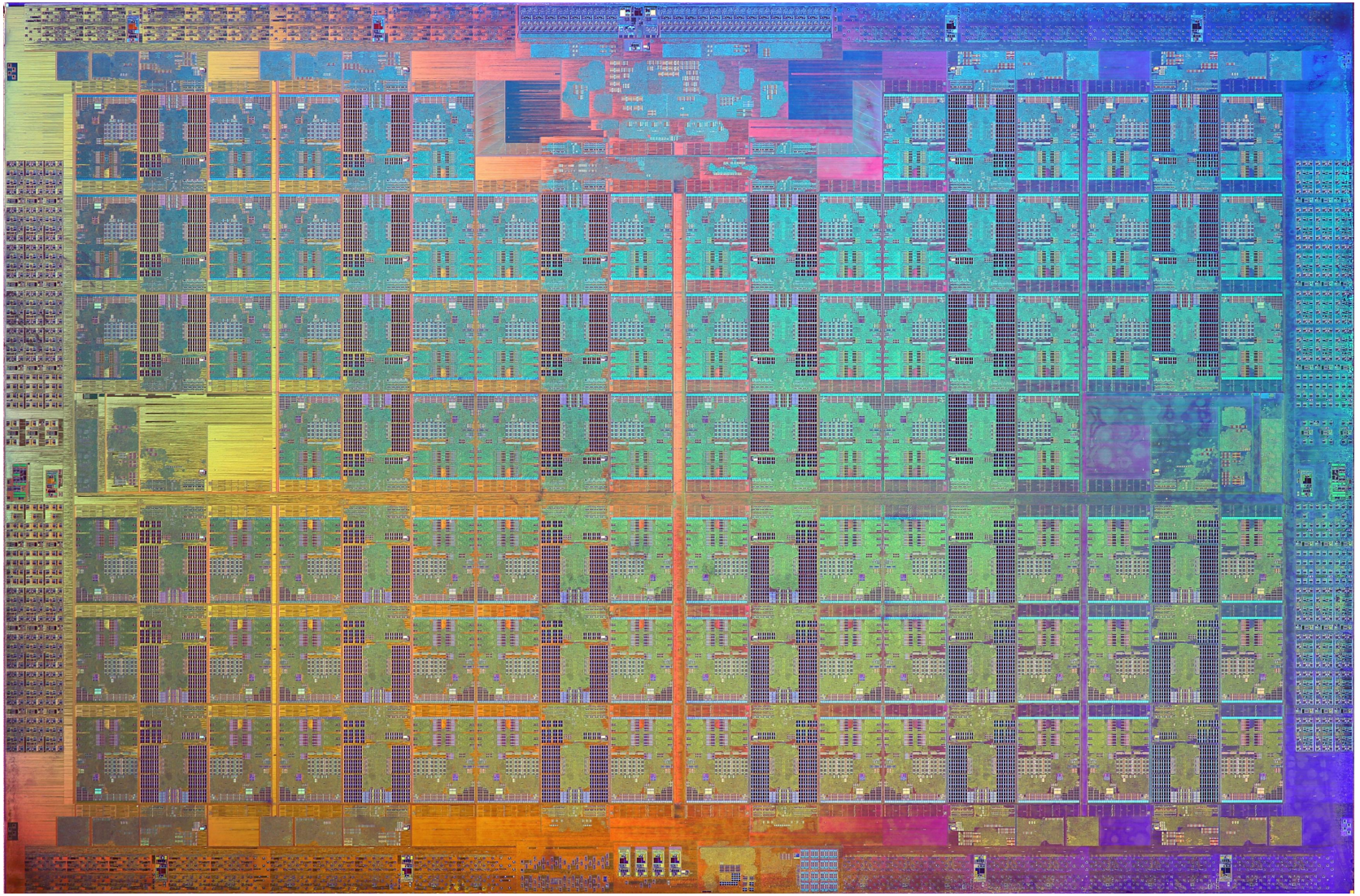
32 x Cores – Intel Xeon Phi / Aubrey Isle (2010-2012)



4 x Cores – Intel Core i7-6700K / Skylake (2016)



72 x Cores – Intel Xeon Phi / Knight's Landing (2016)



End FIT3159 Computer Architecture