CS 5803
Introduction to High Performance Computer Architecture: Beyond RISC

A.R. Hurson

323 CS Building,

Missouri S&T

hurson@mst.edu

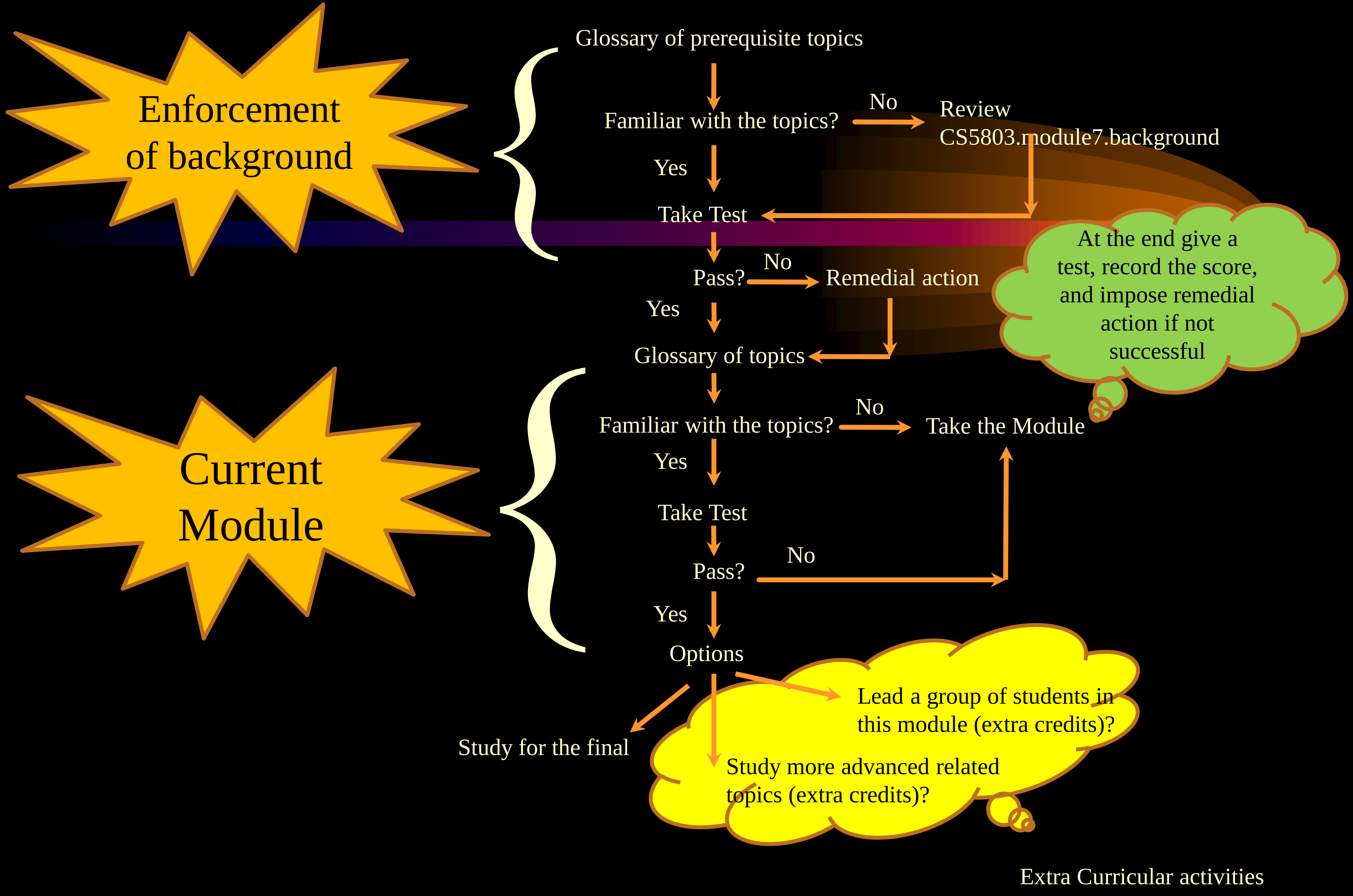
Outline

- Scalar processor
- Instruction Level Parallelism
- How to exploit instruction level parallelism
- In-order issue, In-order completion
- In-order issue, out-of-order completion
- Out-of-order issue, out-of-order completion
- Super-scalar processor
- Super-pipelined processor
- Very Long Instruction Word Computer
- Intel machines: Evolution from 8086 to Pentium III

Note, this unit will be covered in one week. In case you finish it earlier, then you have the following options:

1) Take the early test
2) Study the supplement module (supplement CS5803.module7)
3) Act as a helper to help other students in studying CS5803.module7

Note, options 2 and 3 have extra credits as noted in course outline.

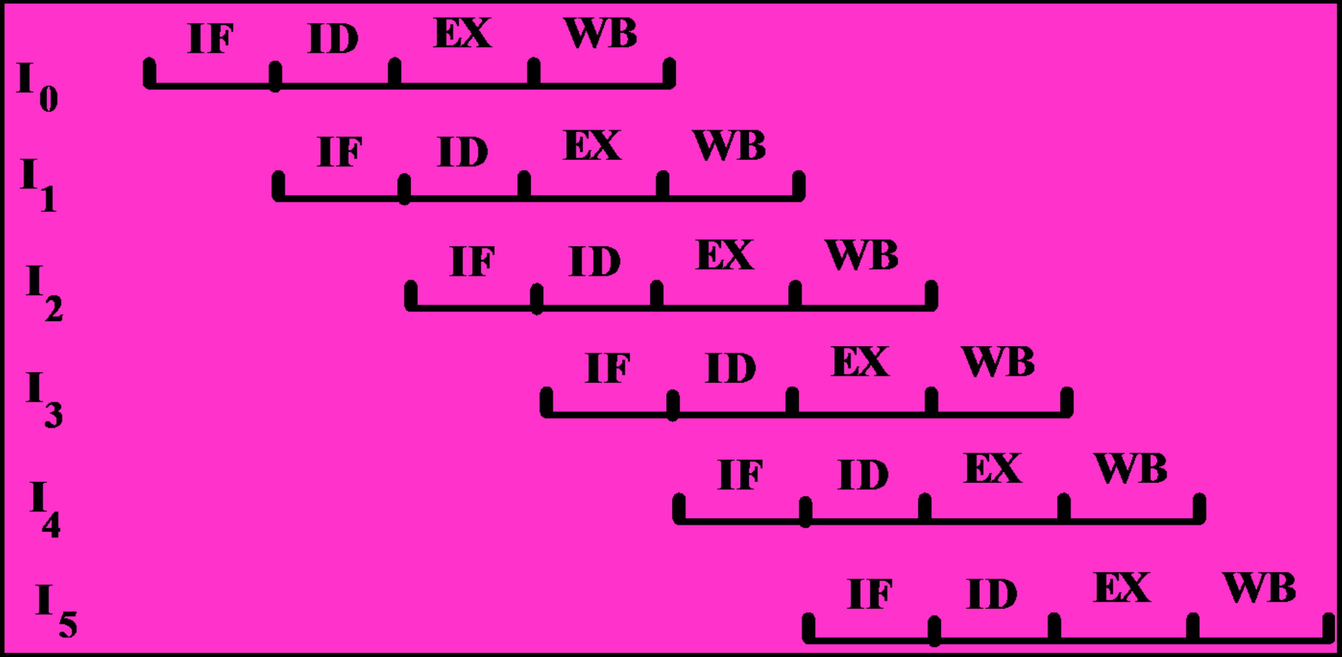

Enforcement of background
Glossary of prerequisite topics
Familiar with the topics?
No
Review CS5803.module7.background
Yes
Take Test
Pass?
No
Remedial action
Yes
Glossary of topics
At the end give a test, record the score, and impose remedial action if not successful
Current Module
Familiar with the topics?
No
Take the Module
Yes
Take Test
Pass?
No
Yes
Options
Study for the final
Lead a group of students in this module (extra credits)?
Study more advanced related topics (extra credits)?
Extra Curricular activities

Beyond RISC

- The term scalar processor is used to denote a processor that fetches and executes one instruction at a time.
- Performance of a scalar processor, as discussed before, can be improved through instruction pipelining and multifunctional capability of *ALU*.
- Refer to a *RISC* philosophy, an improved scalar processor, at best, can perform one instruction per clock cycle.

Beyond RISC

- Traditional *RISC* pipeline



Beyond RISC

- Is it possible to achieve a performance beyond what is being offered by *RISC*?

Beyond RISC

- As noted before, the CPU time is proportional to the:
  - Number of instructions required to perform an application,
  - Average number of processor cycles required to execute each instruction,
  - Processor's cycle time.

Beyond RISC

- CPU Time = Instruction count * CPI * Clock cycle time
  - CPI is the average number of clock cycles needed to execute each instruction.
- How can we improve the performance?
  - Reduce the instruction count,
  - Reduce the CPI,
  - Increase the clock rate.

Beyond RISC

- RISC philosophy attempts to improve performance by reducing the CPI through simplification. However, simplification in general, increases the number of instructions needed for a task.
- RISC designers claim that RISC concept reduces CPI at a faster rate than the increase in instruction count — DEC VAXes have CPIs of 8 to 10 and RISC machines offer CPIs of 1.3 to 3. However, RISC machines require 50 to 150 percent more instructions than VAXes.

Beyond RISC

- How to increase the clock rate?
  - Advances in technology
  - Architectural advances.

- How to reduce the CPI beyond simplicity?
  - Increase the number of operations issued per clock cycle.

Beyond RISC

- What is Instruction Level Parallelism?

- Instruction Level Parallelism (ILP) – Within a single program how many instructions can be executed in parallel?

Beyond RISC

- ILP can be exploited in two largely separable ways:
  - Dynamic approach where mainly hardware locates the parallelism,
  - Static approach that largely relies on software to locate parallelism.

- Summary
  - RISC barrier
  - Scalar processor
  - Instruction Level Parallelism
  - Instruction Issue/Instruction Completion order
  - Dependence graph (program graph)
  - Super Pipeline
  - Superscalar
  - Very Long Instruction Word

Super Scalar System

- Beyond RISC
  - Fundamental Limitations
    - Data Dependency
    - Control Dependency
    - Resource Dependency

Super Scalar System

- Data Dependency
  - Within the scope of data dependency we can talk about:
    - Read after write (flow) dependency
    - Write after read (anti) dependency
    - Write after write (output) dependency
  - The literature has referred to read after write as true dependency, and write after read or write after write as false dependency.

Super Scalar System

- Practically, write after read and write after write are due to storage conflict and originated from the fact that in the traditional systems we are dealing with a memory organization that is globally shared by instructions in the program.

- Storage medium holds different values for different computations.

Super Scalar System

- The processor can remove storage conflict by providing additional registers to reestablish one-to-one correspondence between storage (register) and values — register renaming.

Super Scalar System

- Two constraints are imposed by control dependencies:
  - An instruction that is control dependent on a branch cannot be moved before the branch,
  - An instruction that is not control dependent on a branch cannot be moved after the branch.

Beyond *RISC*

- Resource Dependence
  - A resource conflict arises when two instructions attempt to use the same resource at the same time. Resource conflict is of concern in a scalar pipelined processor.

Beyond RISC

- Straight line code blocks are between four to seven instructions that are normally dependent on each other — degree of parallelism within a code block is limited.
- Several studies have shown that average parallelism within a basic block rarely exceeds 3 or 4.

Beyond RISC

- The presence of dependence indicates the potential for a hazard, but actual hazard and the length of any stalls is a property of the pipeline.

- In general, data dependence indicates:
  - The possibility of a hazard,
  - The order in which results must be calculated,
  - An upper bound on how much parallelism can be possibly exploited.

Beyond RISC

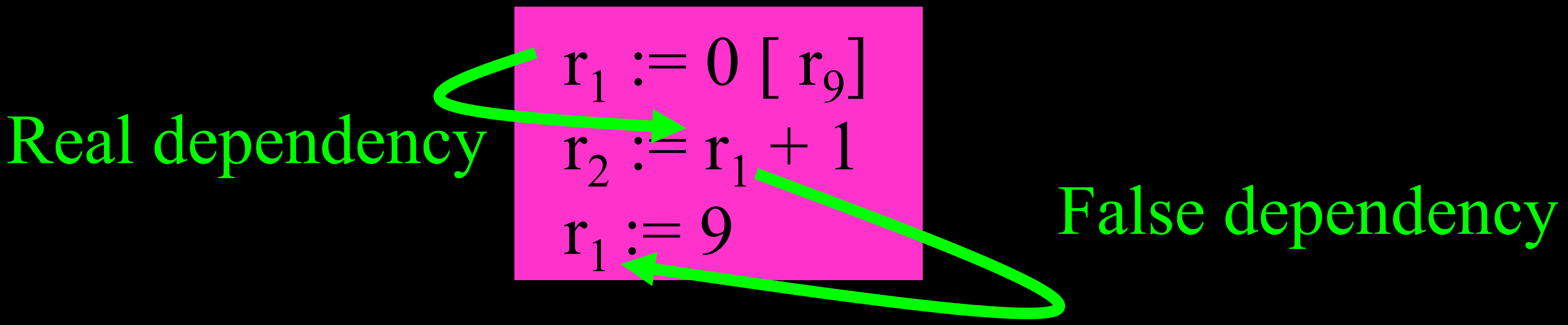
- Branches represent 20% of instructions in a program. Therefore, the length of a basic block is about 5 instructions.
- There is also a chance that some of the instructions in a basic building block are data dependent on each other.

Beyond RISC

- Therefore, to obtain substantial performance gains we must exploit ILP across multiple basic blocks.
- Simplest and most common way to increase parallelism is to exploit parallelism among loop iterations — loop level parallelism.

Beyond RISC

- Increasing parallelism within blocks
  - Parallelism within a basic block is limited by dependencies between instructions. Some of these dependencies are real, some are false:



Beyond RISC

- Increasing parallelism within blocks
  - Smart compiler might pay attention to its register allocation in order to overcome false dependencies.
  - Hardware register renaming is another alternative to overcome false dependencies.

Beyond *RISC*

Register Renaming

- Hardware renames the original register identifier in the instruction to correspond the new register with current value.
- Hardware that performs register renaming creates new register instance and destroys the instance when its value is superseded and there are not outstanding references to the value.
- To implement register renaming, the processor typically allocates a new register for every new value produced — the same register identifier in several different instructions may access different hardware registers.

Beyond *RISC*

- Register Renaming

  - $R_3 \Leftarrow R_3$ op R_5
  - $R_4 \Leftarrow R_3 + 1$
  - $R_3 \Leftarrow R_5 + 1$
  - $R_7 \Leftarrow R_3$ op R_4

$\Rightarrow$

$R_{3b} \Leftarrow R_{3a}$ op R_{5a}
$R_{4b} \Leftarrow R_{3b} + 1$
$R_{3c} \Leftarrow R_{5a} + 1$
$R_{7b} \Leftarrow R_{3c}$ op R_{4b}

Each assignment to a register creates a new instance of the register.

Beyond RISC

- Increasing parallelism Cross block boundaries
  - Branch prediction is often used to keep a pipeline full.
  - Fetch and decode instructions after a branch while executing the branch and the instructions before it — Must be able to execute instructions across an unknown branch speculatively.

Beyond RISC

- Increasing parallelism Cross block boundaries
  - Many architectures have several kinds of instructions that changes the flow of control:
    - Branches are conditional and have a destination some off set from the program counter.
    - Jumps are unconditional and may be either direct or indirect:
      - A direct jump has a destination explicitly defined in the instruction,
      - An indirect jump has a destination which is the result of some computation on registers.

Beyond RISC

- Increasing parallelism Cross block boundaries
  - Loop unrolling is a compiler optimization technique which allows us to reduce the number of iterations — Removing a large portion of branches and creating larger blocks that could hold parallelism unavailable because of the branches.

Beyond RISC

- Assume the following program:

LOOP:

LD	F_0, 0(R_1)	Load vector element into F_0
ADD	F_4, F_0, F_2	Add Scalar (F_2)
SD	F_4, 0(R_1)	Store the vector element
SUB	R_1, R_1, #8	Decrement by 8 (size of a double word)
BNZ	R_1, Loop	Branch if not zero

Beyond RISC

- Instruction cycles for a super scalar machine
  - Assume a super scalar machine that issues two instructions per cycle, one integer (Load, Store, branch, or integer), and one floating point:

IF	ID	EX	MEM	WB		
IF	ID	EX	MEM	WB		
	IF	ID	EX	MEM	WB	
	IF	ID	EX	MEM	WB	
		IF	ID	EX	MEM	WB
		IF	ID	EX	MEM	WB

Beyond RISC

- We will unroll the loop to allow simultaneous execution of floating point and integer operations:

Integer Inst.		Fl. Point Inst.		Clock cycle
LD	F_0, $0(R_1)$			1
LD	F_6, $-8(R_1)$			2
LD	F_{10}, $-16(R_1)$	AD	F_4, F_0, F_2	3
LD	F_{14}, $-24(R_1)$	AD	F_8, F_6, F_2	4
LD	F_{18}, $-32(R_1)$	AD	F_{12}, F_{10}, F_2	5
SD	F_4, $0(R_1)$	AD	F_{16}, F_{14}, F_2	6

Beyond RISC

Integer Inst.		Fl. Point Inst.		Clock cycle
SD	F_8, -8(R_1)	AD	F_{20}, F_{18}, F_2	7
SD	F_{12}, -16(R_1)			8
SD	F_{16}, -24(R_1)			9
SD	F_{20}, -32(R_1)			10
SUB	R_1, R_1, #40			11
BNZ	R_1, Loop			12

Beyond RISC

- Summary
  - Instruction Level Parallelism
    - Dynamic approach
    - Static approach
  - How to improve ILP within a basic block
    - Compiler role
    - Register renaming
  - How to improve ILP cross block boundaries
    - Static approach (loop unrolling)
    - Dynamic approach

Beyond RISC

- Increasing parallelism Cross block boundaries
  - Software pipelining is a compiler technique that moves instructions across branches to increase parallelism — Moving instructions from one iteration to another.

Beyond RISC

- Increasing parallelism Cross block boundaries
  - Trace scheduling is also a compiler scheduling technique.
  - It uses a profile to find a trace (sequence of blocks that are executed often) and schedules the instructions of these blocks as a whole — Prediction of branch statically based on the profile (to cope with failure, code is inserted outside the sequence to correct the potential error).

Beyond RISC

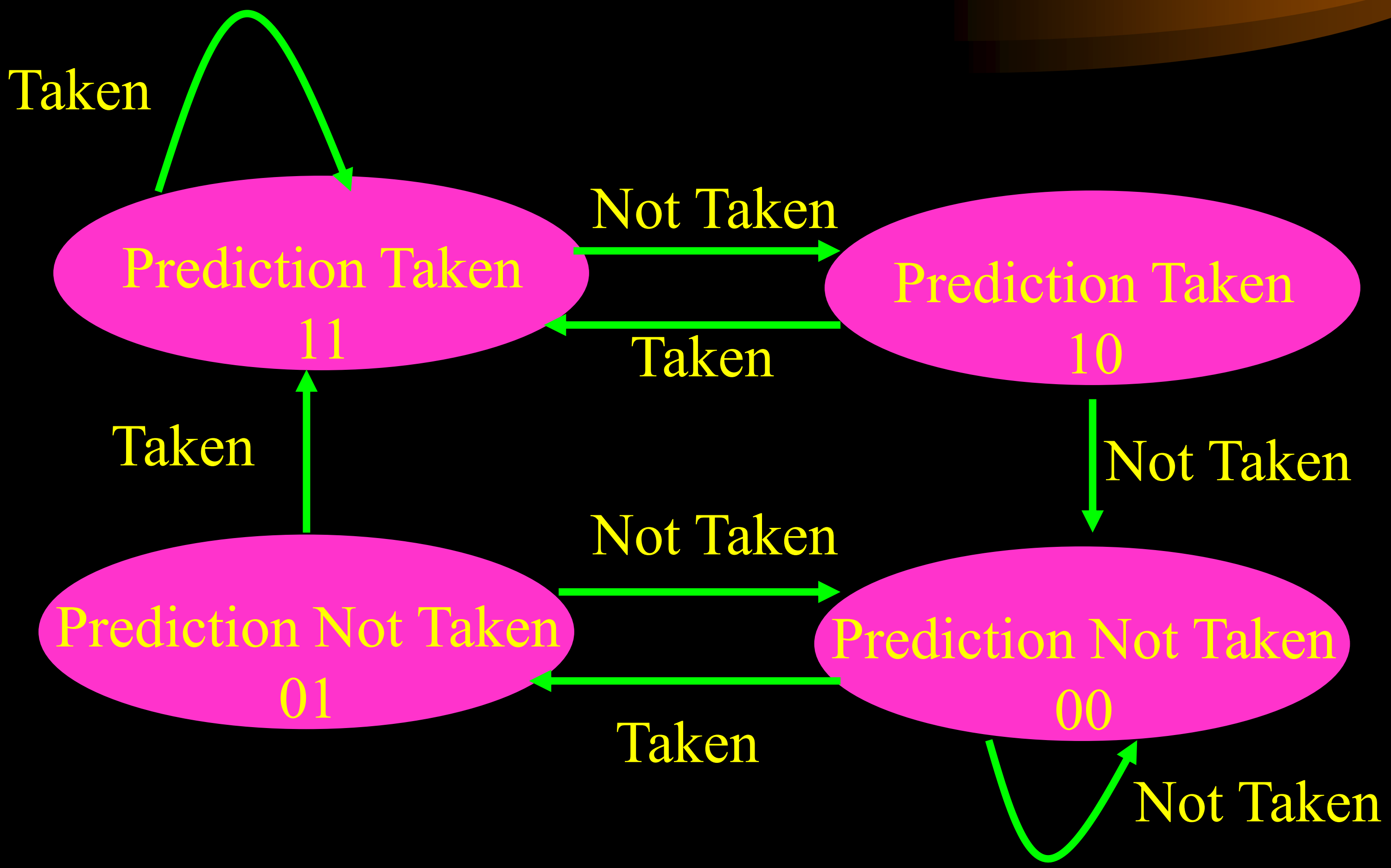
- Branch Prediction
  - Simplest way to have dynamic branch prediction is via the so called prediction buffer or branch history table ─ A table whose entries are indexed by lower portion of the target address.

Beyond RISC

- Branch Prediction
  - Entries in the branch history table can be interpreted as:
    - 1-bit prediction scheme: Each entry says whether or not in previous attempt branch was taken or not.
    - 2-bit Prediction scheme: Each entry is 2-bit long and a prediction must miss twice before it is changed — see the following diagram.

Beyond RISC

- Branch Prediction



Beyond RISC

- Branch Prediction
  - n-bit Saturation counter: An entry has a corresponding history feature. A taken branch increments the counter and untaken branch decrement the counter. A branch is not taken if the counter is below $2^{(n-1)}$.

Beyond RISC

- Branch Prediction
  - Multilevel prediction — Correlating predictors
    - A technique that uses behavior of other branches to make a prediction on a branch.
    - The first level is a table that shows the history of the branch. This may be the history (pattern) of the last *k* branches encountered (global behavior) or the last *k* occurrences of the same branch.
    - The second level shows the branch behavior for this pattern

Beyond RISC

- When instructions are issued in-order and complete in-order, there is one-to-one correspondence between storage locations (registers) and values.
- When instructions are issued out-of-order and complete out-of-order, the correspondence between register and value breaks down. This is even more severe when compiler optimizer does register allocation — tries to use as few registers as possible.

Beyond RISC

- Instruction Issue and Machine Parallelism
  - Instruction Issue is referred to the process of initiating instruction execution in the processor's functional units.
  - Instruction Issue Policy is referred to the protocol used to issue instructions.

Beyond RISC

- Instruction Issue Policy
  - In-order issue with in-order completion.
  - In-order issue with out-of-order completion.
  - Out-of-order issue with out-of-order completion.

Beyond RISC

- Instruction Issue Policy — Assume the following configuration:
  - Underlying Computer contains an instruction pipeline with three functional units.
  - Application Program has six instruction with the following dependencies among them:
    - I_1 requires two cycles to complete,
    - I_3 and I_4 conflict for a functional unit,
    - I_5 is data dependent on I_4, and
    - I_5 and I_6 conflict over a functional unit.

Beyond RISC

- In-order issue with in-order completion

Cycle	ID		EX			WB	
1	I_1	I_2					
2	I_3	I_4	I_1	I_2			
3	I_3	I_4	I_1				
4		I_4			I_3	I_1	I_2
5	I_5	I_6			I_4		
6		I_6		I_5		I_3	I_4
7				I_6			
8						I_5	I_6

- This policy is easy to implement, however, it generates long latencies that hardly justify its simplicity.

Beyond RISC

- Summary
  - How to improve ILP cross block boundaries
    - Static approach
      - loop unrolling,
      - software pipelining,
      - trace scheduling)
    - Dynamic approach
      - Branch prediction
  - Classification based on order of issue/order of completion
    - In order issue/in order completion
  - Project,
  - Homework #8

Beyond RISC

- In a simple pipeline structure, both structural and data hazards could be checked during instruction decode — When an instruction could execute without hazard, it will be issued from instruction decode stage (ID).

Beyond RISC

- To improve the performance, then we should allow an instruction to begin execution as soon as its data operands are available.

- This implies out-of-order execution which results in out-of-order completion.

Beyond RISC

- To allow out-of-order execution, then we split instruction decode stage into two stages:
  - Issue Stage to decode instruction and check for structural hazards,
  - Read Operand Stage to wait until no data hazards exist, then fetch operands.

Beyond RISC

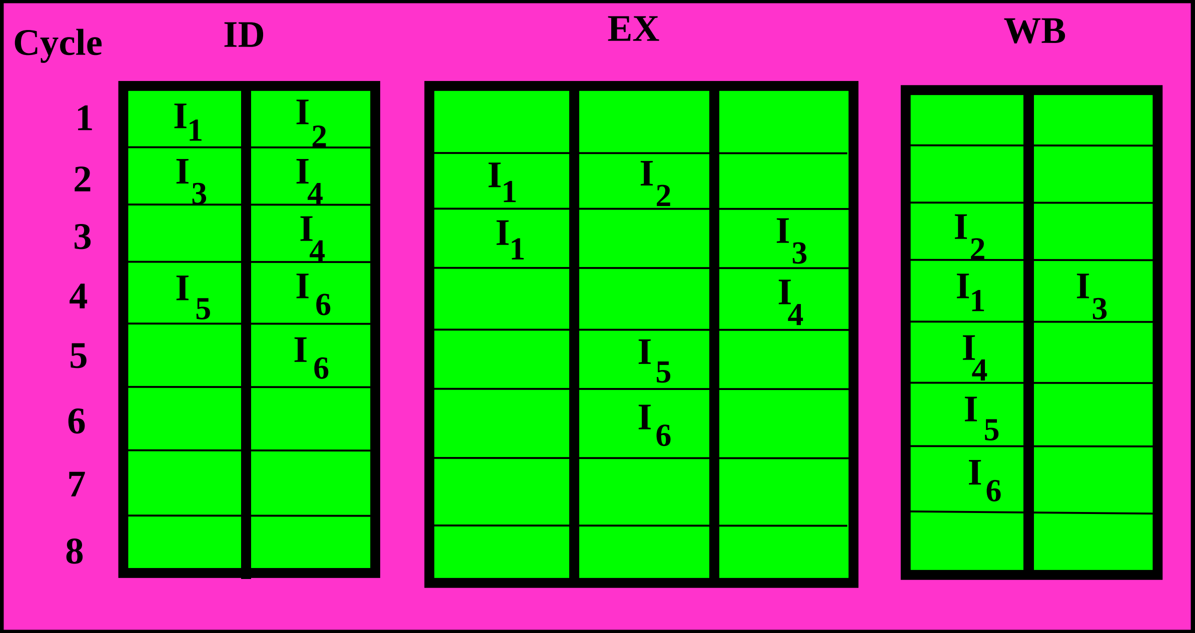
- Dynamic scheduling
  - Hardware rearranges the instruction execution order to reduce the stalls while maintaining data flow and exception behavior.
- Earlier approaches to exploit dynamic parallelism can be traced back to the design of CDC6600 and IBM 360/91.

Beyond RISC

- In a dynamically scheduled pipeline, all instructions pass through the issue stage in order, however, they can be stalled or bypass each other in the second stage and hence enter execution out of order.

Beyond RISC

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



Cycle	ID		EX			WB	
1	I_1	I_2					
2	I_3	I_4	I_1	I_2			
3		I_4	I_1		I_3	I_2	
4	I_5	I_6			I_4	I_1	I_3
5		I_6		I_5		I_4	
6				I_6		I_5	
7						I_6	
8							

Beyond RISC

- In-Order Issue with Out-of-Order Completion
  - Instruction issue is stalled when there is a conflict for a functional unit, or when an issued instruction depends on a result that is yet to be generated (flow dependency), or when there is an output dependency.
  - Out-of-Order completion yields a higher performance than in-order-completion.

Beyond RISC

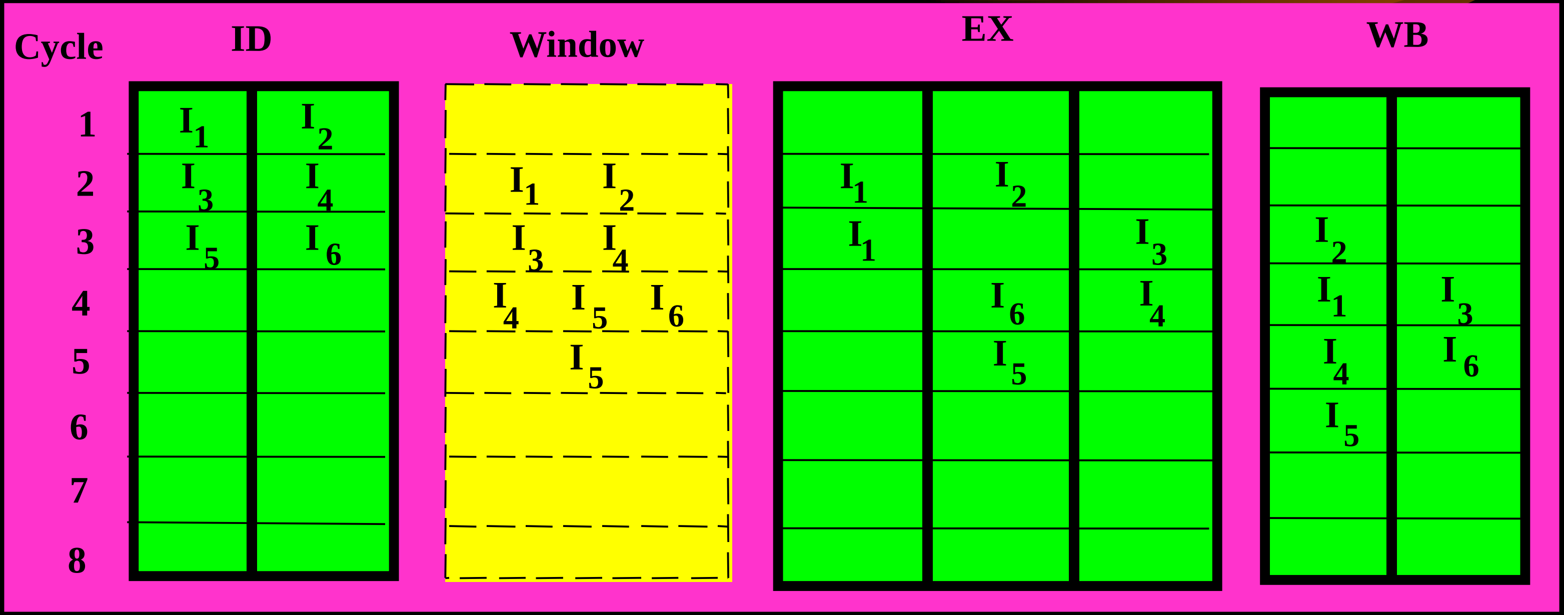
- Out-of-Order Issue with Out-of-Order Completion
  - The decoder is isolated (decoupled) from the execution stage, so that it continues to decode instructions regardless of whether they can be executed immediately.
  - This isolation is accomplished by a buffer between the decoder and execute stages — instruction window.
  - The fact that an instruction is in the window only implies that the processor has sufficient information about the instruction to know whether or not it can be issued.

Beyond RISC

- Out-of-Order Issue with Out-of-Order Completion
  - Out-of-Order issue gives the processor a larger set of instructions available to issue, improving its chances of finding instructions to execute concurrently.

Beyond RISC

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



Cycle	ID	ID	Window	EX	EX	EX	WB	WB
1	I_1	I_2						
2	I_3	I_4	I_1 I_2	I_1	I_2			
3	I_5	I_6	I_3 I_4	I_1		I_3	I_2	
4			I_4 I_5 I_6		I_6	I_4	I_1	I_3
5			I_5		I_5		I_4	I_6
6							I_5	
7								
8								

➢ Out-of-Order issue creates additional problem known as anti-dependency that needs to be taken care of.

Beyond RISC

- Summary
  - Classification based on order of issue/order of completion
    - In order issue/in order completion
    - In order issue/out of order completion
    - Out of order issue/out of order completion
  - Project, Dec 4
  - Final, Dec

Beyond RISC

- Machine with higher clock rate and deeper pipelines have been called super pipelined.
- Machines that allow to issue multiple instructions (say 2-3) on every clock cycles are called super scalar.
- Machines that pack several operations (say 5-7) into a long instruction word are called Very-long-Instruction-Word machines.

Very Long Instruction Word — VLIW

- Very Long Instruction Word (VLIW) design takes advantage of instruction parallelism to reduce number of instructions by packing several independent instructions into a very long instruction.
- Naturally, the more densely the operations can be compacted, the better the performance (lower number of long instructions).

Very Long Instruction Word — VLIW

- During compaction, NOOPs can be used for operations that can not be used.
- To compact instructions, software must be able to detect independent operations.

Very Long Instruction Word — VLIW

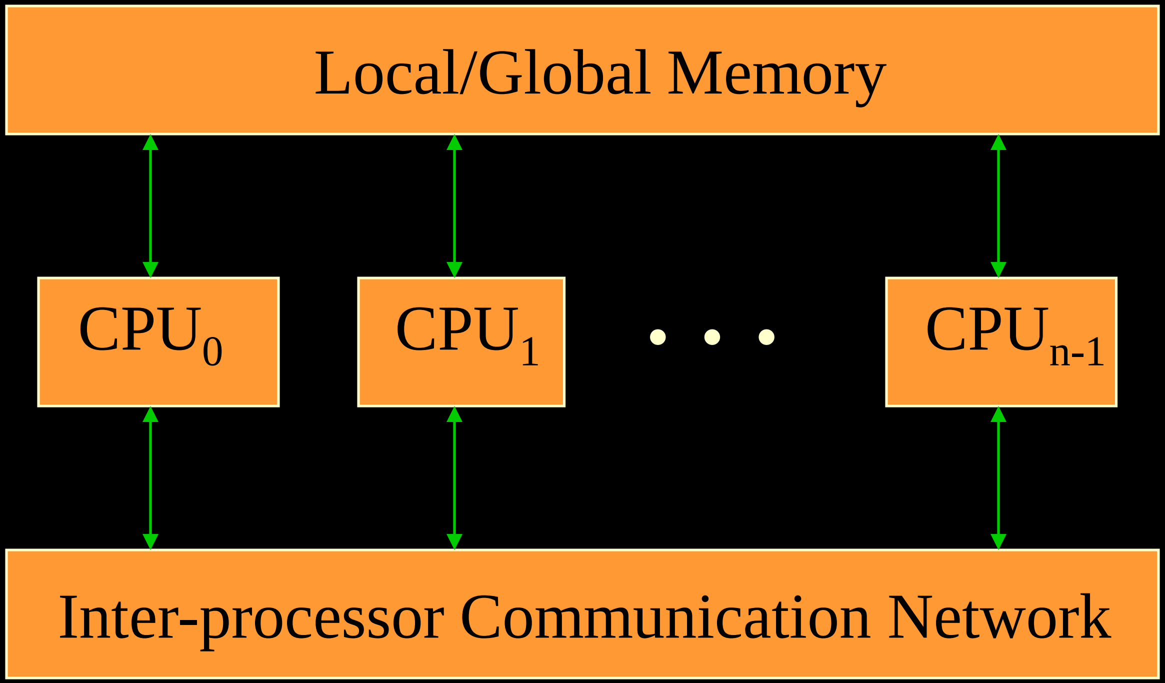
- The principle behind VLIW is similar to that of concurrent computing — execute multiple operations in one clock cycle.
- VLIW arranges all executable operations in one word simultaneously — many statically scheduled, tightly coupled, fine-grained operations execute in parallel within a single instruction stream.

Very Long Instruction Word — VLIW

- A VLIW instruction might include two integer operations, two floating point operations, two memory reference operations, and a branch operation.
- The compacting compiler takes ordinary sequential code and compresses it into very long instruction words through unrolling loops and trace scheduling scheme.

Very Long Instruction Word — VLIW

- Block Diagram



Very Long Instruction Word — VLIW

- Assume the following FORTRAN code and its machine code:

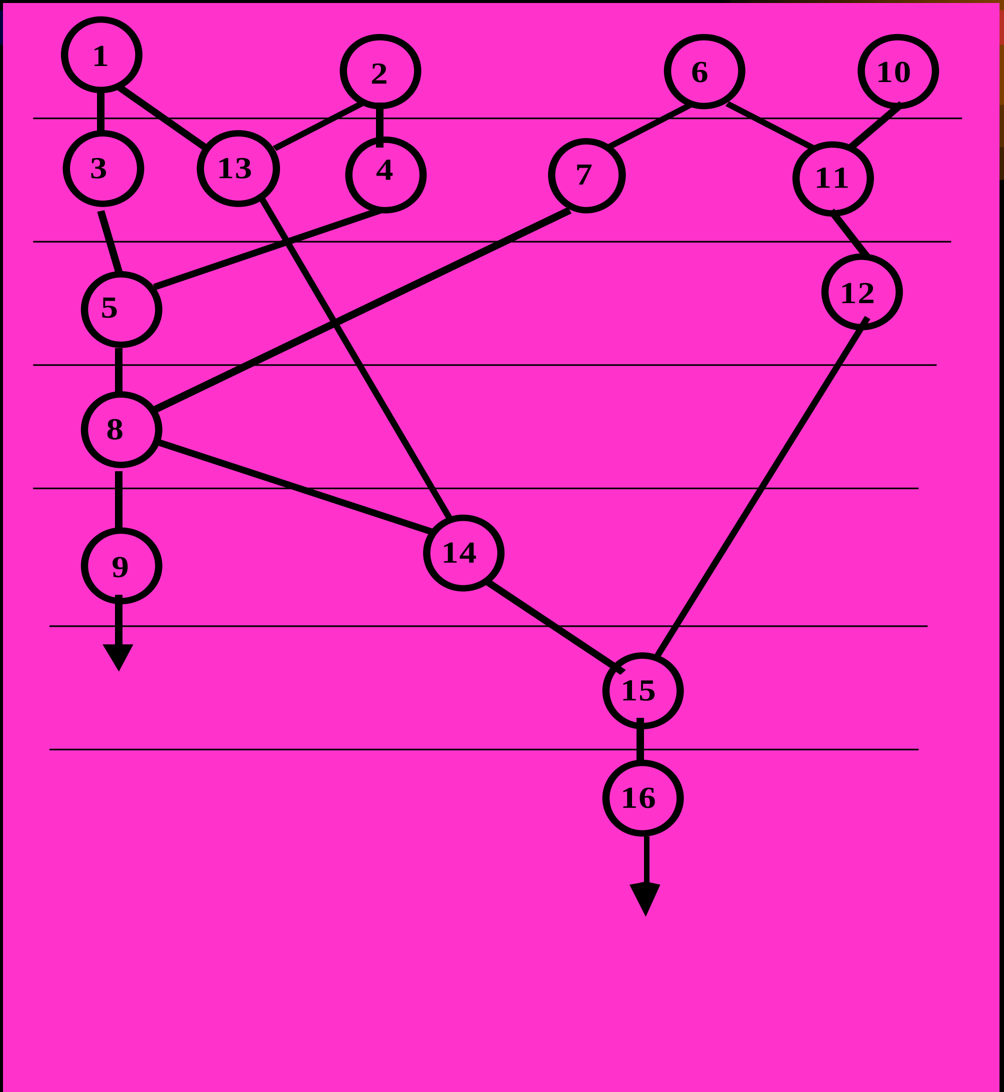
```
C = (A * 2 + B * 3) * 2 * i,
Q = (C + A + B) - 4 * (i + j)
```

Very Long Instruction Word — VLIW

- Machine code:

1)	LD A	2)	LD B
3)	$t_1 = A * 2$	4)	$t_2 = B * 3$
5)	$t_3 = t_1 + t_2$	6)	LD I
7)	$t_4 = 2 * I$	8)	$C = t_4 * t_3$
9)	ST C	10)	LD J
11)	$t_5 = I + J$	12)	$t_6 = 4 * t_5$
13)	$t_7 = A + B$	14)	$t_8 = C + t_7$
15)	$Q = t_8 - t_6$	16)	ST Q

Very Long Instruction Word — VLIW

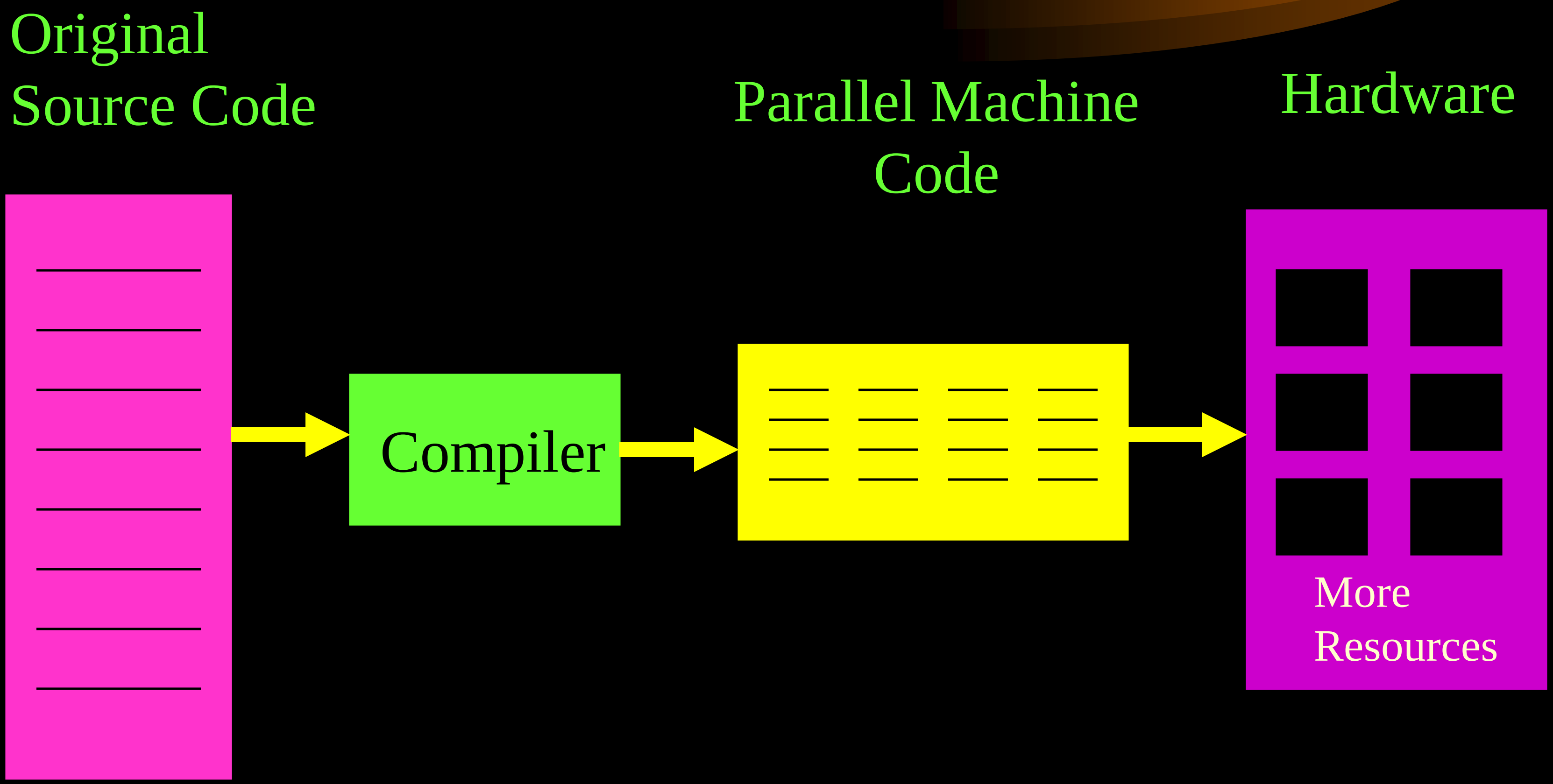


Very Long Instruction Word — VLIW

LD0	LD1	INT0	INT1	FP0	FP1	BRANCH
LD A	LD B					
LD I	LD J			A * 2	B * 3	
		2 * I	I + J	$t_1 + t_2$	A + B	
		4 * t_5		$t_4 - t_3$		
ST C					C + t_7	
				$t_8 – t_6$		
ST Q						

Very Long Instruction Word — VLIW

- Basic Principle of VLIW Architecture



Very Long Instruction Word — VLIW

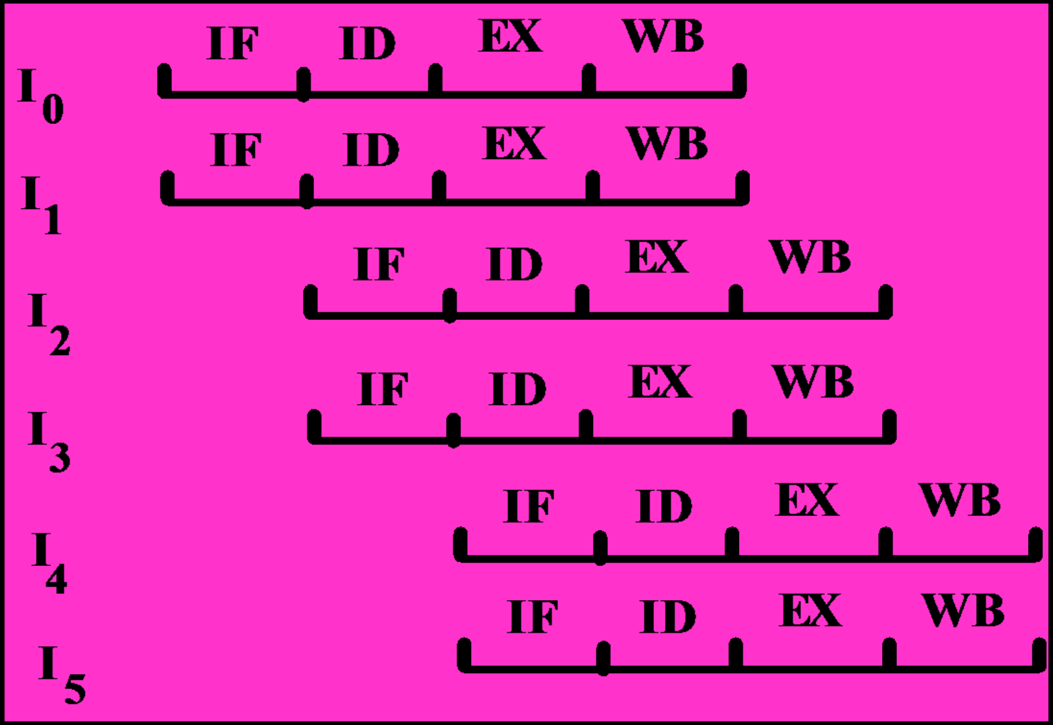
- Questions
  - Compare and contrast VLIW architecture against multiprocessor and vector processor (you need to discuss about issues such as — flow of control, inter-processor communications, memory organization and programming requirements).
  - Within the scope of VLIW architecture, discuss the major source of problems.

Super Scalar System

- A super scalar processor reduces the average number of clock cycles per instruction beyond what is possible in a pipeline scalar *RISC* processor. This is achieved by allowing concurrent execution of instructions in:
  - the same pipeline stages, as well as
  - different pipeline stages
- multiple concurrent operations on scalar quantities.

Super Scalar System

- Instruction Timing in a super scalar processor



Super Scalar System

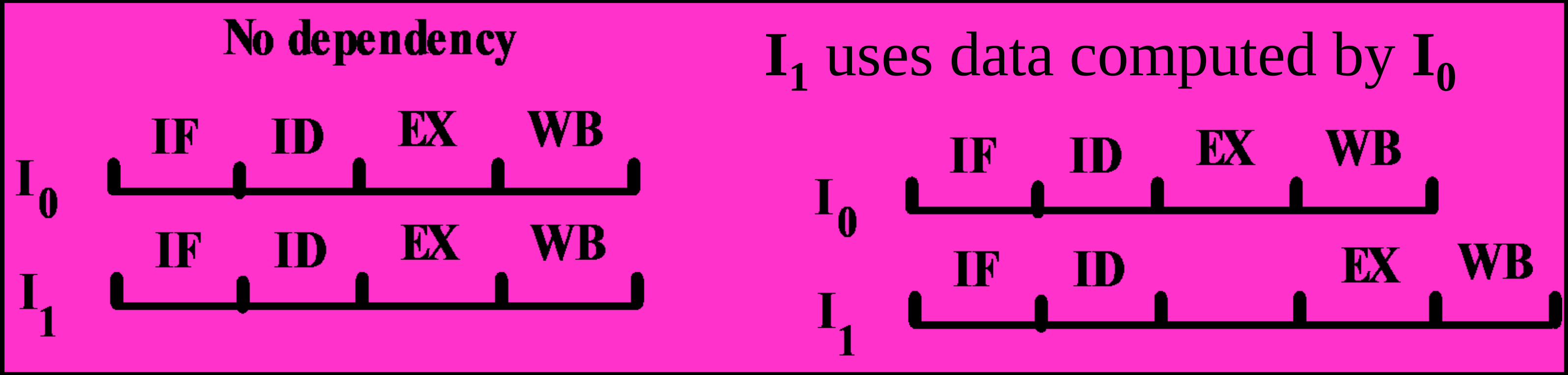
- Fundamental Limitations
  - Data Dependency
  - Control Dependency
  - Resource Dependency

Super Scalar System

- Data Dependency: If an instruction uses a value produced by a previous instruction, then the second instruction has a data dependency on the first instruction.
- Data dependency limits the performance of a scalar pipelined processor. The limitation of data dependency is even more severe in a super scalar than a scalar processor. In this case, even longer operational latencies degrade the effectiveness of super scalar processor drastically.

Super Scalar System

- Data dependency

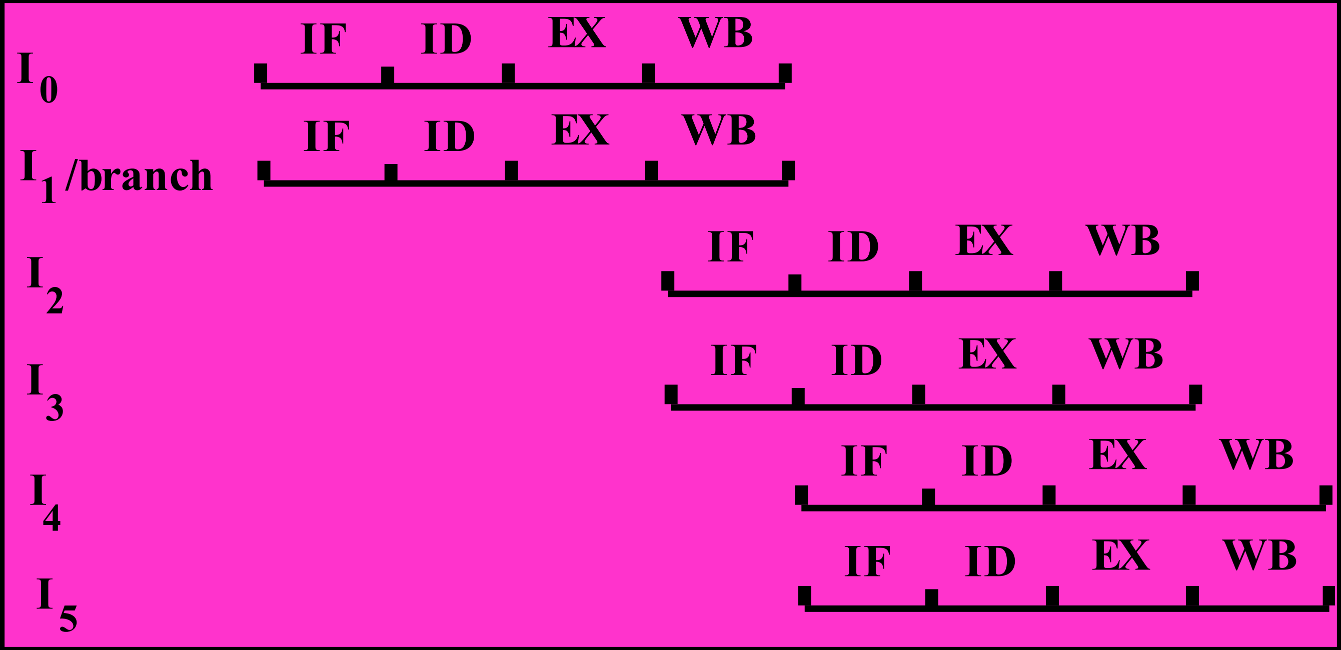


Super Scalar System

- Control Dependency
  - As in traditional *RISC* architecture, control dependency effects the performance of super scalar processors. However, in case of super scalar organization, performance degradation is even more severe, since, the control dependency prevents the execution of a potentially greater number of instructions.

Super Scalar System

- Control Dependency

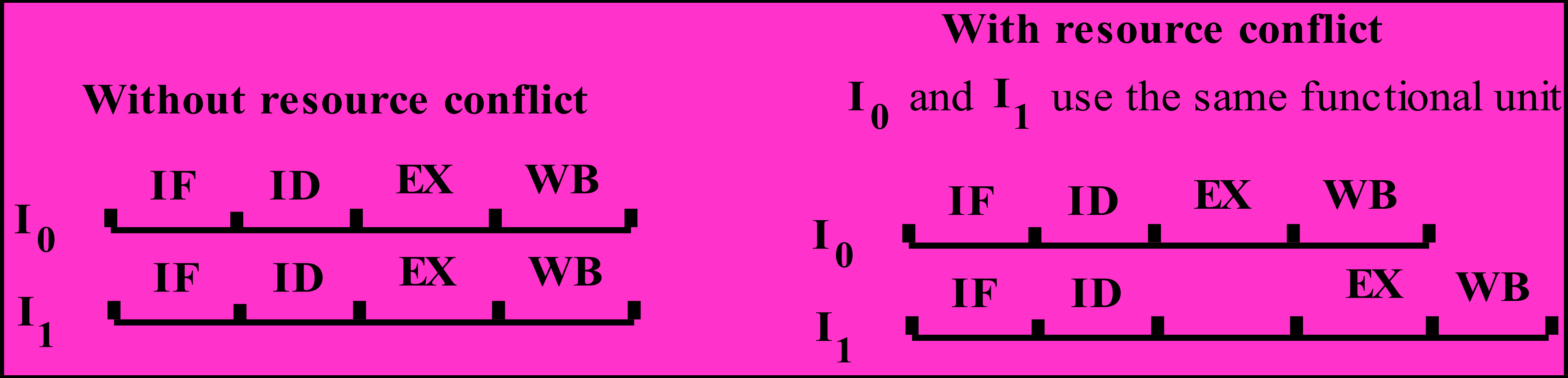


Super Scalar System

- Resource Dependency
  - A resource conflict arises when two instructions attempt to use the same resource at the same time. Resource conflict is also of concern in a scalar pipelined processor. However, a super scalar processor has a much larger number of potential resource conflicts.

Super Scalar System

- Resource Dependency



- Performance degradation due to the resource dependencies can be significantly improved by pipelining the functional units.

Super Scalar System

- Assume the following program:

LOOP:

LD	F_0, 0(R_1)	Load vector element into F_0
ADD	F_4, F_0, F_2	Add Scalar (F_2)
SD	F_4, 0(R_1)	Store the vector element
SUB	R_1, R_1, #8	Decrement by 8 (size of a double word)
BNZ	R_1, Loop	Branch if not zero

Super Scalar System

- Instruction cycles for a super scalar machine
  - Assume a super scalar machine that issues two instructions per cycle, one integer (Load, Store, branch, or integer), and one floating point:

```
IF   ID   EX   MEM   WB
IF   ID   EX   MEM   WB
     IF   ID   EX    MEM   WB
     IF   ID   EX    MEM   WB
          IF   ID    EX    MEM   WB
          IF   ID    EX    MEM   WB
```

Super Scalar System

- We will unroll the loop to allow simultaneous execution of floating point and integer operations:

Integer Inst.		Fl. Point Inst.		Clock cycle
LD	F_0, 0(R_1)			1
LD	F_6, -8(R_1)			2
LD	F_{10}, -16(R_1)	AD	F_4, F_0, F_2	3
LD	F_{14}, -24(R_1)	AD	F_8, F_6, F_2	4
LD	F_{18}, -32(R_1)	AD	F_{12}, F_{10}, F_2	5
SD	F_4, 0(R_1)	AD	F_{16}, F_{14}, F_2	6

Super Scalar System

Integer Inst.		Fl. Point Inst.		Clock cycle
SD	F_8, -8(R_1)	AD	F_{20}, F_{18}, F_2	7
SD	F_{12}, -16(R_1)			8
SD	F_{16}, -24(R_1)			9
SD	F_{20}, -32(R_1)			10
SUB	R_1, R_1, #40			11
BNZ	R_1, Loop			12

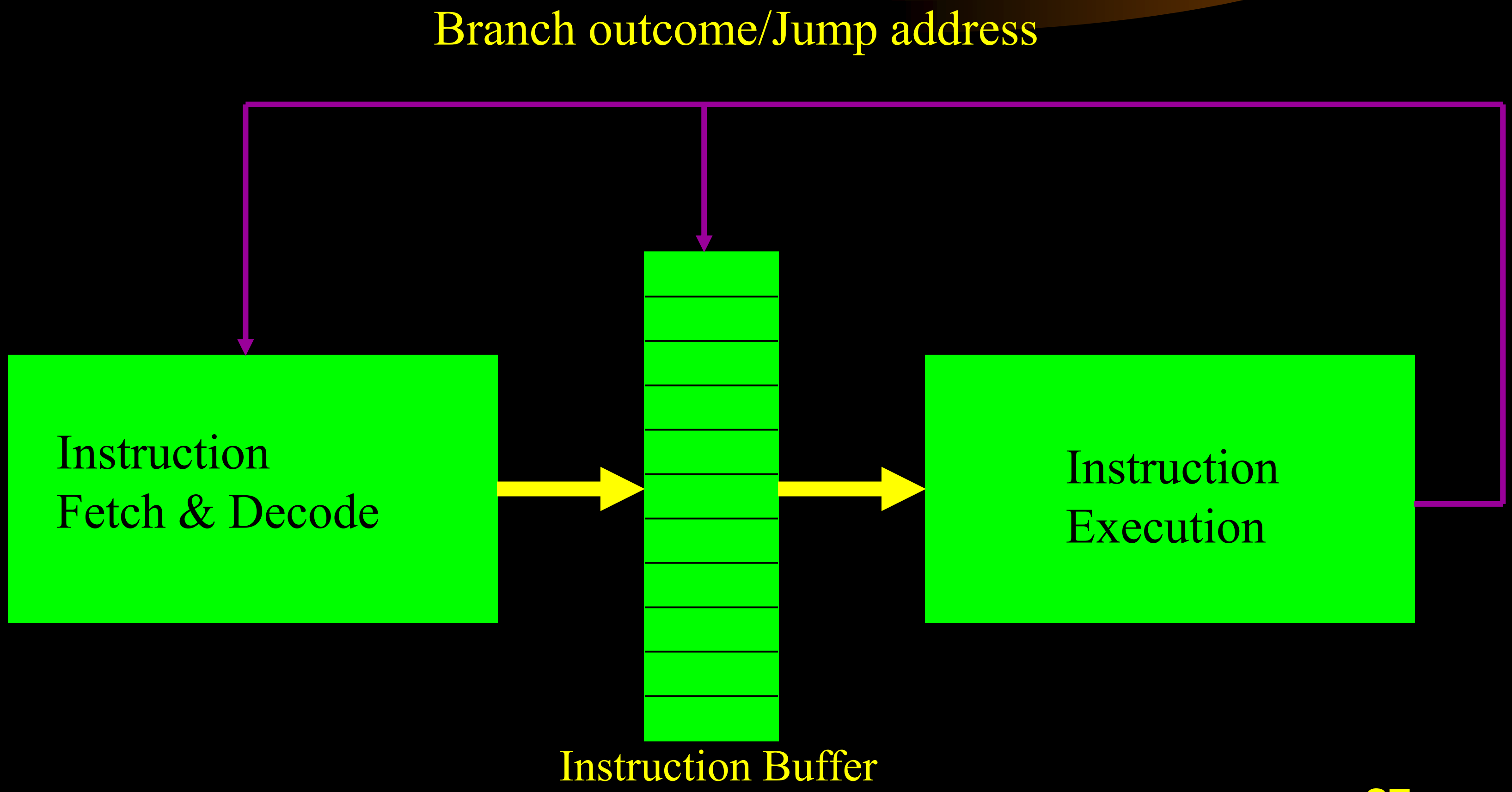
Super Scalar System

- As noted before, achieving a higher performance means processing a given task in a smaller amount of time. To reduce the time to execute a sequence of instructions, one can:
  - Reduce individual instruction latencies, or
  - Execute more instructions concurrently.

- Superscalar processors exploit the second alternative.

Super Scalar System

- General Configuration



Super Scalar System

- General Configuration
  - Instruction fetch unit acts as a producer, which fetches, decodes, and places decoded instructions into the buffer.
  - Instruction execution engine is the consumer, which removes instructions from buffer and executes them, subject to data dependence and resource constraints.
  - Control dependences provides a feedback mechanism between the producer and consumer.

Super Scalar System

- General Configuration
  - Systems having this organization employ aggressive techniques to exploit instruction level parallelism.

Super Scalar System

- General Configuration
  - Wide dispatch and issue paths,
  - Large issue buffer,
  - Large pool of physical registers,
  - Large number of parallel functional units,
  - Speculation of past multiple branches.

Fetch, decode, and issue several instructions

Register Renaming -- False Dependence

Resource Dependence

Control Dependence

Are some techniques that allow aggressive exploitation of Instruction Level Parallelism.

Super Scalar System

- Flow of Operations
  - A typical superscalar processor fetches and decodes several incoming instructions at a time.
  - The outcomes of conditional branch instructions are usually predicted in advance to ensure an uninterrupted stream of instructions
  - The incoming instructions are then analyzed for data and structural dependencies, and then independent instructions are distributed to functional units for execution.

Super Scalar System

- Flow of Operations
  - Simultaneously fetching several instructions, often predicting the outcomes of, and fetching beyond, conditional branch instructions,
  - Exploit dynamic parallelisms in the program:
    - Determine true dependencies involving register values and communicating these values to the target instructions during the course of execution,
    - Detect and remove false dependencies,
  - Initiate or issue multiple instructions in parallel,

Super Scalar System

- Flow of Operations
  - Manage resources for parallel execution of instructions, including:
    - Multiple pipeline functional units,
    - Memory hierarchy
  - Committing the process state in correct order.

Super Scalar System

- Flow of Operations
  - The key issue to the success of superscalar systems is the dynamic scheduling of the instructions in the program.

Super Scalar System

- Historical Perspective
  - The development of architectures to exploit instruction level parallelism in the form of pipelining can be traced back to the design of CDC6600 and IBM 360/91.
  - Within the scope of these systems, practice showed a pipeline initiation rate at one instruction per cycle.

Super Scalar System

- Processing Flow
  - An application is represented in a high level language program,
  - This high level program is then compiled into the static machine level program — The static program describes a set of executions and its implicit sequencing model (the order in which instructions are executed).

Super Scalar System

- Program Representation — High Level Construct

```
For 0 = i < last
        If a(i) > a(i+1)
                temp = a(i)
                a(i) = a(i+1)
                a(i+1) = temp
End
```

Super Scalar System

- Program Representation — Assembly code

```
L2:   Move   r3, r7          r7 points to an element of the array
      LW     r8, (r3)        r8 holds the ith element of the array
      Add    r3, r3, 4       advancing the index
      LW     r9, (r3)        r9 holds the i+1th element of the array
      Ble    r8, r9, L3

      Move   r3, r7          In this block ith and i+1th elements
      SW     r9, (r3)        are swapped
      Add    r3, r3, 4
      SW     r8, (r3)
      Add    r5, r5, 1

L3:   Add    r6, r6, 1       r6 holds the index
      Add    r7, r7, 4
      Blt    r6, r4, L2      r4 holds the "last"
```

Super Scalar System

- Processing Flow
  - During the course of execution, the sequence of executed instructions forms a dynamic instruction stream.
  - As long as instructions to be executed are sequential, static instruction sequencing can be entered into the dynamic instruction sequencing by incrementing the program counter.

Super Scalar System

- Processing Flow
  - However, in the presence of conditional branches and jumps the program counter must be updated to a nonconsecutive address — control dependence.
  - The first step in increasing instruction level parallelism is to overcome control dependencies.

Super Scalar System

- Control Dependencies — Straight line code
  - Let us talk about control dependencies due to the incrementing the program counter:
    - The static program can be viewed as a collection of basic blocks, each with a single entry point and a single exit point, refer to our example, we have three basic blocks.

Super Scalar System

◆ Control Dependencies — Straight line code

- Once a basic block is entered, its instructions are fetched and execute to completion, therefore, sequence of instructions in a basic block can be initiated into a conceptual window of execution.
- Once the instructions are initiated, they are free to execute in parallel, subject only to the data dependence constraints and availability of the hardware resources.

Super Scalar System

- Control Dependencies — Conditional Branch
  - To achieve a higher degree of parallelism, a super scalar processor should address updates of the program counter due to the conditional branches.
  - A method is to predict the outcome of a conditional branch and speculatively fetch and execute instructions from the predicted path.
  - Instructions from predicted path are entered into the window of execution.

Super Scalar System

- Control Dependencies — Conditional Branch
  - If prediction is later found to be correct, then the speculation status of the instructions are removed and their effect on the state of the system is the same as any other instructions.
  - If prediction is later found to be incorrect, the speculative execution was incorrect and recovery actions must be taken to undo the effect of incorrect actions.

Super Scalar System

- Processing Flow
  - In our running example, the *ble* instruction creates a control dependence.
  - To overcome this dependence, the branch could be predicted as not taken and hence, instructions between the branch and label L3 being executed speculatively.

Move	r_3, r_7
SW	$r_9, (r_3)$
Add	$r_3, r_3, 4$
SW	$r_8, (r_3)$
Add	$r_5, r_5, 1$

Super Scalar System

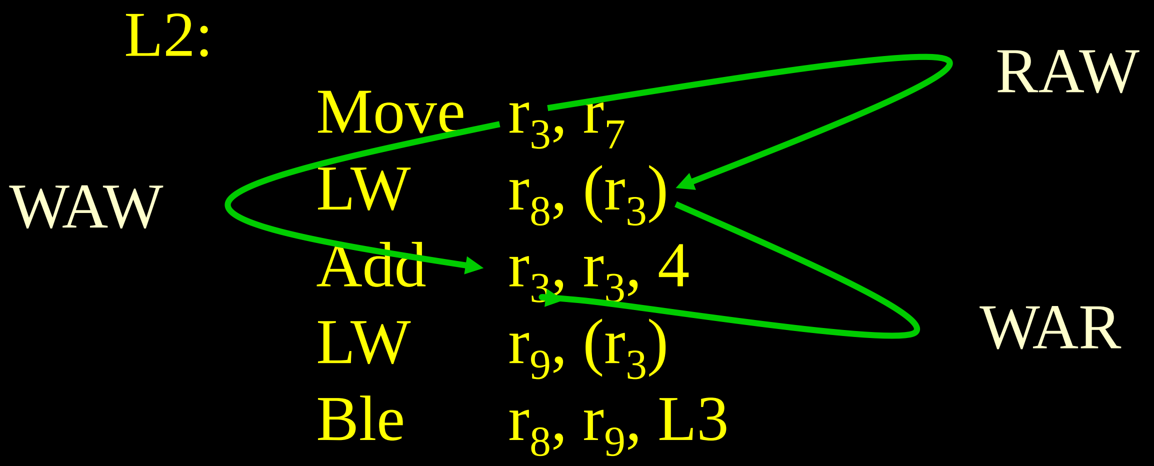
- Data Dependencies
  - Instructions placed in the window of execution may begin execution subject to data dependence constraints.
  - Note that data dependence comes in the form of:
    - Read After Write (RAW),
    - Write After Read (WAR), and
    - Write After Write (WAW).

Super Scalar System

- Data Dependencies
  - Note that, among the three aforementioned data dependence, RAW is the true dependence and the other two are false (artificial) data dependence.
  - In the process of execution, the false dependencies have to be overcome to increase degree of parallelism.

Super Scalar System

- Data Dependencies



L2:

Move	r_3, r_7
LW	r_8, (r_3)
Add	r_3, r_3, 4
LW	r_9, (r_3)
Ble	r_8, r_9, L3

Super Scalar System

- Processing Flow
  - After resolving control and artificial dependencies, instructions are issued and begin execution in parallel.
  - The hardware form a parallel execution schedule.
  - The execution schedule takes constraints such as true data dependence and hardware resource constraints into account.

Super Scalar System

- Processing Flow
  - A parallel execution schedule means that instructions complete in an order different than instructions order dictated by the sequential execution model.
  - Speculative execution means that some instructions may complete execution beyond the scope of the sequential execution model.

Super Scalar System

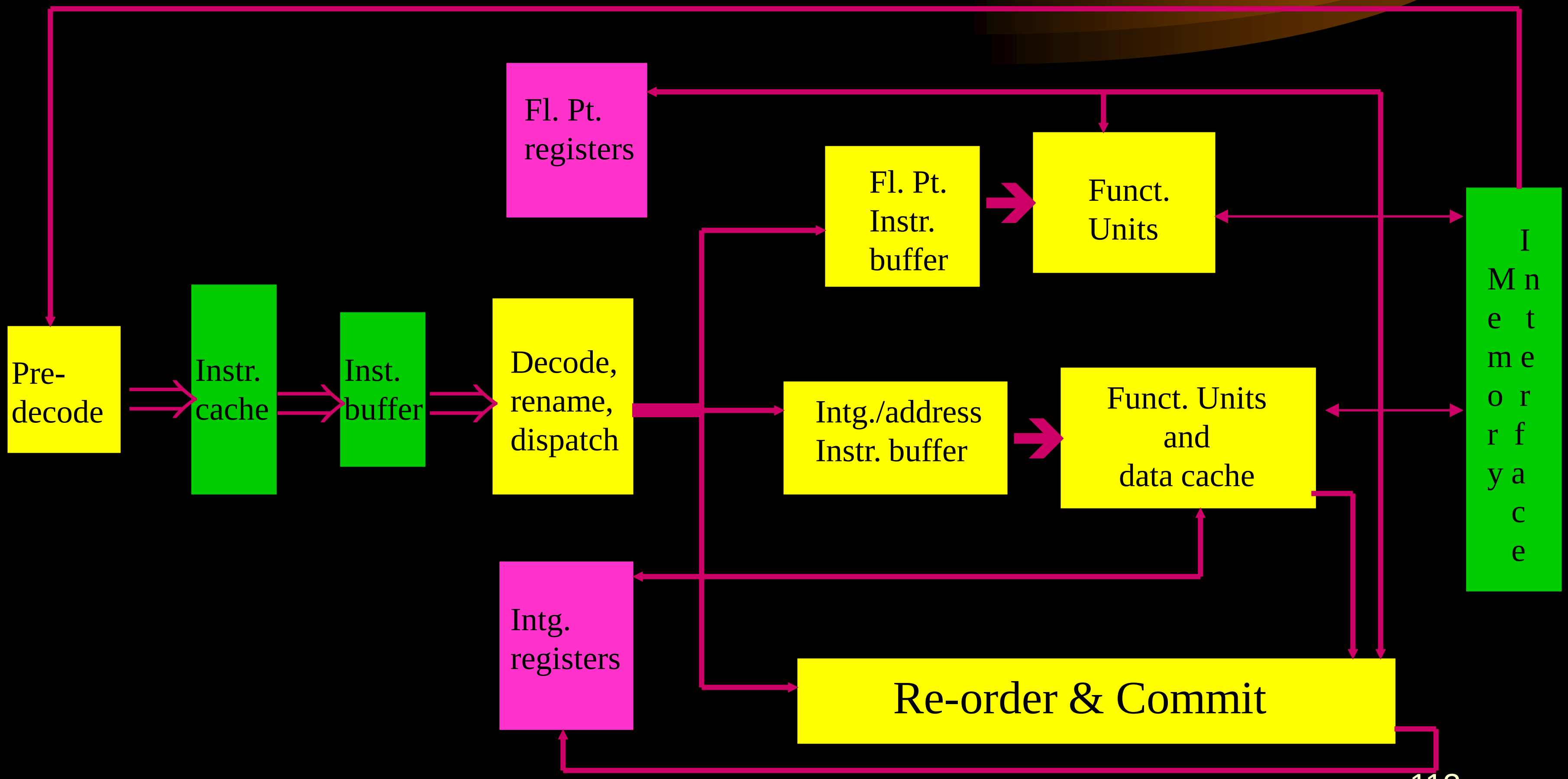
- Summary
  - Out-of-Order Issue, Out-of-Order Completion
  - Super Scalar processor
  - Dynamic exploitation of ILP
  - General Configuration of Super Scalar
  - Flow of Operations in a Super Scalar

Super Scalar System

- Processing Flow
  - Speculative execution implies that the execution results cannot be recorded permanently right away.
  - As a result, results of an instruction must be held in a temporary status until the architectural state can be updated.
  - Eventually, when it is determined that the sequential model would have executed an instruction, its temporary results are made permanent by updating the architectural state — Instruction is committed or retired.

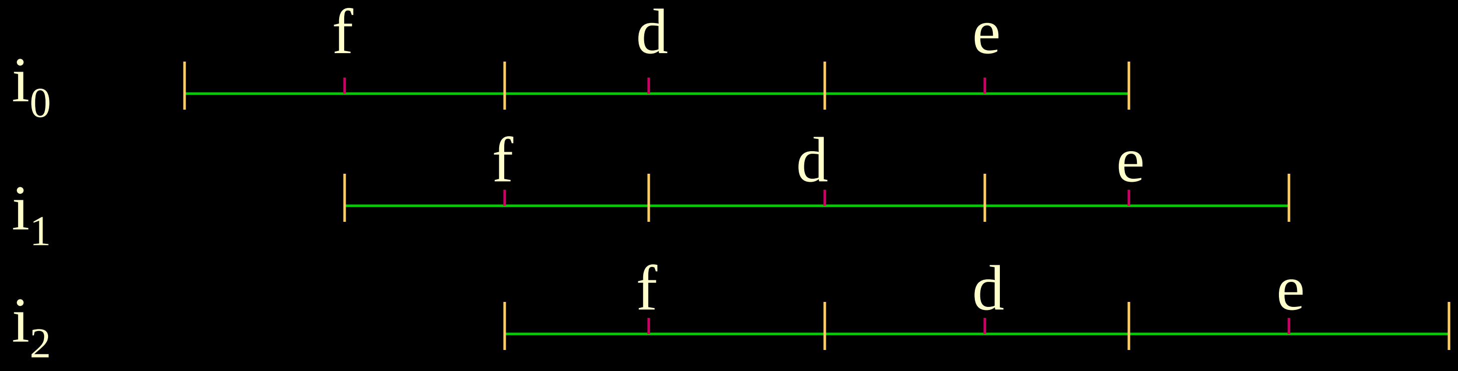
Super Scalar System

- Super Scalar Architecture



Super Pipelined Processor

- In a super Pipelined Processor, the major stages of a pipelined processor are divided into sub-stages.
- The degree of super pipelining is a measure of the number of sub-stages in a major pipeline stage.



2-Stage Super Pipelined Processor

Super Pipelined Processor

- Naturally, in a super Pipelined Processor, sub-stages are clocked at a higher frequency than the major stages.
- Reducing processor cycle time, hence higher performance, relies on instruction parallelism to prevent pipeline stalls in the sub-stages.

Super Pipelined Processor

- In comparison with Super Scalar:
  - For a given set of operations, the super pipelined processor takes longer to generate all results than the super scalar processor.
  - Simple operations take longer time to execute in a super scalar than super pipelined, since there are no clock with finer resolution.

Beyond RISC

- Summary
  - Scalar System
  - Super Scalar System
  - Super pipeline System
  - Very Long Instruction Word System
  - In-order-issue, In-order-Completion
  - In-order-issue, Out-of-order-Completion
  - Dynamic Scheduling
  - Out-of-order Issue, Out-of-order-Completion

Super Pipelined Processor

- From hardware point of view, super scalar processors are more susceptible to resource conflicts than super pipelined processor. As a result hardware should be duplicated for super scalar processor. On the other hand, in super pipelined processor, we need latches between pipeline sub-stages. This adds overhead to computation — degree of super pipelining could add severe overhead.

Intel Architecture

- Development of Intel Architecture (IA) can be traced back to the design of 8085 and 8080 microprocessors to the 4004 microprocessors (the first µprocessor designed by Intel in 1969).
- However, the 1st actual processor in the IA family is the 8086 model that quickly followed by 8088 architecture.

Intel Architecture

- The 8086 Characteristics:
  - 16-bit registers
  - 16-bit external data bus
  - 20-bit address space
- The 8088 is identical to the 8086 except it has a smaller external data bus (8 bits).

Intel Architecture

- The Intel 386 processor introduced 32-bit registers into the architecture. Its 32-bit address space was supported with an external 32-bit address bus.
- The instruction set was enhanced with new 32-bit operand and addressing modes with added new instructions, including the instructions for bit manipulation.
- Intel 386 introduces paging in the IA and hence support for virtual memory management.
- Intel 386 also allowed instruction pipelining of six stages.

Intel Architecture

- The Intel 486 processor added more parallelism by supporting deeper pipelining (instruction decode and execution units has 5 stages).
- 8-kByte on chip L_1 cache and floating point functional unit were added to the CPU chip.
- Energy saving mode and power management feature was added in the design of Intel 486 and Intel 386 as well (Intel 486SL and Intel 386SL).

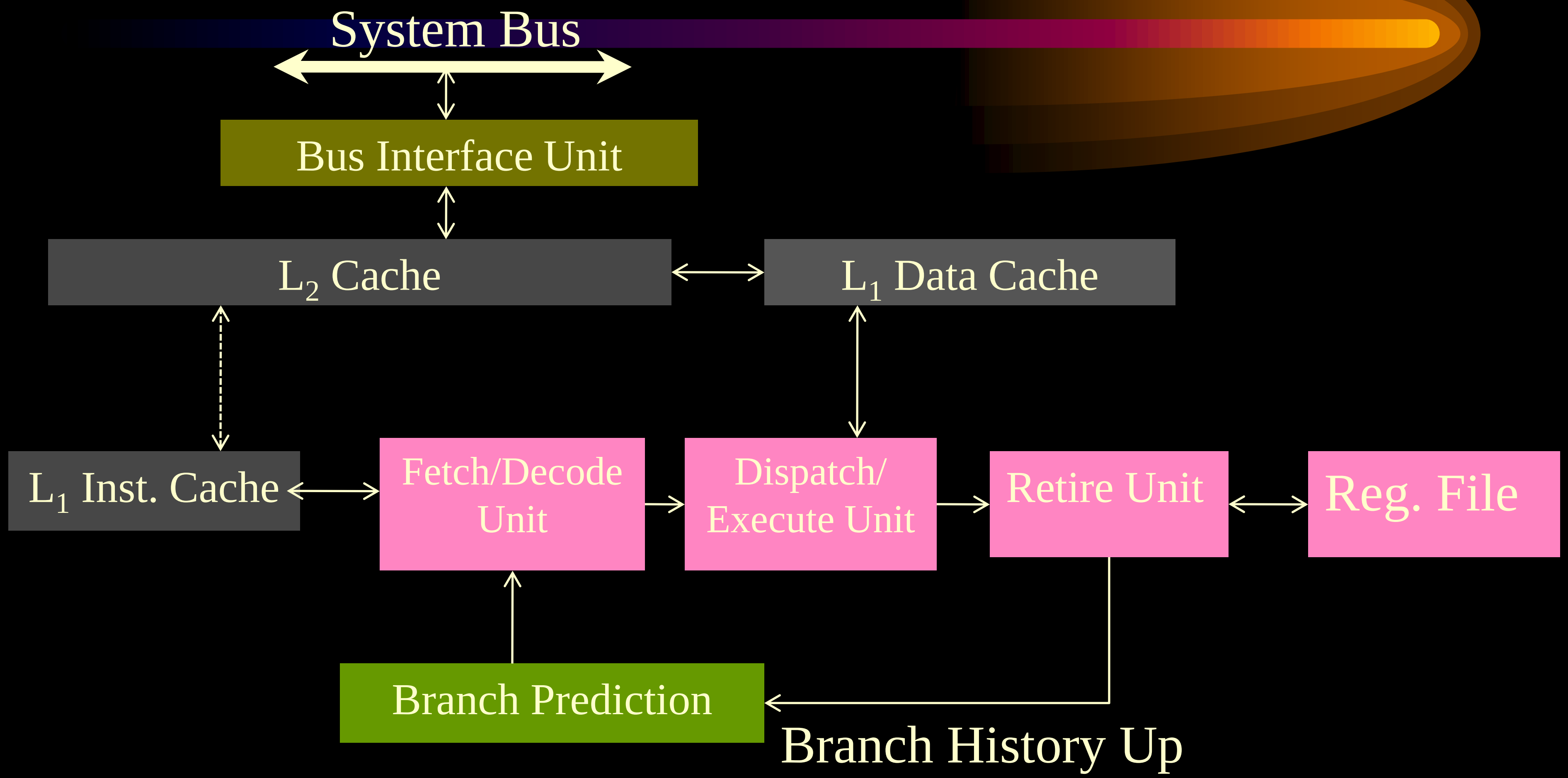
Intel Architecture

- Intel Pentium added a 2nd execution pipeline to achieve superscalar capability.
- On-chip dedicated L_1 caches were also added to its architecture (8 KBytes instruction and an 8 KBytes data caches).
- To support Branch prediction, the architecture was enhanced by an on-chip branch prediction table.
- The register size was 32 bits, however, internal data path of 128 bits and 256 bits have been added.
- Finally it has added features for dual processing.

Intel Architecture

- Intel Pentium Pro processor is a non-blocking, 3-way super scalar architecture that introduced "dynamic parallelism".
  - It allows micro dataflow analysis, out of order execution, superior branch prediction, and speculative execution.
  - It is consist of 5 parallel execution units (2 integer units, 2 floating point units, and 1 memory interface unit).
- Intel Pentium Pro has 2 on-chip 8 KBytes L_1 caches and one 256 KBytes L_2 on-chip cache using a 64-bit bus. L_1 cache is dual-ported and L_2 cache supports up to 4 concurrent accesses.
- Intel Pentium Pro supports 36-bit address space.
- Intel Pentium Pro uses a decoupled 12-stage instruction pipeline.

Intel Architecture



Intel Architecture

- Pentium II is an extension of Pantium Pro with added MMX instructions. L_2 cache is off-chip and of size 256 KBytes, 512 KBytes, 1 MBytes, or 2 MBytes. However, L_1 caches are extended to 16 kBytes.
- Pentium II uses multiple low power states (power management); Auto HALT, Stop-Grant, Sleep, and Deep Sleep.

Beyond RISC

- Pentium III is built based on Pentium Pro and Pentium II processors. It introduces 70 new instructions with a new SIMD floating point unit.

Intel Architecture

Intel Processor		Perform (MIPS)	Clock Frequency	# trans on Die	Register Size	External Data Bus Size	Addr. Space	Caches
8086	1978	0.8	8 MHz	29 K	16 bits	16 bits	1 MB	None
Intel 286	1982	2.7	12.5 MHz	134 K	16 bits	16 bits	16 MB	None
Intel 386	1985	6.0	20 MHz	275 K	32 bits	32 bits	4 GB	None
Intel 486	1989	20	25 MHz	1.2 M	32 bits	32 bits	4 GB	8KB L_1
Pentium	1993	100	60 MHz	3.1 M	32 bits	64 bits	4 GB	16KB L_1
Pentium Pro	1995	440	200 MHz	5.5 M	32 bits	64 bits	64 GB	16KB L_1, 256KB L_2 or 512KB L_2
Pentium II	1997	466	266 MHz	7 M	32 bits	64 bits	64 GB	32KB L_1, 256KB L_2 or 512KB L_2
Pentium III	1999	1000	500 MHz	8.2 M	32 (GP), 128 (FP)	64 bits	64 GB	32KB L_1, 512KB L_2

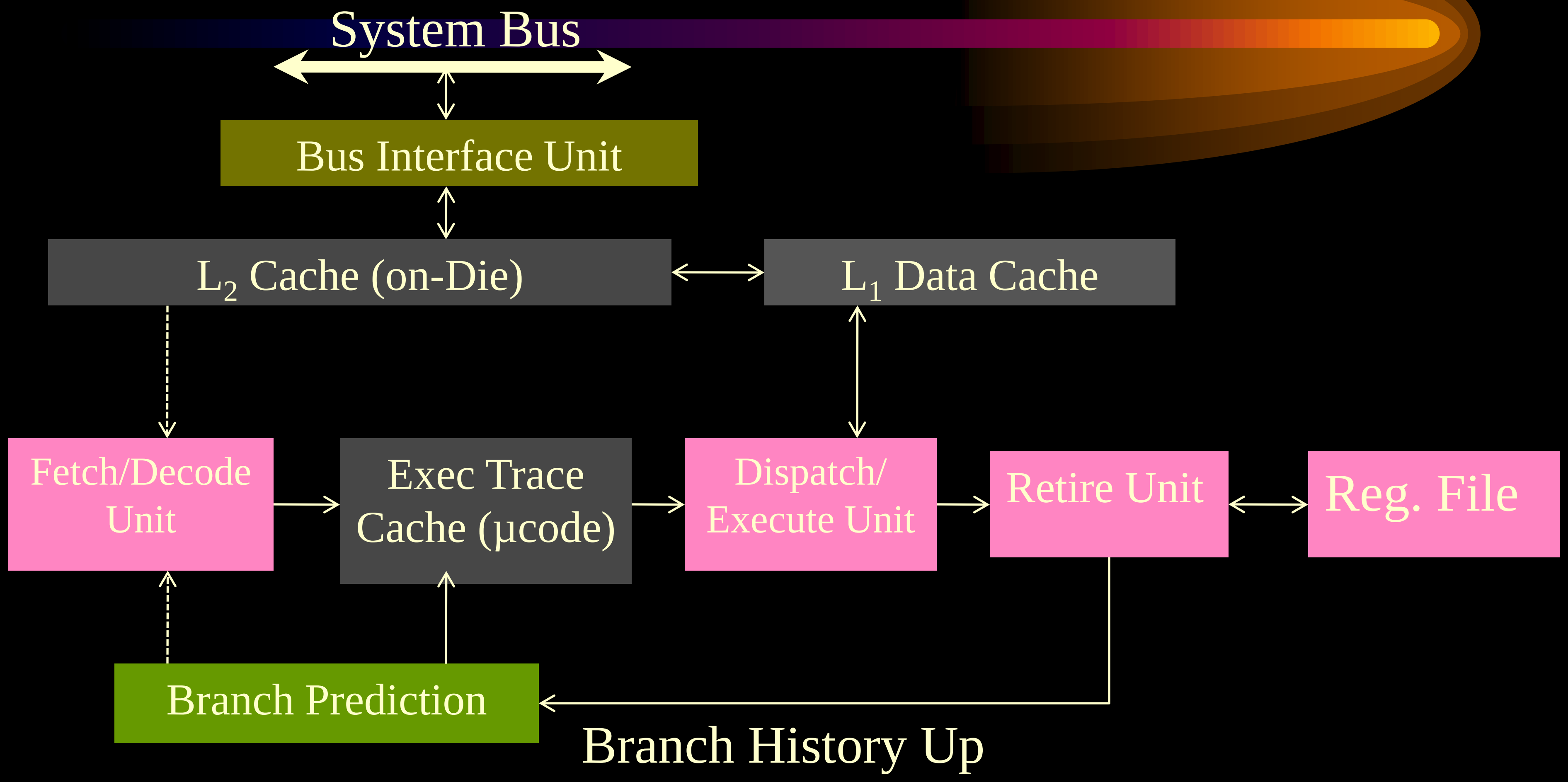
Intel Architecture

- Pentium 4 offers new features that allows higher performance in multimedia applications.
- The SSE2 extensions allow application programmers to control cacheability of data.
- Pentium 4 has 42 million transistors using 0.18µ CMOS technology.

Intel Architecture

Intel Processor		μ-arch.	Clock Freq.	Tran/Die	Reg. Size (bits)	Bus Bandwidth	Addr. Space	On-die Caches
Pentium III	1999	P6	700 MHz	28 (M)	GP: 32 FPU: 80 MMX: 64 XMM: 128	Up to 1.06 GB/s	64 GB	32KB L_1, 256KB L_2
Pentium 4	2000	NetBurst	1.50 GHz	42 (M)	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12Kμop Exec. Trace Cache; 8KB L_1, 256KB L_2

Intel Architecture



Intel Architecture

- First Level Caches:
  - Execution Trace Cache stores decoded instructions and removes decoder latency from main execution loops.
  - Low latency data cache has 2 cycle latency.
- Very deep (20-satge mis-prediction pipeline), out-of-order, speculative execution engine.
  - Up to 126 instructions in flight.
  - Up to 48 loads and 24 stores in pipeline.
  - Arithmetic Logic Units runs at twice the processor frequency (3GHz).
  - Basic integer operations executes ½ processor cycle time.

Intel Architecture

- Enhance branch prediction:
  - Reduce mis-prediction penalty
  - Advanced branch prediction algorithm
  - 4k-entry branch target array.
- Can retire up to three µoperations per clock cycle.

Wish you all the best