Beyond *RISC*

A.R. Hurson
Computer Science Department
Missouri Science & Technology
hurson@mst.edu

Beyond *RISC*

- Why did I ask some questions ab CDC6600 and 7600?

- Earlier notion of Super scalar processor v discussed.

- What is Instruction Level Parallelism?

Beyond *RISC*

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

Fall 2010

Beyond *RISC*

- Straight line code blocks are between f to seven instructions that are norm dependent on each other — degree parallelism within a code block is limited

- Several studies have shown that aver parallelism within a basic block ra exceeds 3 or 4.

Beyond *RISC*

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

Beyond *RISC*

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

Beyond *RISC*

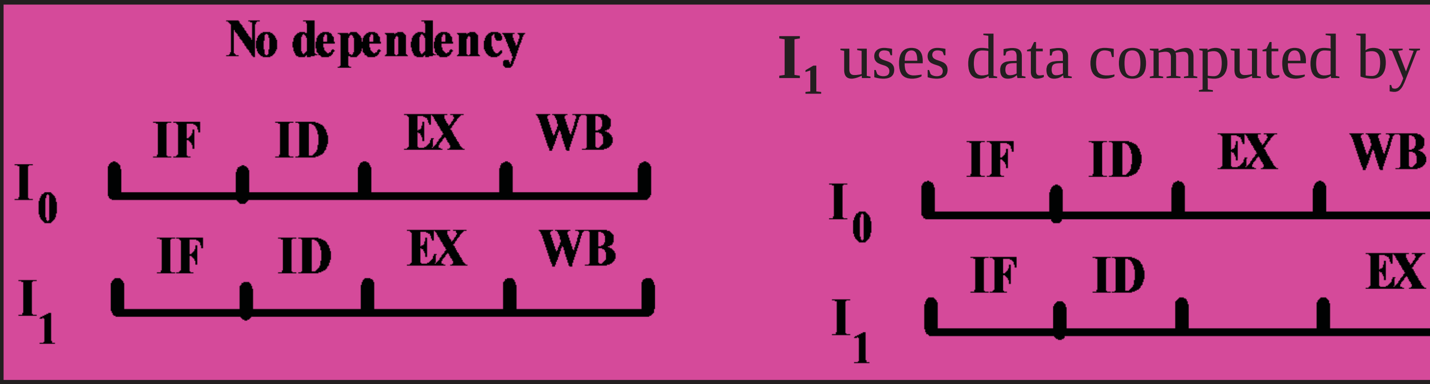
- Therefore, to obtain substan performance gains we must exploit I across multiple basic blocks.
- Simplest and most common way increase parallelism is to exp parallelism among loop iterations loop level parallelism.

Beyond *RISC*

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

Beyond *RISC*

- Data dependency



No dependency

I_1 uses data computed by

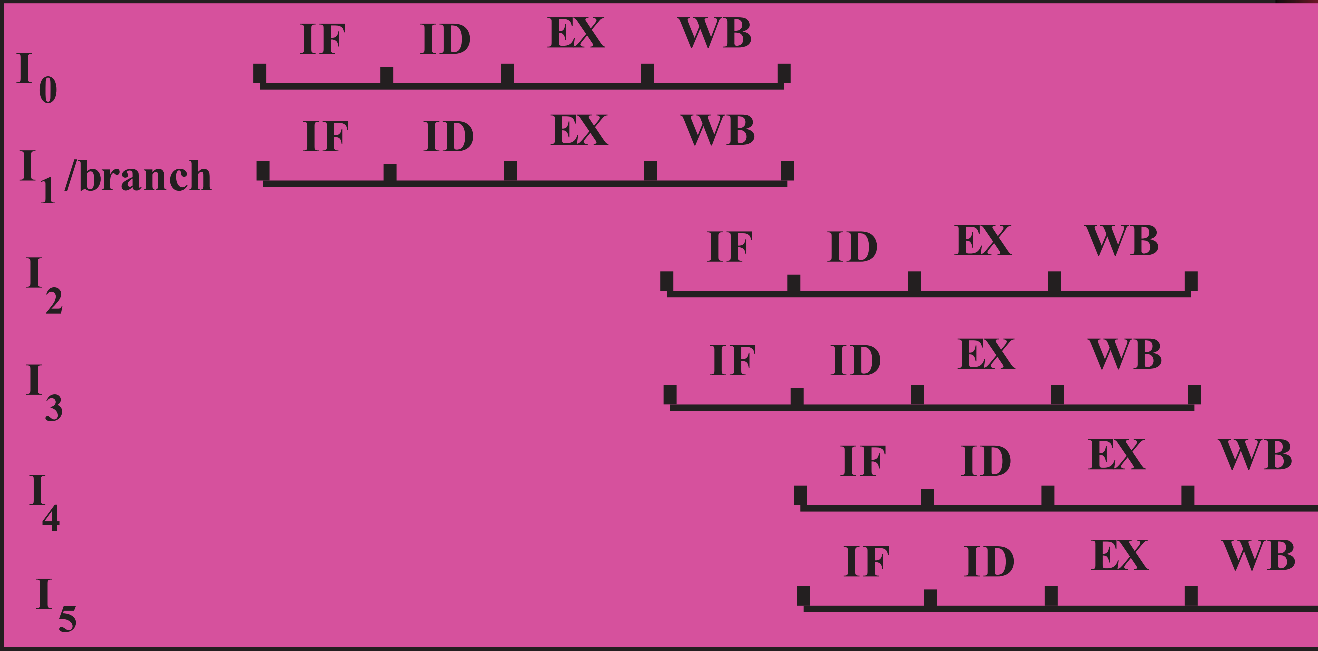
Fall 2010

Beyond *RISC*

- Control Dependence
  - As in traditional *RISC* architecture, con dependence effects the performance of su scalar processors.
  - However, in case of super scalar organizat performance degradation is even more sev since, the control dependence prevents execution of a potentially greater numbe instructions.

Beyond *RISC*

Control Dependency



I_0 IF ID EX WB

I_1 /branch IF ID EX WB

I_2 IF ID EX WB

I_3 IF ID EX WB

I_4 IF ID EX WB

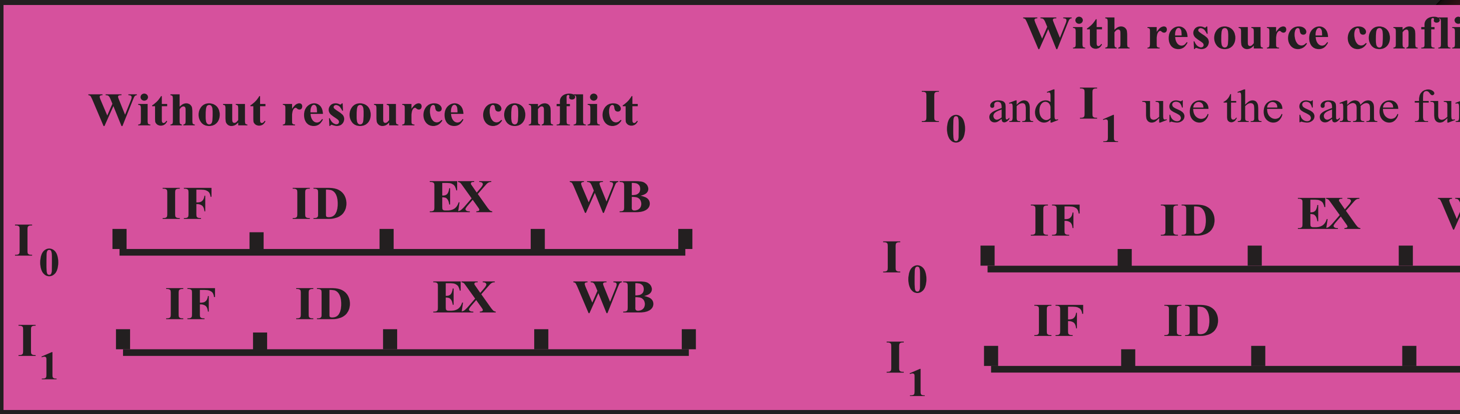
I_5 IF ID EX WB

Beyond *RISC*

- Resource Dependence
  - A resource conflict arises when two instruct attempt to use the same resource at the s time. Resource conflict is of concern in a sc pipelined processor.
  - A super scalar processor has a much la number of potential resource conflicts.

Beyond *RISC*

Resource Dependency



Without resource conflict

I_0 IF ID EX WB

I_1 IF ID EX WB

With resource confli

I_0 and I_1 use the same fu

I_0 IF ID EX V

I_1 IF ID

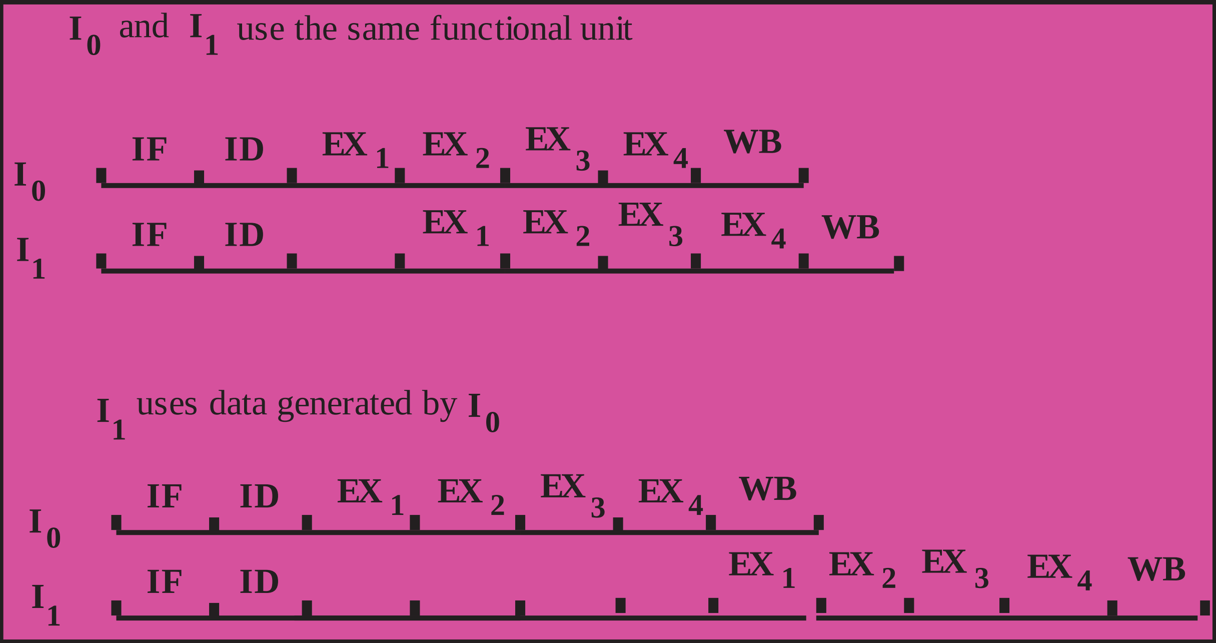
Performance degradation due to the resource depende can be significantly improved by pipelining the funct units.

Beyond *RISC*

- A quick look at our previous examples c imply that the data dependencies and reso dependencies have the same effect on instruc pipelining.
- Resource dependence can be resolved moderated by duplicating the hardware pipelining the hardware. However, this is not for the data dependence case.

Beyond *RISC*

- Resource and Data Dependencies — Assu
pipelined functional units:



I_0 and I_1 use the same functional unit

I_0 IF ID EX_1 EX_2 EX_3 EX_4 WB

I_1 IF ID EX_1 EX_2 EX_3 EX_4 WB

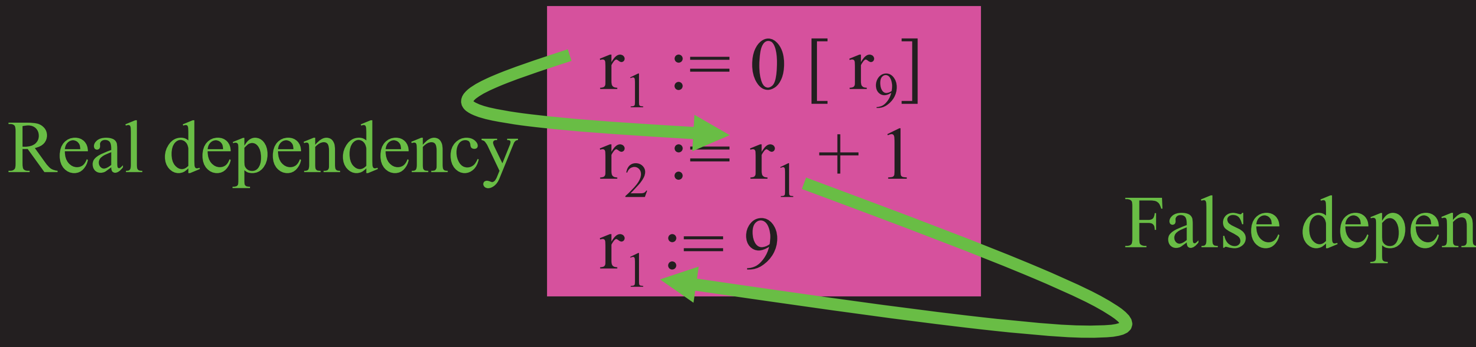
I_1 uses data generated by I_0

I_0 IF ID EX_1 EX_2 EX_3 EX_4 WB

I_1 IF ID EX_1 EX_2 EX_3 EX_4 WB

Beyond *RISC*

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



$r_1 := 0 [r_9]$

$r_2 := r_1 + 1$

$r_1 := 9$

Real dependency

False depen

Beyond *RISC*

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

Beyond *RISC*

Register Renaming

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

→

$R_{3b} \Leftarrow R_{3a}$ 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 o register.

Beyond *RISC*

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

Beyond *RISC*

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

Beyond *RISC*

- Increasing parallelism Cross block boundar
  - Loop unrolling is a compiler optimiza technique which allows us to reduce number of iterations – Removing a l portion of branches and creating larger blo that could hold parallelism unavailable beca of the branches.

Beyond *RISC*

- Assume the following program:

```
LOOP:
    LD      F0, 0(R1)       Load vector element into F0
    ADD     F4, F0, F2      Add Scalar (F2)
    SD      F4, 0(R1)       Store the vector element
    SUB     R1, R1, #8      Decrement by 8 (size of a double
    BNZ     R1, Loop        Branch if not zero
```

Beyond *RISC*

- Instruction cycles for a super scalar machine
  - Assume a super scalar machine that issues instructions per cycle, one integer (Load, St 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 simultane execution of floating point and integer operatio

Integer Inst.	Fl. Point Inst.	Cloc
LD F_0, $0(R_1)$		
LD F_6, $-8(R_1)$		
LD F_{10}, $-16(R_1)$	AD F_4, F_0, F_2	
LD F_{14}, $-24(R_1)$	AD F_8, F_6, F_2	
LD F_{18}, $-32(R_1)$	AD F_{12}, F_{10}, F_2	
SD F_4, $0(R_1)$	AD F_{16}, F_{14}, F_2	

Beyond *RISC*

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

Beyond *RISC*

- Increasing parallelism Cross block boundar
  - Software pipelining is a compiler technique moves instructions across branches to incr parallelism — Moving instructions from iteration to another.

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
    - Dynamic approach

Beyond *RISC*

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

Beyond *RISC*

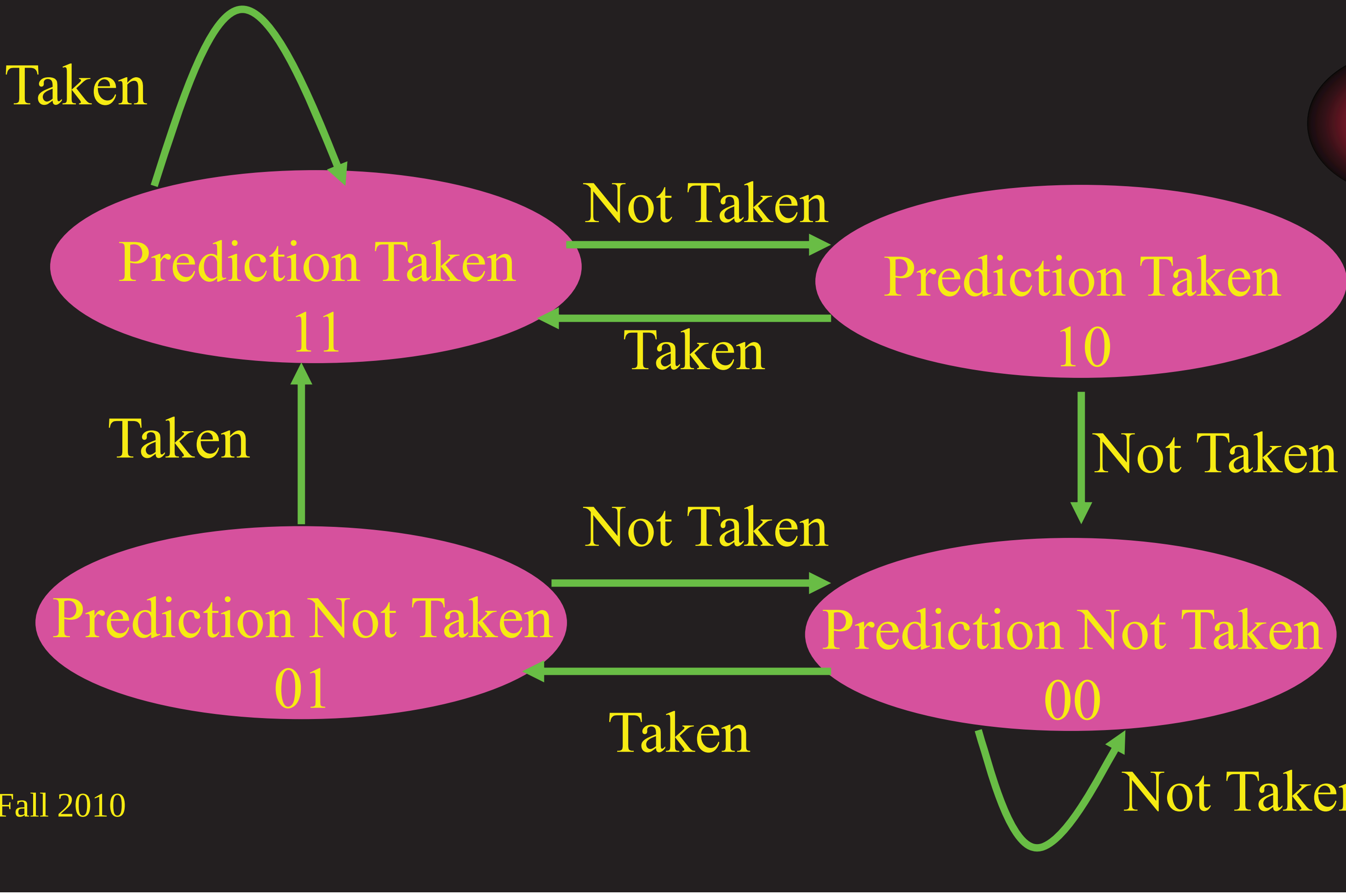
- Branch Prediction
  - Simplest way to have dynamic bra prediction is via the so called prediction bu or branch history table — A table whose en are indexed by lower portion of the ta address.

Beyond *RISC*

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

Beyond *RISC*

- Branch Prediction



Beyond *RISC*

Branch Prediction

- n-bit Saturation counter: An entry ha corresponding history feature. A taken br increments the counter and untaken br decrement the counter. A branch is not taken i counter is below $2^{(n-1)}$.

Beyond *RISC*

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

Beyond *RISC*

- Question
  - With respect to our earlier definition of *C* time, discuss how the performance can improved?

$$T = I_c * CPI * \tau = \sum_{i=1}^{n} \left(CPI_i * I_i\right) *$$

Beyond *RISC*

- Beyond RISC
  - Is it possible to achieve a performa beyond what is being offered by *RISC*

Beyond *RISC*

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

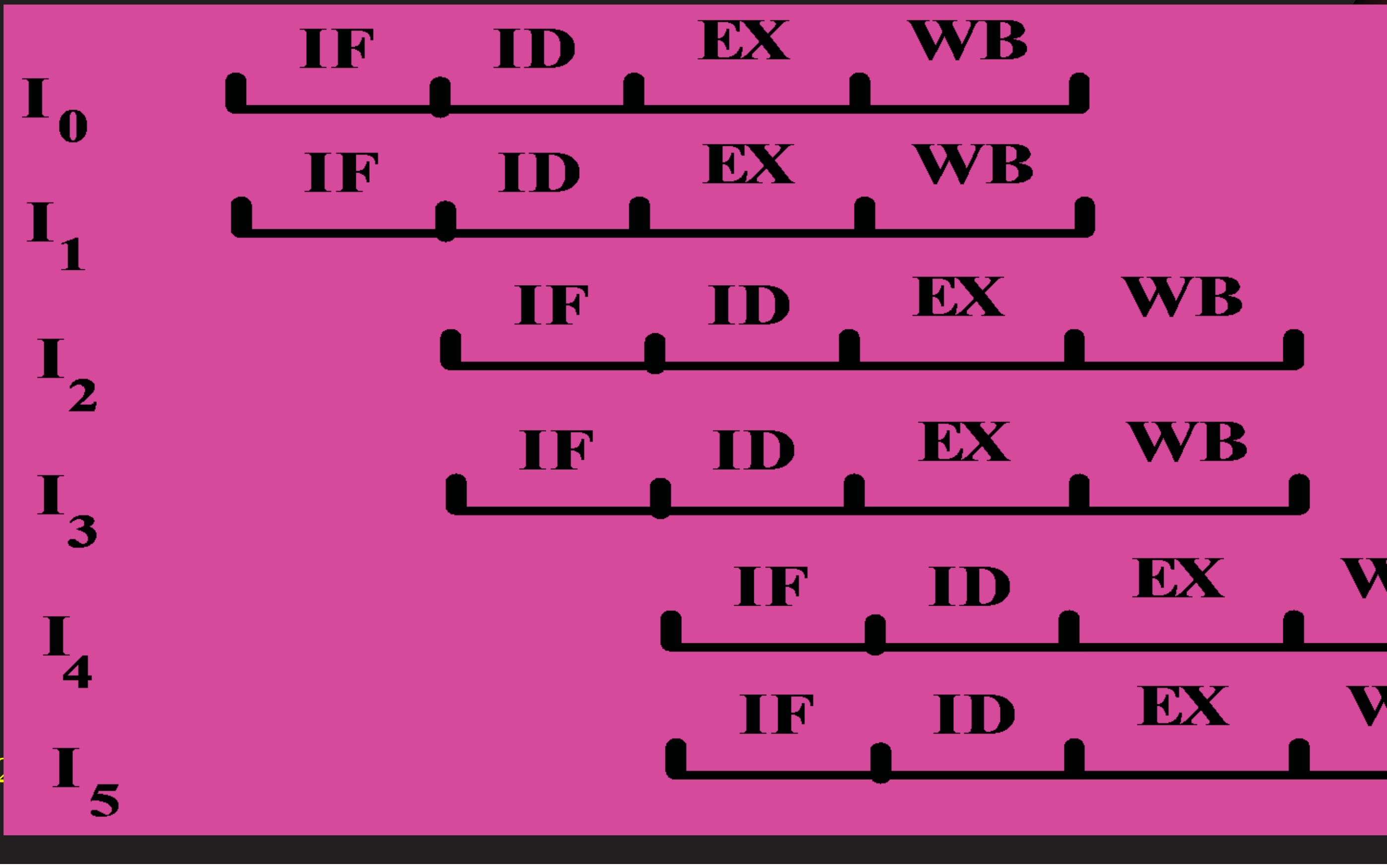
Super Scalar System

- Beyond RISC
  - A super scalar processor reduces the average nun of clock cycles per instruction beyond what is poss in a pipeline scalar *RISC* processor. This is achie 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

Beyond RISC

- Instruction Timing in a super scalar processor



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 ab
    - Read after write (flow) dependency
    - Write after read (anti) dependency
    - Write after write (output) dependency
  - The literature has referred to read after write as dependency, and write after read or write after wri false dependency.

Super Scalar System

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

- Storage medium holds different values different computations.

Super Scalar System

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

Super Scalar System

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

Super Scalar System

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

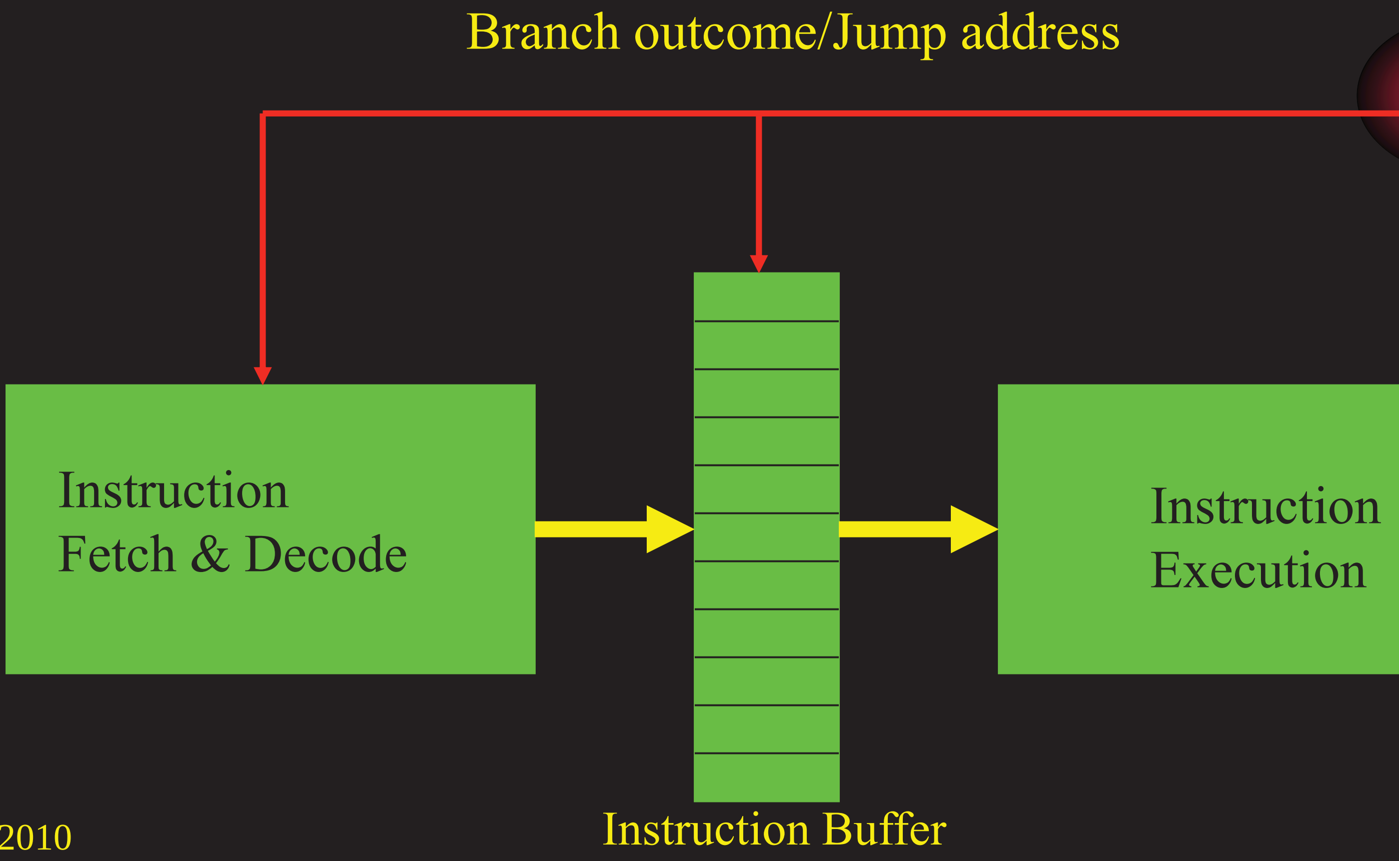
Super Scalar System

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

- Superscalar processors exploit the sec alternative.

Super Scalar System

- General Configuration



Super Scalar System

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

Super Scalar System

- General Configuration
  - Systems having this organization emp aggressive techniques to exploit instruc level parallelism.

Super Scalar System

- General Configuration
  - Wide dispatch and issue paths,

    Fetch, decode, and issue several instructions
  - Large issue buffer,
- Large pool of physical registers,

  Register Renaming – False Dependence
- Large number of parallel functional units,

  Resource Dependence
- Speculation of past multiple branches.

  Control Dependence

Are some techniques that allow aggressi exploitation of Instruction Level Parallel

Super Scalar System

Flow of Operations

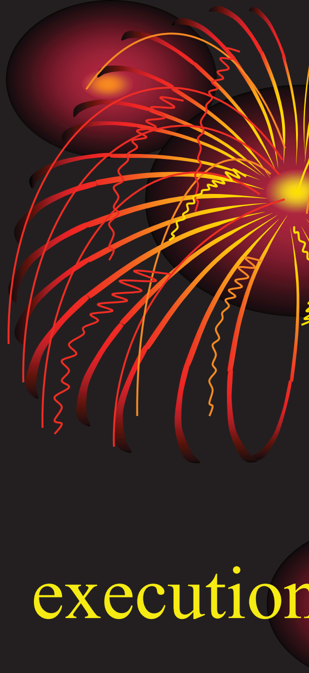
- A typical superscalar processor fetches and dec several incoming instructions at a time.
- The outcomes of conditional branch instructions usually predicted in advance to ensure an uninterru stream of instructions
- The incoming instructions are then analyzed for and structural dependencies, and then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

Super Scalar System

Flow of Operations

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

Super Scalar System



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

Super Scalar System

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

Super Scalar System

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

Super Scalar System

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

Super Scalar System

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

Super Scalar System

- Program Representation — High Level Constr

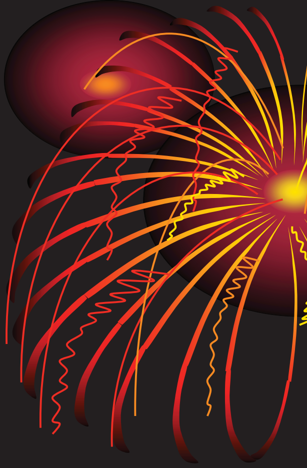
```
For 0 = i < last
        If a(i) > a(i+1)
                temp = a(i)
                 a(i) = a(i+1)
                 a(i+1) = temp
End
```

Super Scalar System

▶Program Representation — Assembly code

L2:	Move	r_3, r_7	r_7 points to an element of the
	LW	$r_8, (r_3)$	r_8 holds the ith element of the
	Add	$r_3, r_3, 4$	advancing the index
	LW	$r_9, (r_3)$	r_9 holds the i+1th element of
	Ble	$r_8, r_9, L3$	
	Move	r_3, r_7	In this block ith and i+1th ele
	SW	$r_9, (r_3)$	are swapped
	Add	$r_3, r_3, 4$	
	SW	$r_8, (r_3)$	
	Add	$r_5, r_5, 1$	
L3:	Add	$r_6, r_6, 1$	r_6 holds the index
	Add	$r_7, r_7, 4$	
	Blt	$r_6, r_4, L2$	r_4 holds the “last”

Super Scalar System



Processing Flow

- During the course of execution, the sequenc executed instructions forms a dynamic instruc stream.
- As long as instructions to be executed sequential, static instruction sequencing car entered into the dynamic instruction sequencin incrementing the program counter.

Super Scalar System

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

Super Scalar System

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

Super Scalar System

- Control Dependencies — Straight line code
  - Once a basic block is entered, its instructions fetched and execute to completion, there sequence of instructions in a basic block ca initiated into a conceptual window of execution.
  - Once the instructions are initiated, they are fr execute in parallel, subject only to the dependence constraints and availability of hardware resources.

Super Scalar System

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

Super Scalar System

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

Super Scalar System

Processing Flow

- In our running example, the *ble* instruc creates a control dependence.
- To overcome this dependence, the branch c be predicted as not taken and he instructions between the branch and label 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 execu may begin execution subject to data depend constraints.

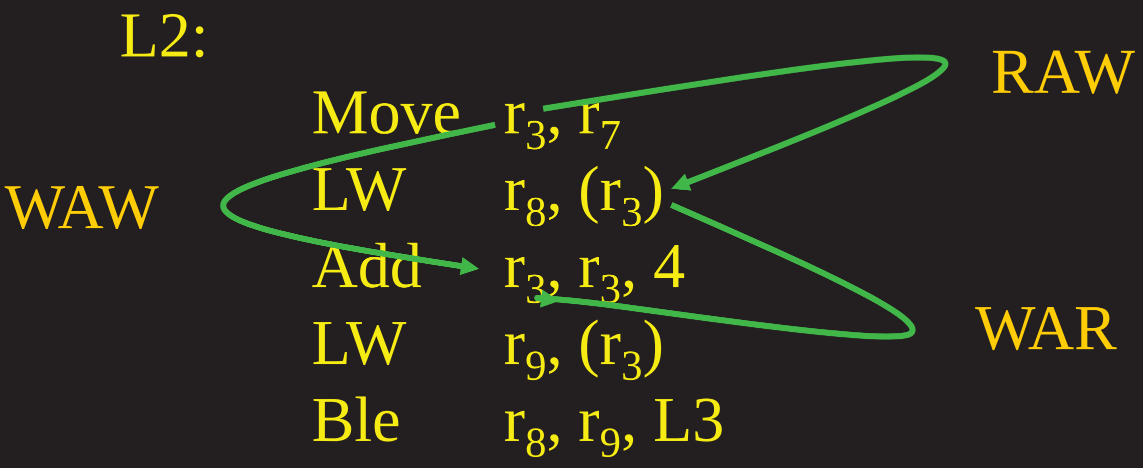
- Note that data dependence comes in the form o
  - Read After Write (RAW),
  - Write After Read (WAR), and
  - Write After Write (WAW).

Super Scalar System

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

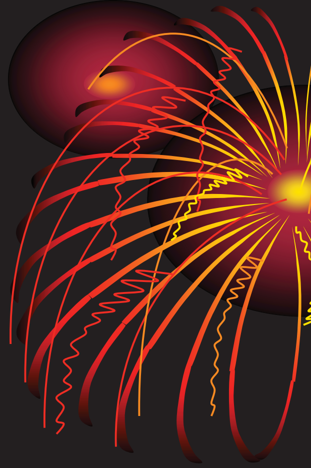
- Processing Flow
  - After resolving control and artif dependencies, instructions are issued and be execution in parallel.
  - The hardware form a parallel execu schedule.
  - The execution schedule takes constraints s as true data dependence and hardware reso constraints into account.

Super Scalar System

Processing Flow

- A parallel execution schedule means instructions complete in an order different instructions order dictated by the sequen execution model.
- Speculative execution means that s instructions may complete execution bey the scope of the sequential execution model.

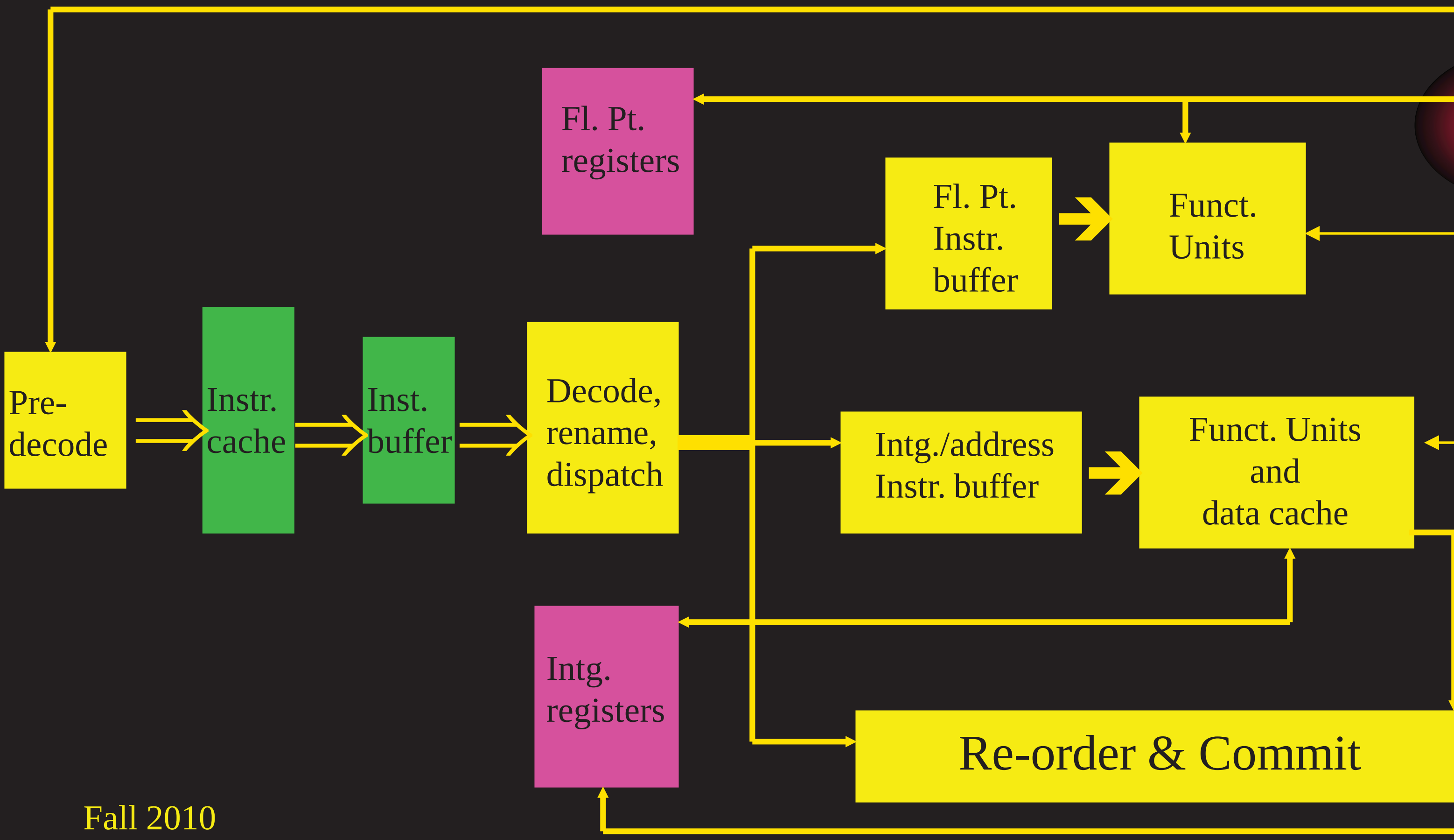
Super Scalar System



- Processing Flow
  - Speculative execution implies that the execu
    results cannot be recorded permanently right awa
  - As a result, results of an instruction must be held
    temporary status until the architectural state ca
    updated.
  - Eventually, when it is determined that the seque
    model would have executed an instruction,
    temporary results are made permanent by upda
    the architectural state — Instruction is committe
    retired.

Super Scalar System

Super Scalar Architecture



Fall 2010

Super Scalar System

- Instruction Fetch and Branch Prediction
  - For a super scalar implementation, the fetch p must be able to fetch multiple instructions cycle. To achieve this, it has been found usef separate the instruction cache from the data cac
  - The number of instructions fetched per c should at least match the peak instruction dec and execution rate.
  - Instruction buffer is used to smooth the instruc fetch irregularities due to cache misses branches.

Super Scalar System

Instruction Fetch and Branch Prediction

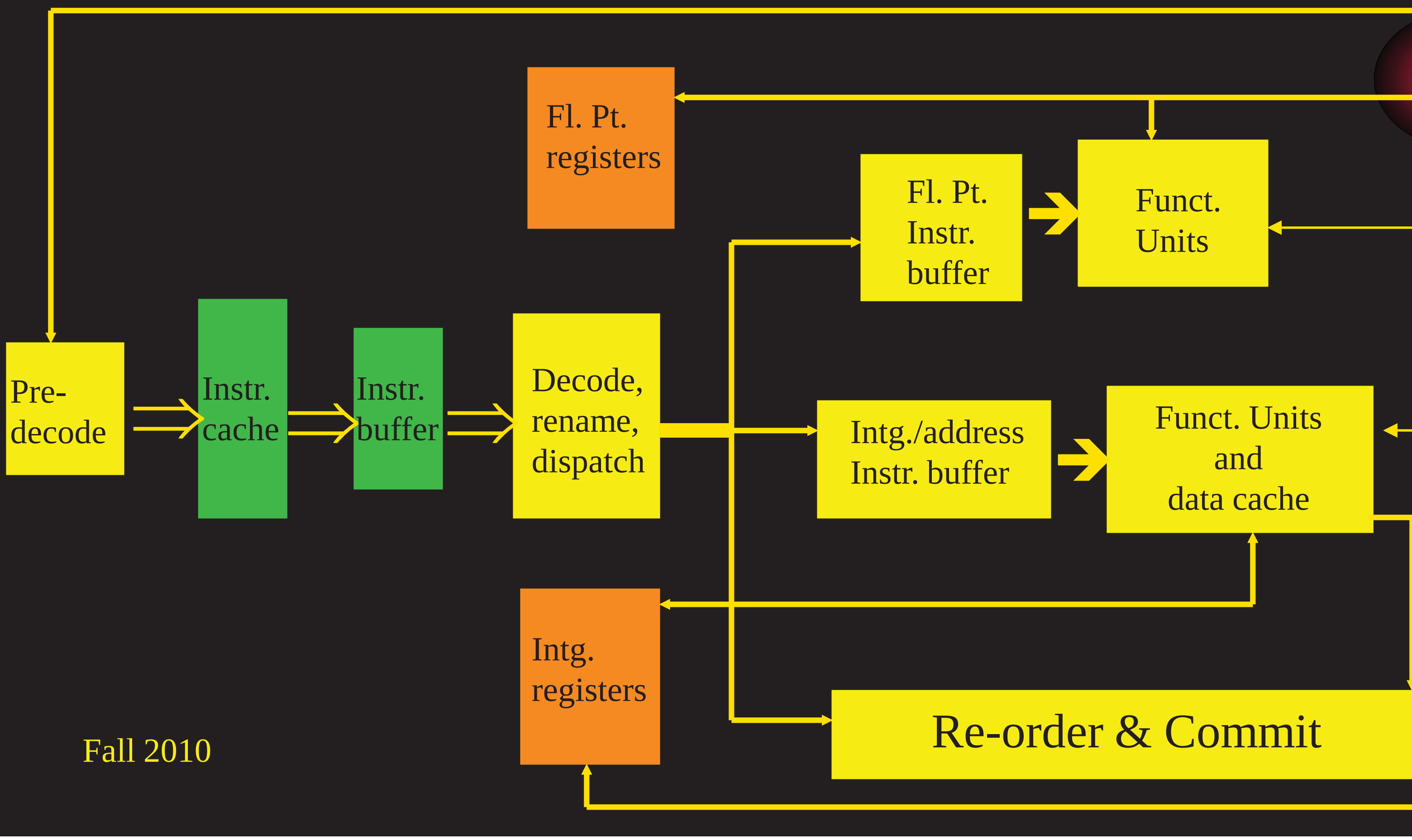
- The default instruction fetching method is increment the program counter by the numbe fetched instructions.
- The handling of branch (specially the conditi branch) is critical to good performance of a s scalar processor. Processing conditional br involves:
  - Recognizing the branch,
  - Determining the branch outcome,
  - Computing the branch target, and
  - Transferring control.

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Recognizing the branch
    - In general, recognizing the instruction type advanced, can speed up the instruction flow to execution buffer.
    - This can be achieved by pre-decoding instruction prior to its residence in the cach Each instruction in the cache is extended by bits.

Super Scalar System

- Recognizing the branch



Super Scalar System

- Instruction Fetch and Branch Prediction
  - Determining the branch outcome
    - Several techniques can be used to predict outcome of a branch.
    - Some predictors use static information, others dynamic information based on the branch histor
    - Note that, if prediction was incorrect, instru fetching must be redirected to the correct pat addition, if instructions were executed speculati they must be purged and their results mus nullified.

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Computing the branch target
    - Early calculation of the target branch c
      improve the performance.
    - This can be sped up by having a branch ta
      buffer that holds the target address that was
      the last time the branch was executed.
    - As an example PowerPC 64 uses the Bra
      Target Address Cache.

Super Scalar System

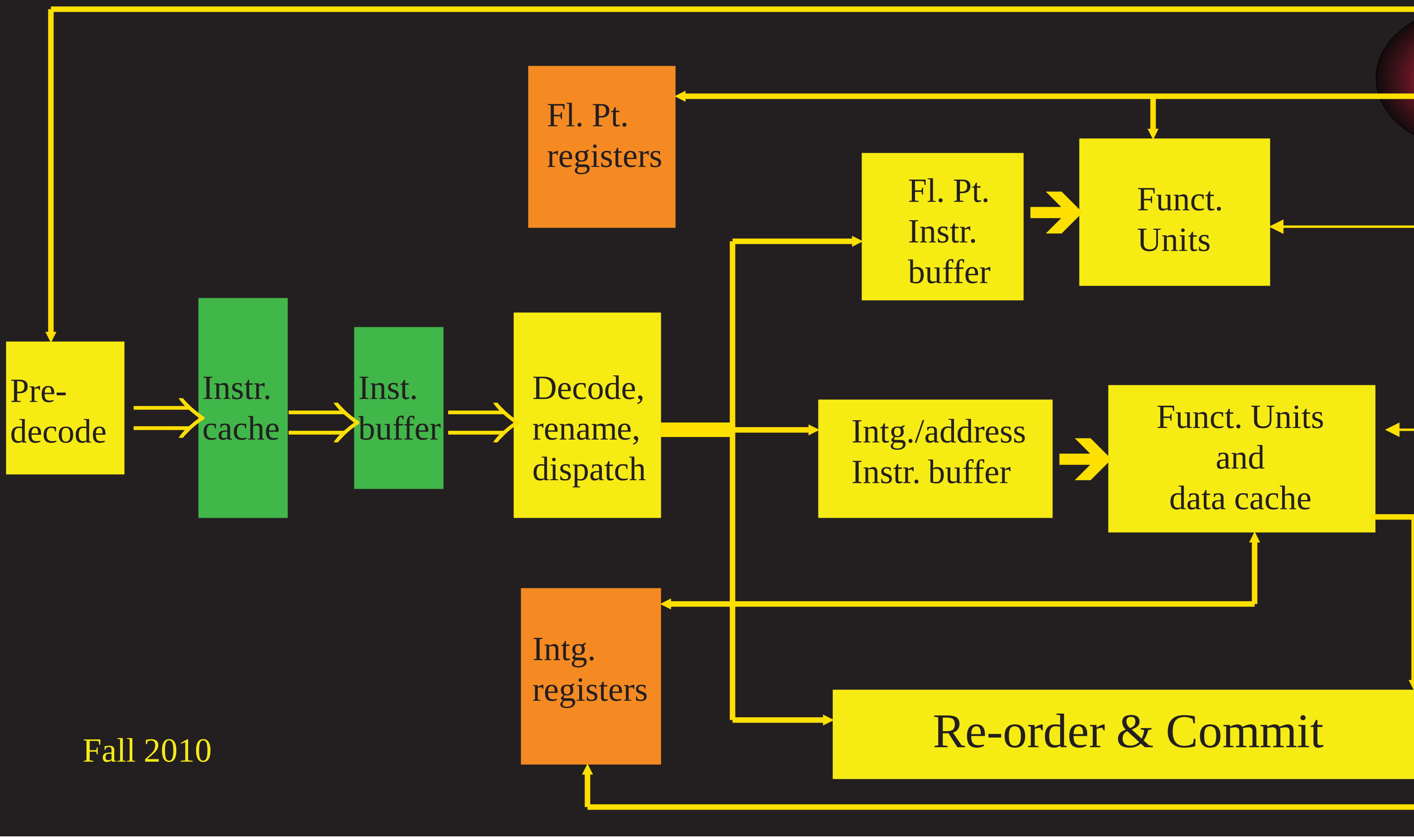
- Instruction Fetch and Branch Prediction
  - Transferring Control
    - In case of a taken branch, there is at least one c cycle delay — to recognize the branch, calculat program counter, and fetching instruction fron target address.
    - Several techniques can be used to mask out delay:
      - Use instructions in the instruction buffer,
      - Fill out instruction buffer by taken and not taken path
      - Use of delayed branch.

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Instruction Decoding, Renaming, and Dispatch
    - At this stage, instructions from fetch buffer removed, examined, control and data depend relationships are set up, and dispatched to instruction buffers associated with the funct units.
    - Often to improve the degree of parallelism, this tries to remove the false dependence by renar the physical storage locations as defined in instructions.

Super Scalar System

Instruction Fetch and Branch Prediction



Super Scalar System

Register Renaming

- $R_3 \Leftarrow R_3 \text{ op } R_5$ $\qquad$ $R_{3b} \Leftarrow R_{3a} \text{ op } R_{5a}$
- $R_4 \Leftarrow R_3 + 1$ $\qquad$ $R_{4b} \Leftarrow R_{3b} + 1$
- $R_3 \Leftarrow R_5 + 1$ $\qquad$ $R_{3c} \Leftarrow R_{5a} + 1$
- $R_7 \Leftarrow R_3 \text{ op } R_4$ $\qquad$ $R_{7b} \Leftarrow R_{3c} \text{ op } R_{4b}$

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

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Register Renaming
    - There are two register renaming techniqu
      - The physical register file is larger logical register file,
      - The physical register file is the same siz the logical register file.

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Register Renaming — The physical register file is large logical register
    - A mapping table is used to associate a physical register the current value of a logical register.
    - For each logical destination register a physical register, the list of free registers, is extracted and association be the two is recorded in the mapping table.
    - As part of rename operation, for each source logical reg the mapping table is investigated to find its associated ph register.

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Register Renaming — The physical register is larger than logical register file

Before Add r_3,r_3,4

Mapping Table

r_0	R_8
r_1	R_7
r_2	R_5
r_3	R_1
r_4	R_9
	⋮

Free list R_2,R_6,R_{13}

After Add R_2,R_1,4

Mapping Table

r_0	R_8
r_1	R_7
r_2	R_5
r_3	R_2
r_4	R_9
	⋮

R_6,R_{13}

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Register Renaming — The physical register file is large logical register file
    - After a physical register has been read for the last time, be returned to the free list to be reused.
    - A counter can be associated to each physical register,
      - It will be incremented whenever it is renamed as a source,
      - It will be decremented each time an instruction issue actually reads a value from the register.
      - A physical register is returned back to free space, if its cou zero and corresponding logical register is renamed.

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Register Renaming — The physical register fi the same as logical register file
    - This model uses a so called reorder buffer maintains proper instruction ordering (instruc that are dispatched but not yet completed) precise interrupts.
    - Reorder buffer is organized as a circular queue.

Super Scalar System

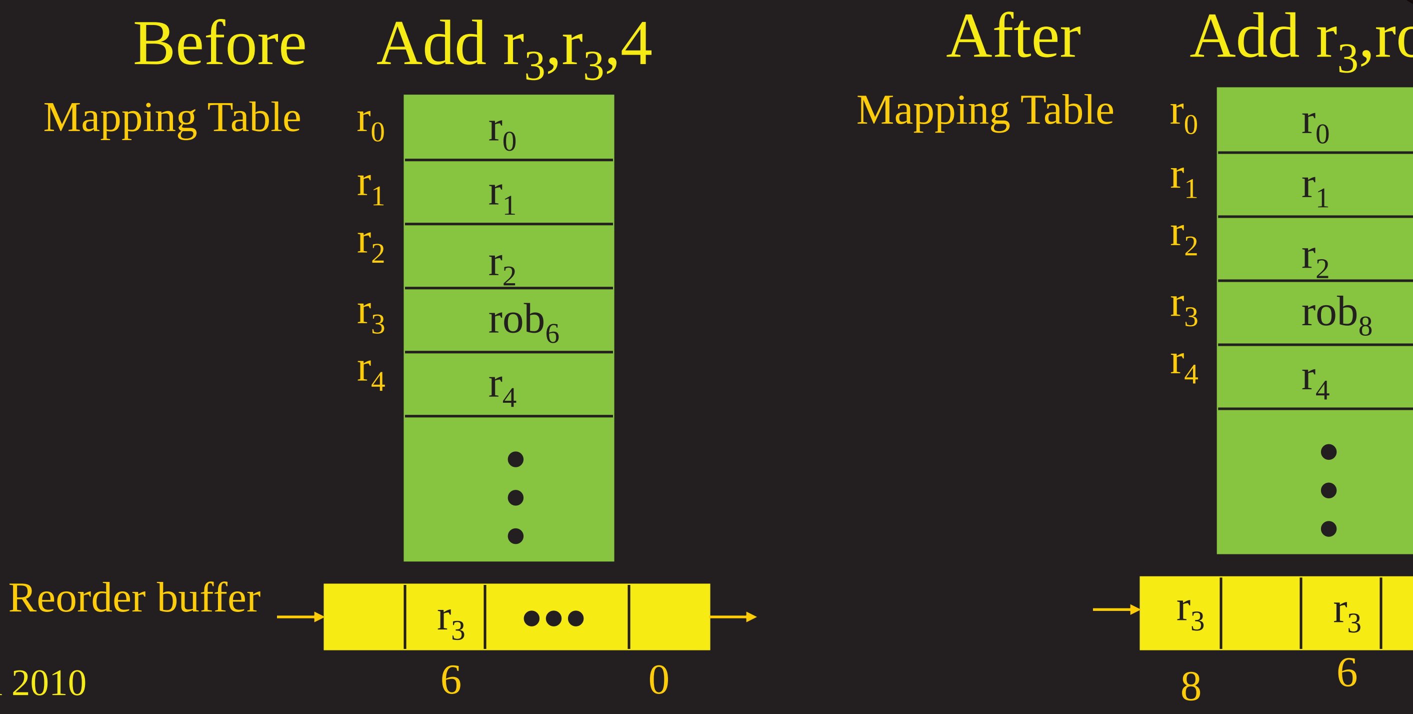
- Instruction Fetch and Branch Prediction
  - Register Renaming — The physical register fi the same as logical register file
    - As instructions are dispatched based on seque ordering of the program, they are entered into reorder FIFO buffer.
    - As instructions complete execution, their r values are inserted into the previously assi entry, wherever it may happen to be in the re buffer.

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Register Renaming — The physical register fi the same as logical register file
    - When an instruction reaches the head of the bu if it is completed, it is removed from the buffer its result is stored into the register file.
    - An incomplete instruction at the head of the b blocks the buffer until it completes.

Super Scalar System

- Instruction Fetch and Branch Prediction
  - Register Renaming — The physical register fi
    the same as logical register file



Before Add r_3,r_3,4

Mapping Table

r_0	r_0
r_1	r_1
r_2	r_2
r_3	rob_6
r_4	r_4
	• • •

Reorder buffer

	r_3	•••	
	6		0

After Add r_3,ro

Mapping Table

r_0	r_0
r_1	r_1
r_2	r_2
r_3	rob_8
r_4	r_4
	• • •

r_3		r_3
8		6

Super Scalar System

- Instruction Issue and Execution
  - Instruction issue is defined as the run-t checking for availability of data and resourc
  - After decode/rename/dispatch phase, the f step is to determine which instruction types be issued for execution.
  - Three topology has been discussed:

Super Scalar System

- Instruction Issue and Execution
  - Single Queue Model
    - If there is a single queue and no out-of-order exec then renaming is not necessary and operand availa can be managed via a simple reservation bits assig each register.
    - A register is reserved when an instruction modifyin issued.
    - A register is cleared when the instruction completes.
    - An instruction is issued if there is no functional co and no reservation on its operand.

Super Scalar System

- Instruction Issue and Execution
  - Multiple Queue Model
    - In this case, instructions are issued from each q in order, but the queues may issue out of order respect to one another.
    - The instruction queues are organized accordi the instruction types.

Super Scalar System

Instruction Issue and Execution

Reservation Station

- In this case, instructions are issued out-of-order of the reservation stations simultaneously mo their source operand for availability of data.

- When an instruction is dispatched to the reserv station, any already available operand values are from the register file into the reservation station

Super Scalar System

- Instruction Issue and Execution
  - Reservation Station
    - Reservation station compares the ope designators of unavailable data with the r designators of completing instructions. When t is a match, the result value is pulled into matching reservation station — sort of forwardi
    - When all the operands are available in reservation station, the instruction may be issued

Super Scalar System

Committing State

- The final phase of an instruction is commit or r
state.

- In the commit state, the effects of the instructio
allowed to modify the logical state of the process.

- The purpose of this phase is to implement
appearance of a sequential execution model
though the actual execution is not sequential due t
speculative execution and out-of-order execution.

Super Scalar System

- Committing State
  - In one approach, the state of the machin certain points are saved (check pointed) ei in a history buffer or a checkpoint.
  - Instructions update the state of the machin they execute and when a precise state is nee it is recovered from the history buffer.

Super Scalar System

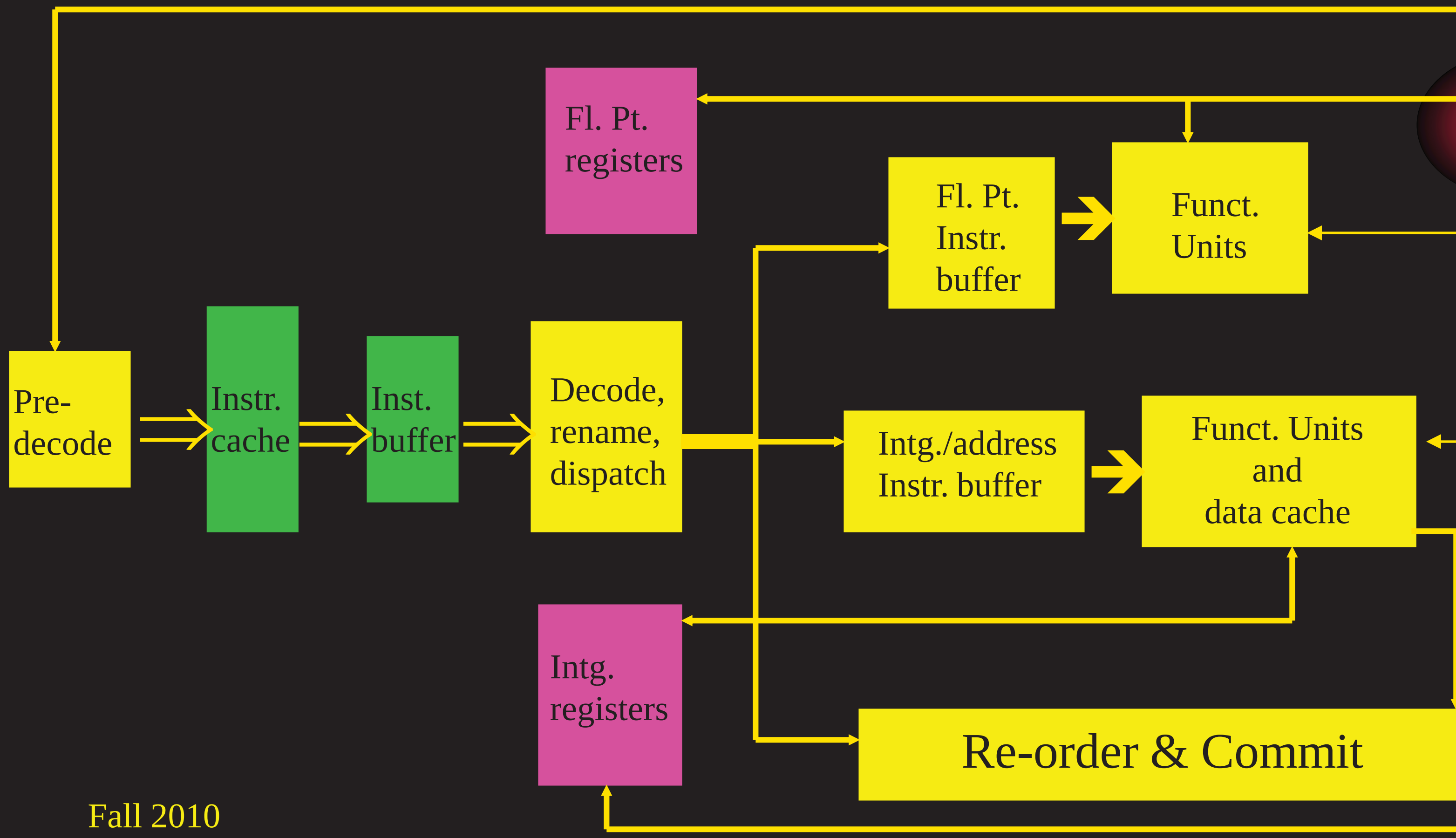
- Committing State
  - In another approach, the state of the machin separated into the:
    - Physical state, and
    - Logical (architectural) state.

Super Scalar System

- Committing State
  - The physical state is updated as the oper completes.
  - The logical state is updated in sequential prog order, as the speculative state of the operatio cleared. The speculative state is maintained in a reo buffer.
  - To commit an instruction, its result has to be rem from the reorder buffer into the architectural reg file.

Super Scalar System

▶ MIPS R1000 – General Configuration

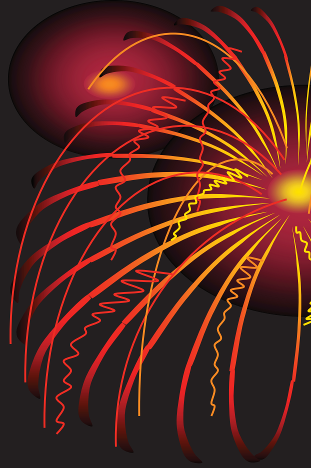


Fall 2010

Super Scalar System

- MIPS R1000
  - Performs extensive dynamic scheduling,
  - Fetches four pre-decoded instructions at a t
    from an instruction cache of 512 lines,
  - Pre-decoding extends each binary instruc
    by 4 bits,
  - Physical register file (64) is twice the log
    register file (32),

Super Scalar System



- MIPS R1000
  - Supports a Branch prediction table of 512 entries a contained within the instruction cache mechar Each entry holds 2-bit counter to show the hi information,
  - For predicted taken branch, one cycle is neede redirect instruction fetching. During this c sequential instructions are fetched and placed resume cache — a space of four instructions blocks,

Super Scalar System

- MIPS R1000
  - In case of a predicted branch, a snapshot of register mapping table is taken – up to snapshots can be stored at the same time,

Super Scalar System

- MIPS R1000
  - Dispatches up to four instructions into t instruction queues:
    - Memory,
    - Integer, and
    - Floating point
  - Instruction queues are 16 entries deep and a full queue can block dispatch unit,

Super Scalar System

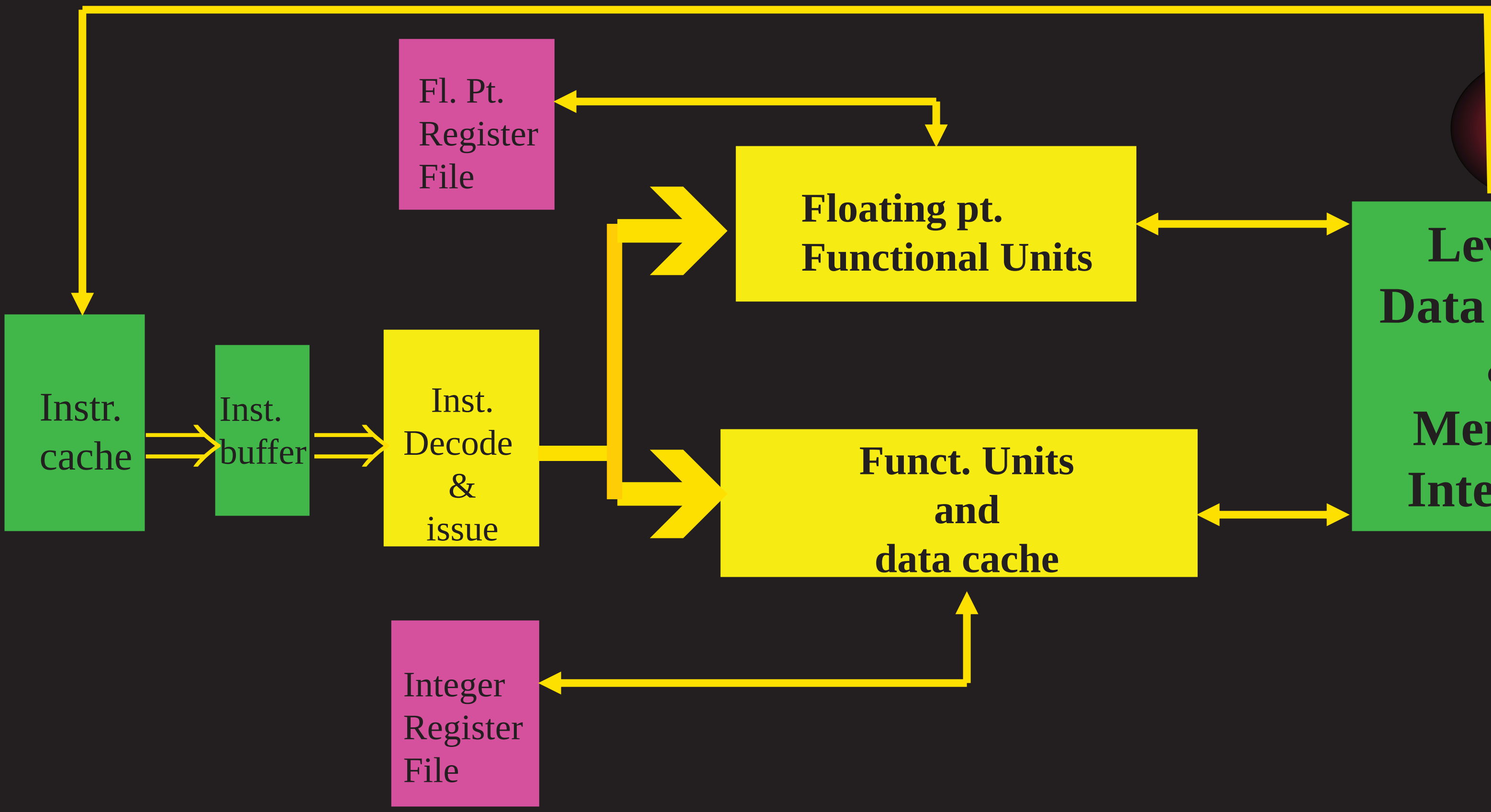
- MIPS R1000
  - Supports five functional units:
    - An address adder,
    - Two integer ALUs,
    - A floating point multiplier/divider/square rooter
    - A floating point adder

Super Scalar System

- MIPS R1000
  - Supports an on-chip 2-way set associative Kb) primary cache and an off-chip secon cache,
  - Uses re-order buffer to maintain a precise s at the time of exception,
  - Commits instructions in the original prog sequence, up to four at a time.

Super Scalar System

- DEC Alpha 21164 – General Configuration



Super Scalar System

DEC Alpha 21164

- Compromises dynamic scheduling in favor of a hi clock rate,
- Fetches four instructions at a time from an 8 K instruction cache,
- Has two instruction buffers, each capable of ho four instructions,
- Issues instructions from instruction buffer in program order. An instruction buffer must be em before the other being used,

Super Scalar System

- DEC Alpha 21164
  - Instruction cache is enhanced by a bra history table, each entry is a 2-bit counter,
  - At most one predicted and yet unresol branch can exist at a time,
  - Following instruction fetch and dec instructions are inspected and upon availab of operands they are issued — during process instructions are not allowed to pass another,

Super Scalar System

- DEC Alpha 21164
  - Supports four functional units:
    - Two integer ALUs,
    - A floating point adder, and
    - A floating point multiplier,
  - Supports two level of on-chip caches:
    - Primary cache – dedicated each 8 Kbytes,
    - Secondary cache – unified 96 Kbytes.

Super Scalar System

- Limitations
  - Limit on instruction level parallelism,
  - Complexity of superscalar.

Very Long Instruction Word — VLIW

- Hardware complexity and scalability two major issues that question the vali of the Super Scalar machines.

- Consider the following cases:

Fall 2010

Very Long Instruction Word — VLIW

- The instruction issue logic of PA-8000 four issue Super Scalar machine with instruction queue entries) occupies 20% the die area.
- As the issue width increases the need register renaming, complexity of bypass forwarding, and interlocks increa dramatically.

Very Long Instruction Word — VLIW

- Increasing hardware complexity pays b
  lesser dividends:
  - Performance of a 200 MHZ MIPS R500
    single issue machine) on SPEC95 is about
    of a 200MHZ MIPS R10000 (a four i
    machine).

Very Long Instruction Word — VLIW

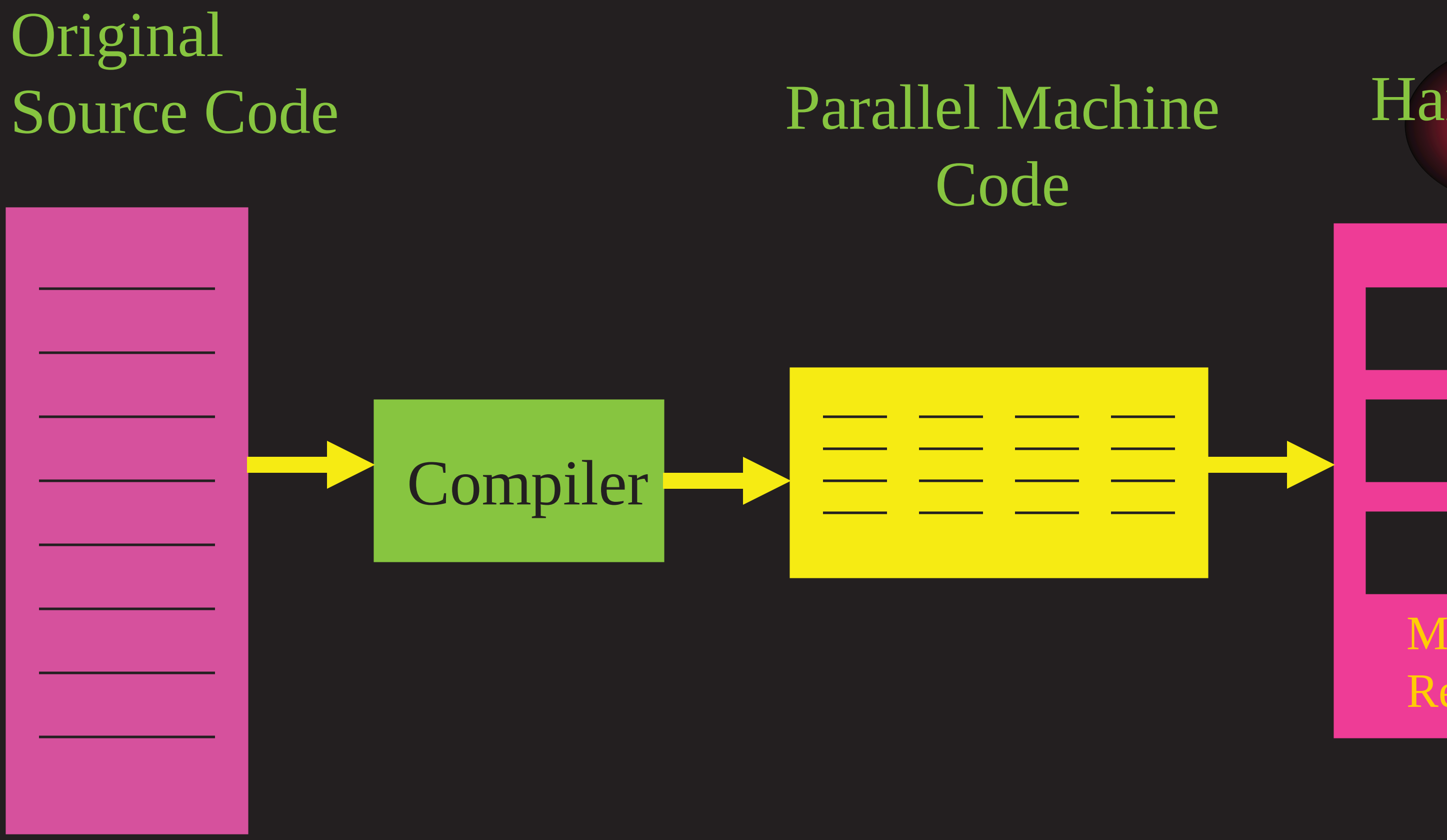
- Commercial VLIW machines w introduced in early 80's.
  - Multiflow delivered:
    - Trace/200 — 256-bit 7 wide issue machine,
    - Trace/300 — 256-bit 7 wide issue machine, and
    - Trace/500 — 512, 1024-bit 14, 28 wide machine.
  - Cydrome delivered:
    - Cydra 5 — 256-bit 6 wide issue machine.

Very Long Instruction Word — VLIW

- Present commercial VLIW machines:
  - IA-64 is a joint venture between Intel and H
  - Philips Trimedia processor, a DSP chip multimedia applications,
  - CRUSOE by Transmeta.

Very Long Instruction Word — VLIW

- Basic Principle of VLIW Architecture



Original
Source Code

Compiler

Parallel Machine
Code

Ha

M
Re

Very Long Instruction Word — VLIW

- Very Long Instruction Word (VLIW) des takes advantage of instruction parallelism reduce number of instructions by pack several independent instructions into a v long instruction.
- The principle behind VLIW is similar that of parallel computing — exec multiple operations in one clock cycle.

Very Long Instruction Word — VLIW

- VLIW arranges all executable operations in word simultaneously — many stati scheduled, tightly coupled, fine-grained operat execute in parallel within a single instruc stream.
- Naturally, the more densely the operations ca compacted, the better the performance (lo number of long instructions).

Very Long Instruction Word — VLIW

- During compaction, NOOPs can be used
  operations that can not be used.

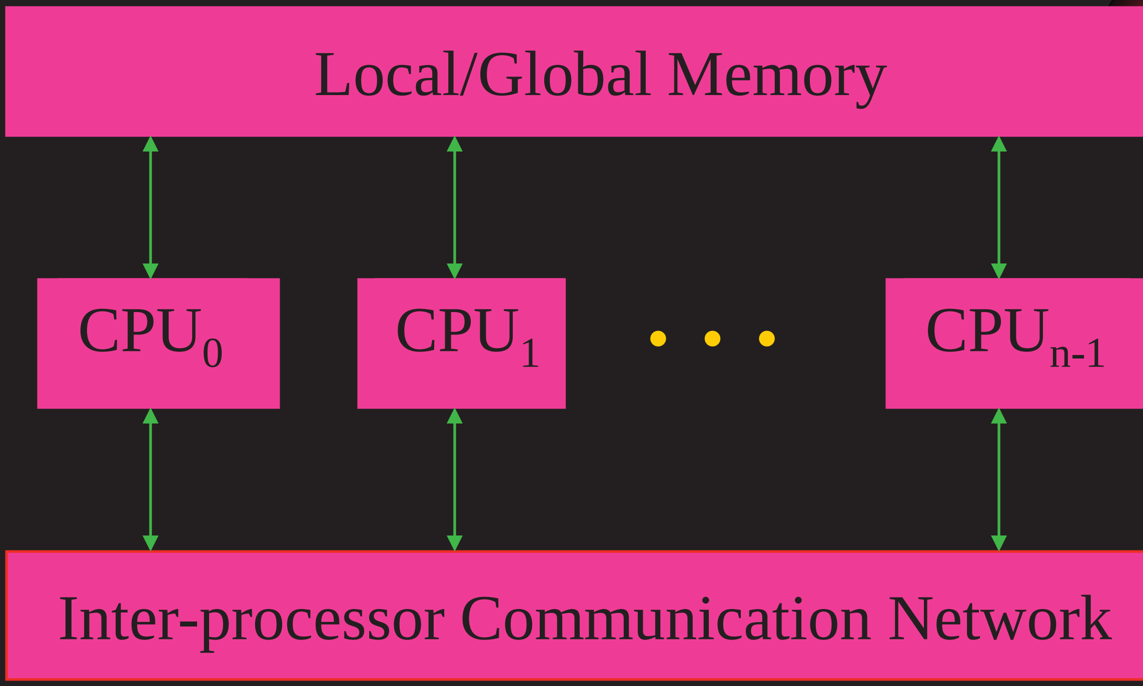
- To compact instructions, software must be
  to detect independent operations.

Very Long Instruction Word — VLIW

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

Very Long Instruction Word — VLIW

- Block Diagram



Local/Global Memory

CPU_0 CPU_1 . . . CPU_{n-1}

Inter-processor Communication Network

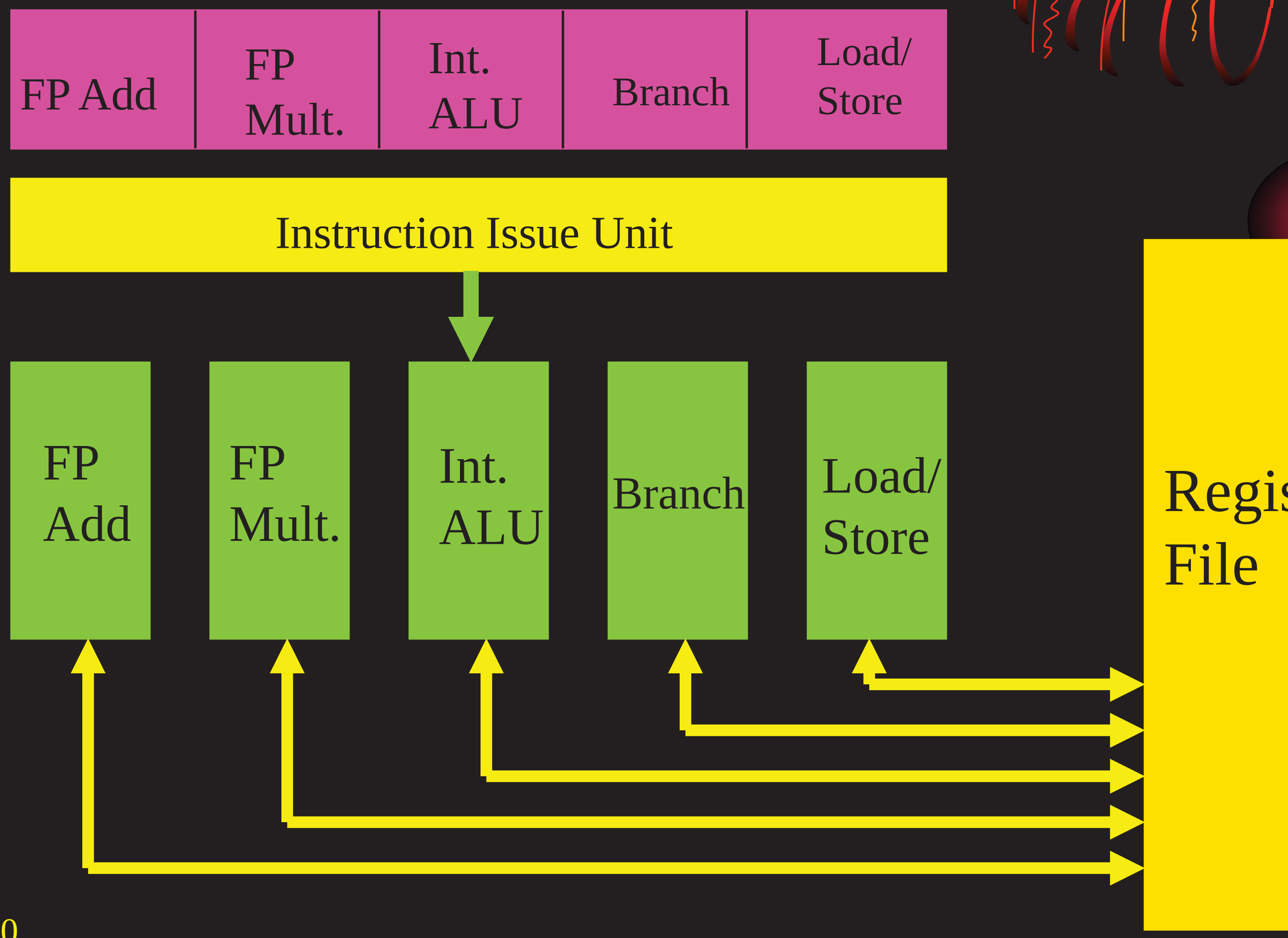
Very Long Instruction Word — VLIW

- This organization is very similar to heterogeneous multiprocessor system, howeve
  - Each long instruction contains op. code to control individual processors,
  - Instructions are in a single flow of control — instruction is fetched, all the processors do individual operations, then the next instructio fetched — Only one locus of control,
  - Instruction word completely controls all communication among the processors.

Very Long Instruction Word — VLIW

- VLIW fits somewhere between SIMD and MI in Flynn's taxonomy.
- VLIW consists of multiple functional unit single control unit, and a single monolithic reg file.
- The processor fetches from the instruction cac very long instruction containing a set of prim instructions, and dispatches them simultaneo for parallel execution.

Very Long Instruction Word — VLIW



FP Add | FP Mult. | Int. ALU | Branch | Load/ Store

Instruction Issue Unit

FP Add

FP Mult.

Int. ALU

Branch

Load/ Store

Regis File

Very Long Instruction Word — VLIW

- This architecture allows any functional u to access any registered data. This imp too many ports to fully connect the regi file to all functional units.
- More realistic solution, partitions regi file into banks and allows a subset functional units to have exclusive acces each register bank.

Very Long Instruction Word — VLIW

- Assume the following FORTRAN code 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) $t_6 = 4 * t_5$

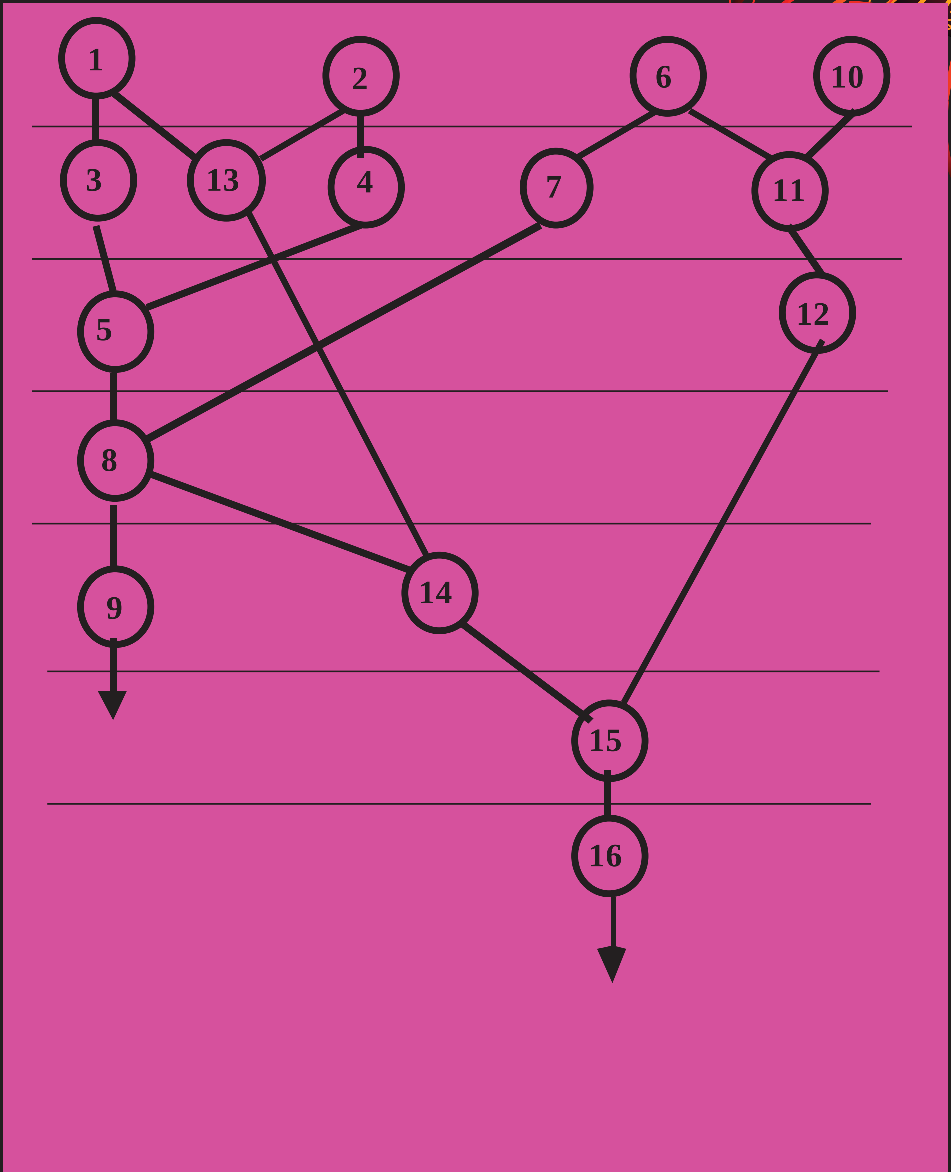
13) $t_7 = A + B$

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	BR
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

- Summary
  - VLIW – General Philosophy
  - VLIW – General Configuration
  - VLIW – Advantages and Disadvantages
  - VLIW – An example

Very Long Instruction Word — VLIW

- Assume a VLIW machine capable issuing two floating point operations, memory reference operations and integer/branch operation.
- Further assume the following loop:

Very Long Instruction Word — VLIW

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
BNZ	R_1, Loop	Branch if not zero

We will unroll the loop seven times to a optimal performance:

Very Long Instruction Word — VLIW

	Memory Reference 1	Memory Reference 2	Fl. Point 1	Fl. Point 2	Inte
1	LD F_0, 0(R_1)	LD F_6, -8(R_1)			
2	LD F_{10}, -16(R_1)	LD F_{14}, -24(R_1)			
3	LD F_{18}, -32(R_1)	LD F_{22}, -40(R_1)	AD F_4, F_0, F_2	AD F_8, F_6, F_2	
4	LD F_{26}, -48(R_1)		AD F_{12}, F_{10}, F_2	AD F_{16}, F_{14}, F_2	
5			AD F_{20}, F_{18}, F_2	AD F_{24}, F_{22}, F_2	
6	SD F_4, 0(R_1)	SD F_8, -8(R_1)	AD F_{28}, F_{26}, F_2		
7	SD F_{12}, -16(R_1)	SD F_{16}, -24(R_1)			
8	SD F_{20}, -32(R_1)	SD F_{24}, -40(R_1)			SUB
9	SD F_{28}, -48(R_1)				BNZ

Very Long Instruction Word — VLIW

VLIW Compiler Technology

- Effectiveness of VLIW architecture is he dependent on the exploitation of parallelisms in application program by the compiler.
- The compiler must be efficient and clever.
- VLIW compilers heavily use:
  - Speculative scheduling when branch is encountered,
  - Loop unrolling to expose more instruction level parallelis
  - Software pipelining,
  - Inlining to reduce the overhead associated with proc calls, and
  - Trace Scheduling.

Very Long Instruction Word — VLIW

- VLIW machine can actually run two pos paths of a branch.
- When the branch is computed at run time anc correct path is known, the incorrect branc discarded and execution continues as if there no branch.
- Consequently, the branch penalty can be al zero compared to a scalar processor.

Very Long Instruction Word — VLIW

Trace Scheduling

- This technique is applied once all intermed optimizations are complete and the code i machine level instructions.
- Trace is a linear section of code that mus executed together.
- Trace scheduling is composed on two parts:
  - Selection, and
  - Compaction

Very Long Instruction Word — VLIW

- Trace Scheduling
  - Selection modules, recursively, selects the t
    with highest probability of execution and se
    it to the compaction module.
  - Trace compaction uses several technique
    rearrange instructions within a trace
    minimize wasted instructions in a long w
    and consequently, to reduce the total numbe
    long words.

Very Long Instruction Word — VLIW

- Trace Scheduling
  - If an instruction in the trace is moved f before a conditional jump to after the jum copy of it must be placed in the off-trace e of the jump.

1: O_1

2: if cond. → 4: O_3

3: O_2 5: O_4

2: if cond. → $1'$:

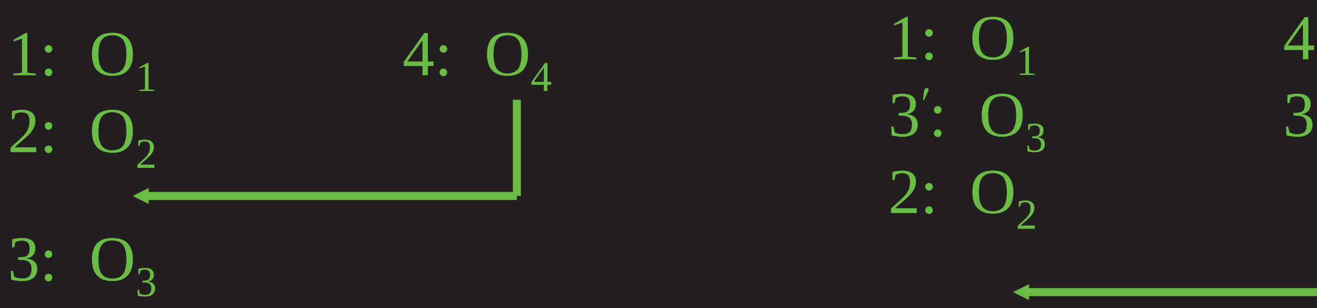
$1'$: O_1 4:

3: O_2 5:

Very Long Instruction Word — VLIW

Trace Scheduling

- If a trace operation is moved above a rejoin
  trace, then a copy must be placed on the
  trace rejoin edge.



1: O_1

4: O_4

2: O_2

3: O_3

1: O_1

4

3′: O_3

3

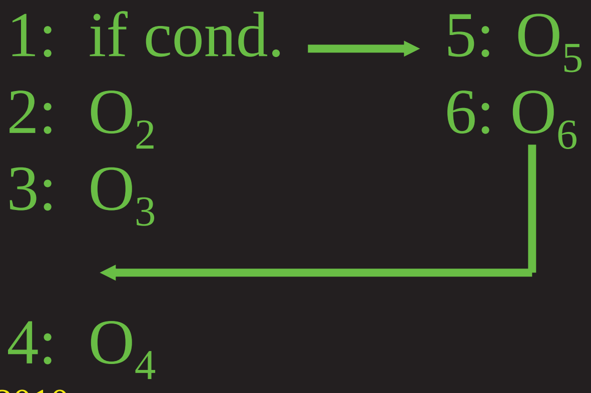
2: O_2

Fall 2010

Very Long Instruction Word — VLIW

Trace Scheduling

- Block *X* is said to post dominate block
  every path through *Y* must ultimately
  through *X*.

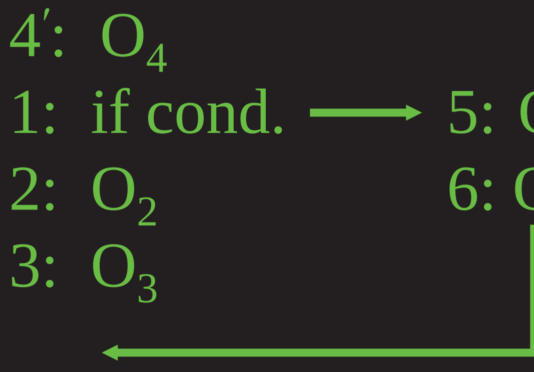


1: if cond. → 5: O_5

2: O_2 6: O_6

3: O_3

4: O_4



4′: O_4

1: if cond. → 5: O

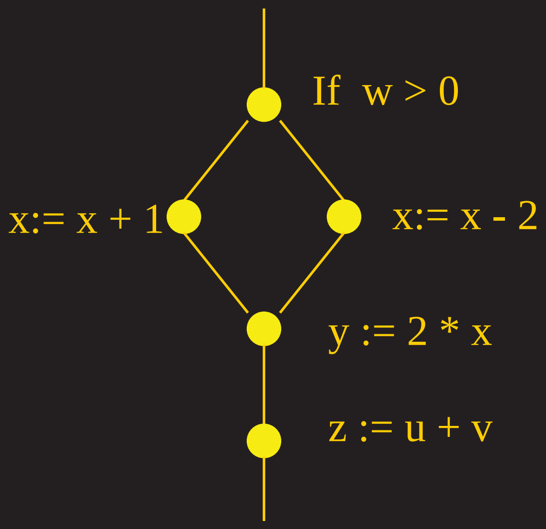
2: O_2 6: O

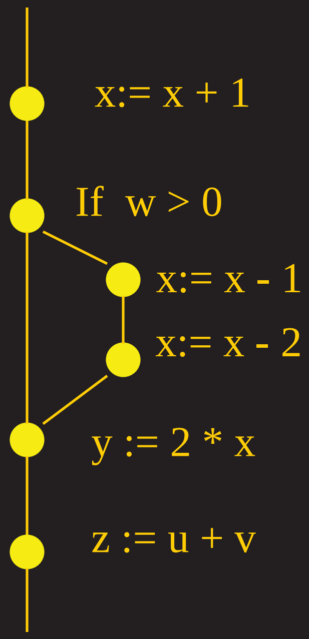
3: O_3

Very Long Instruction Word — VLIW

Trace Scheduling

- In some cases compensation code
bookkeeping code is inserted into the progra





Fall 2010

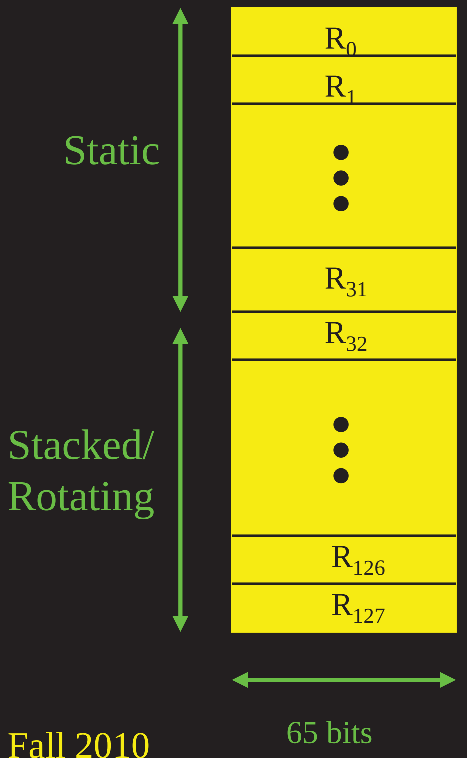
Very Long Instruction Word — VLIW

- IA-64 — General Philosophy
  - A full 64-bit address space,
  - Large directly accessible register file,
  - Enough instruction bits to communi information from compiler to hardware,
  - Ability to express large amount of Instruc Level Parallelism.

Very Long Instruction Word — VLIW

▶ IA-64 — Register Configuration

General Purpose Registers

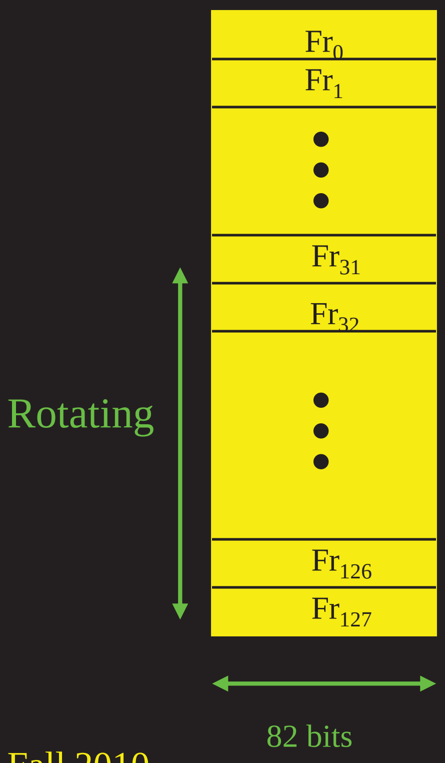


First 32 registers are used Statically, ar
rest will be used as Stacked/rotating re
Stacked/rotating registers are used to s
Procedure calls.

Very Long Instruction Word — VLIW

IA-64 — Register Configuration

Floating Point Registers



Fr_{0}

Fr_{1}

•
•
•

Fr_{31}

Fr_{32}

•
•
•

Fr_{126}

Fr_{127}

Rotating

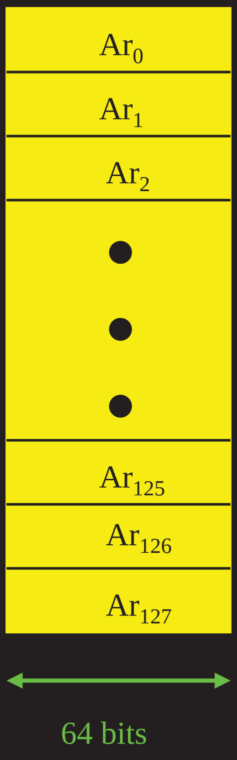
82 bits

Register rotating is used to ease the Task of allocating registers in softw Pipelined loops.

Very Long Instruction Word — VLIW

▶IA-64 — Register Configuration

Special Application Registers



Ar_0

Ar_1

Ar_2

●

●

●

Ar_{125}

Ar_{126}

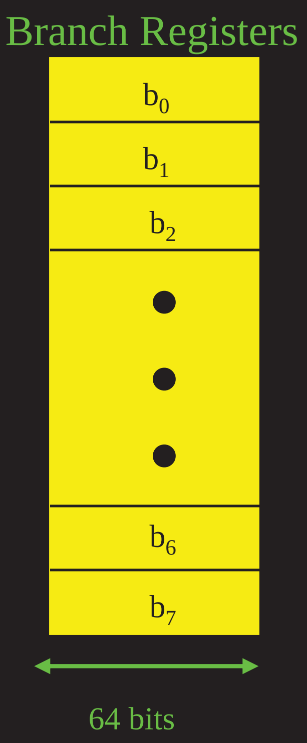
Ar_{127}

64 bits

Special purpose application registers are used to support features such as Register stack, Software pipelining,.

Very Long Instruction Word — VLIW

IA-64 — Register Configuration



Branch Registers

b_0

b_1

b_2

●

●

●

b_6

b_7

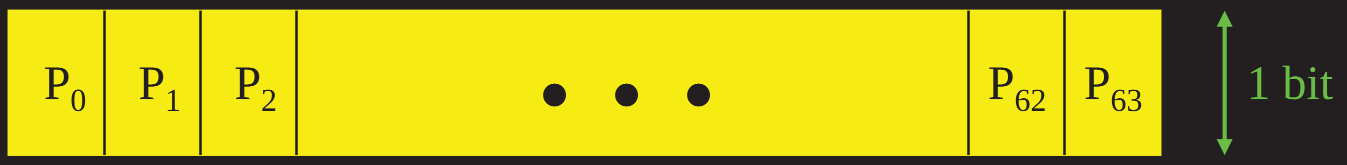
64 bits

Branch registers are used
For indirect branches.

Very Long Instruction Word — VLIW

▶IA-64 — Register Configuration

Predicate Registers

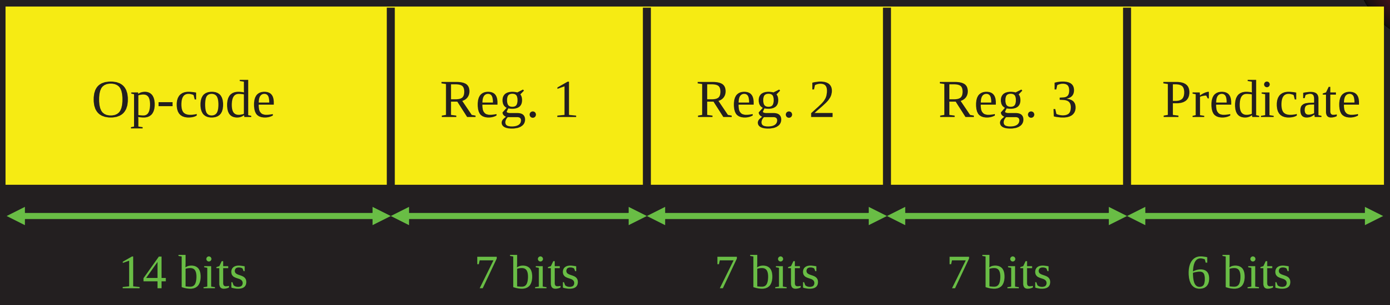


P_0	P_1	P_2	• • •	P_{62}	P_{63}

Each bit represents the result of a conditional expression evaluation.

Very Long Instruction Word — VLIW

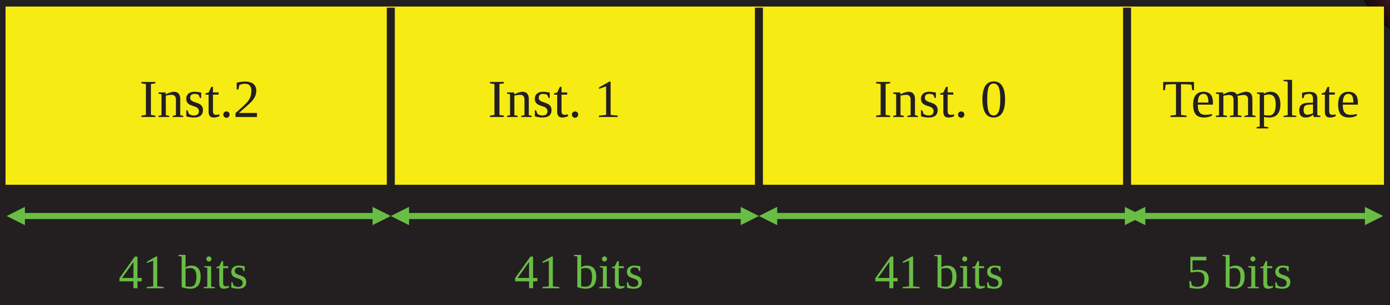
▶IA-64 – Instruction Format



Op-code	Reg. 1	Reg. 2	Reg. 3	Predicate
14 bits	7 bits	7 bits	7 bits	6 bits

Very Long Instruction Word — VLIW

- IA-64 — Instruction Format (instruction bund



Inst.2	Inst. 1	Inst. 0	Template
41 bits	41 bits	41 bits	5 bits

Template field specifies what type of Execution Units each instruction in a group requires.

Very Long Instruction Word — VLIW

- Summary
  - Compilation techniques to exploit ILP
    - Loop Unrolling
    - Trace Scheduling
    - Software Pipelining
    - Speculative scheduling
  - IA-64
    - General Configuration
    - Register sets
    - Instruction Format

Very Long Instruction Word — VLIW

- IA-64
  - Two levels of parallelisms are provided:
    - Instruction Level Parallelism — Compiler cr instruction groups (a collection of instru bundles) so that all instructions in an instru group can be executed in parallel safely.
    - Control Flow Parallelism — is provided executing compound And and Or condition parallel. This allows several multiway branch be grouped together and executed in a s instruction group.

Very Long Instruction Word — VLIW

- IA-64 — Compound Conditional Code
  - Assume the following code:

```
If ((a = 0) || ($b \leq 5$) || ($c \neq d$) || ($f > 10$))
    $r_3 = 8$;
```

Very Long Instruction Word — VLIW

IA-64 — Compound Conditional Code

- In IA-64 the aforementioned code is express

Cmp.ne $p_1 = r_0 , r_0$
Add $t = -5 , b$
Add $k = -10, f$

Cmp.eq.or $p_1 = 0 , a$
Cmp.ge.or $p_1 = 0 , t$
Cmp.ne.or $p_1 = c , d$
Cmp.gt.or $p_1 = 0 , k$

(p_1) mov $r_3 = 8$

Very Long Instruction Word — VLIW

- IA-64 — Compound Conditional Code
  - Register p_1 is initialized to false,
  - The conditions for each of the OR express
    is calculated in parallel, and
  - Final result of the p_1 is used in the
    instruction.

Very Long Instruction Word — VLIW

IA-64 — Branches

- IA-64 reduces the negative effect of branches.
- It allows the compiler to generate the cod execute instructions from multiple conditi paths at the same time.

Very Long Instruction Word — VLIW

- IA-64 – Branches
  - Assume the following code:

```
If ($r_1 = r_2$)
        $r_9 = r_{10} - r_{11}$;
else
        $r_5 = r_6 + r_7$;
```

Very Long Instruction Word — VLIW

IA-64 – Branches

- In IA-64 we have the following:

Predicate regis

$\text{Cmp.eq } p_1, p_2 = r_1 , r_2;$

$(p_1)\ \text{sub } r_9 = r_{10} , r_{11};$

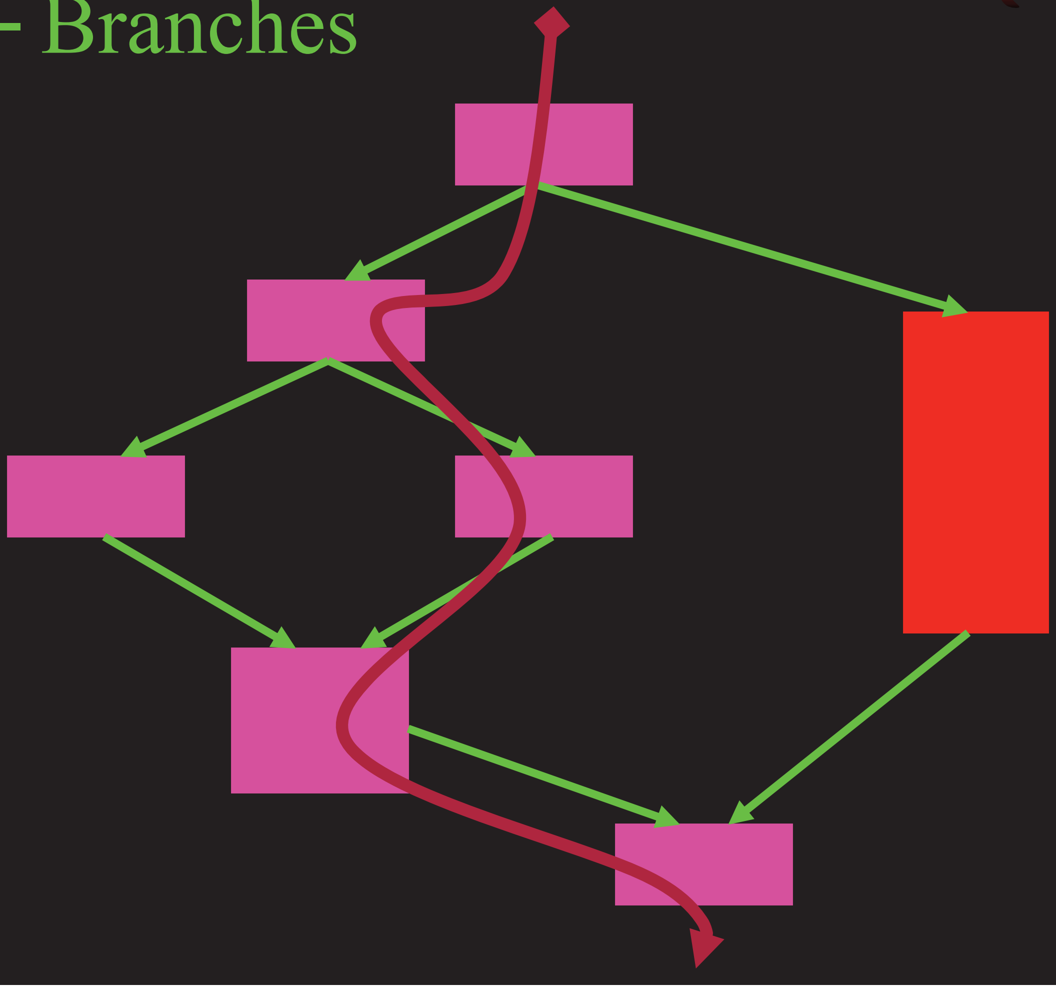
$(p_2)\ \text{add } r_5 = r_6 , r_7;$

Very Long Instruction Word — VLIW

- IA-64 — Branches
  - Ability to calculate compound conditi codes in parallel and associating a predi to each statement allows compiler to b larger basic blocks and hence to increase degree of instruction level parallelism.

Very Long Instruction Word — VLIW

- IA-64 — Branches



Very Long Instruction Word — VLIW

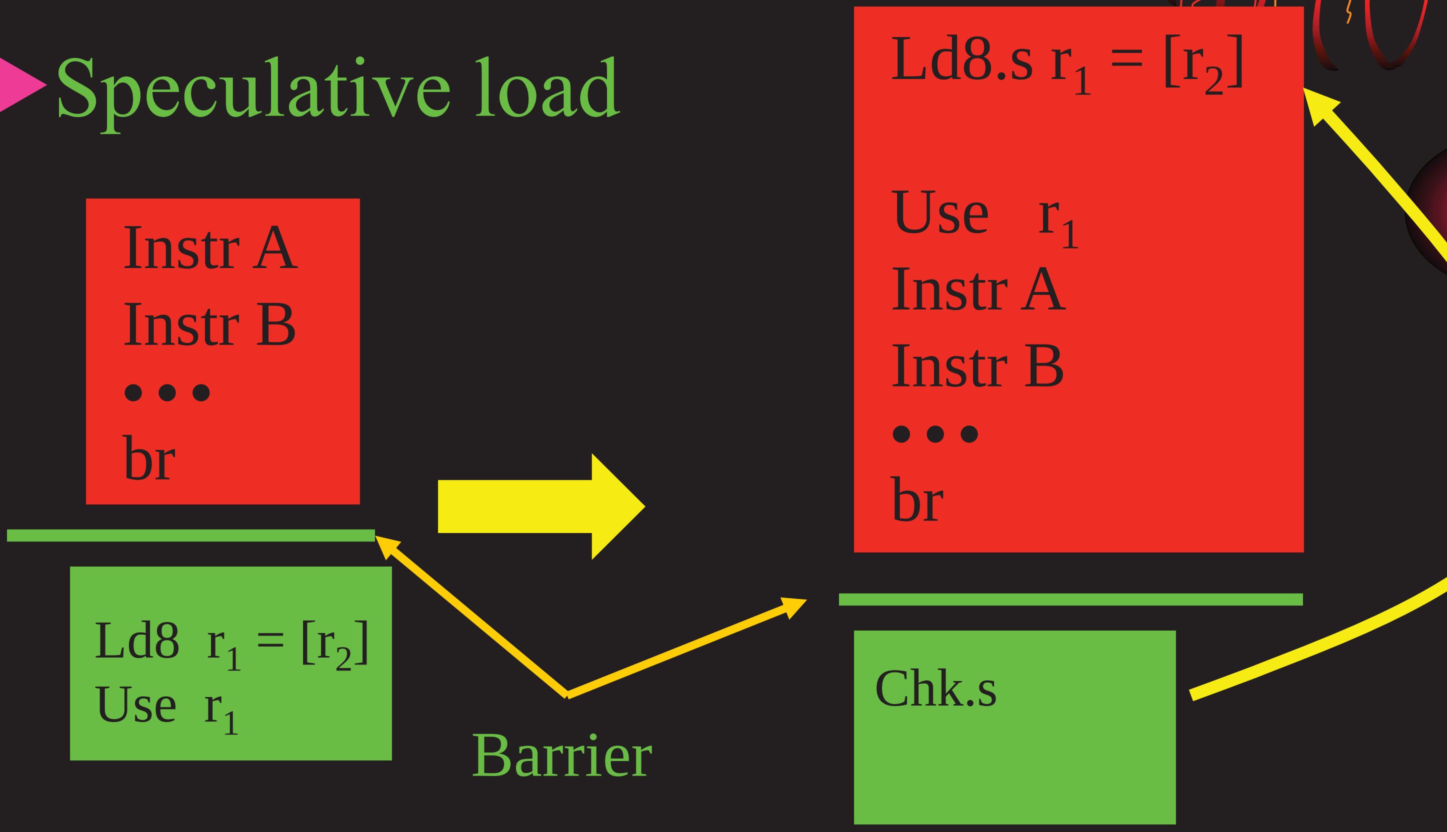
- IA-64 — Branches
  - Trace scheduling then will be app extensively to schedule traces with hi probability of execution earlier.

Very Long Instruction Word — VLIW

- Speculative load
  - Speculative load is an interesting technique allows to speculatively early start execution of critical instructions (load).
  - Load can be safely scheduled ahead of one or r branches.
  - In case of exception, flag is raised and attached t load result — At runtime, a deferred exception tok written to the target register (extra bit of register).
  - At proper moment, this flag is checked to redirec control to a fix-up code.

Very Long Instruction Word — VLIW

- Speculative load



Instr A
Instr B
• • •
br

Ld8 $r_1 = [r_2]$
Use r_1

Barrier

Ld8.s $r_1 = [r_2]$

Use r_1
Instr A
Instr B
• • •
br

Chk.s

Very Long Instruction Word — VLIW

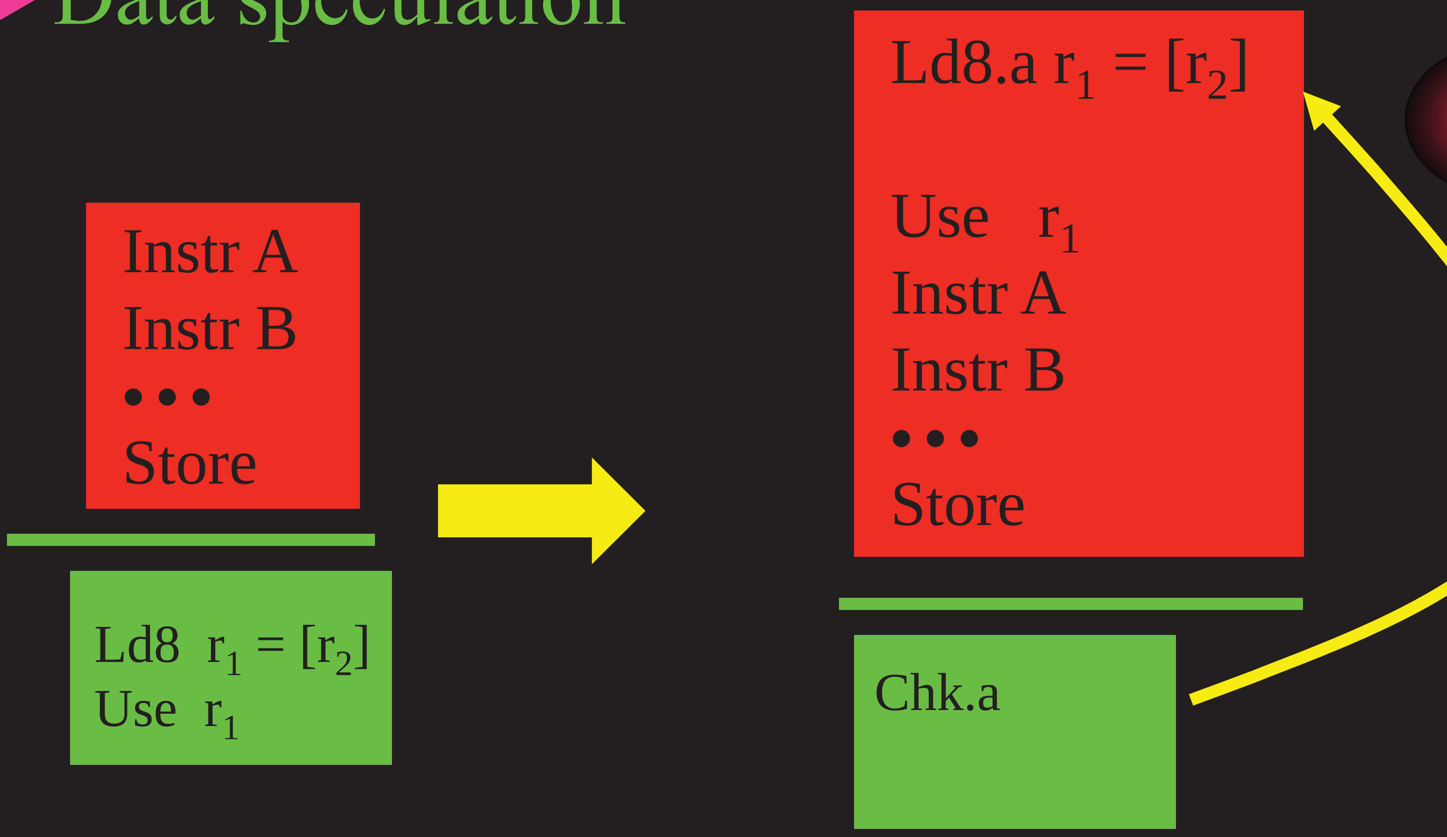
- Speculative load
  - Note that almost all instructions in IA propagate the tag on a register, as a re entire calculation chains may be sched speculatively.
  - The compiler only inserts a single *chk.s* (ch speculate) to check the result of mult speculative computations.

Very Long Instruction Word — VLIW

- Data speculation
  - IA-64 allows the compiler to schedule a before one or more prior stores — speculation.
  - Advanced load instruction (*ld.a*) along advanced load check instruction (*chk.a*) used to accomplish this.

Very Long Instruction Word — VLIW

- Data speculation



Instr A
Instr B
• • •
Store

Ld8 $r_1 = [r_2]$
Use r_1

Ld8.a $r_1 = [r_2]$

Use r_1
Instr A
Instr B
• • •
Store

Chk.a

Very Long Instruction Word — VLIW

- Data speculation
  - An advanced load is a load that has b
    speculatively moved above store instructio
    which it is potentially dependent.

Very Long Instruction Word — VLIW

- Data speculation
  - Advanced load is similar to traditional l During the run time, system rec information such as:
    - The target register,
    - Memory address accessed, and
    - Access sized

  In the advanced load address table.

Very Long Instruction Word — VLIW

- Data speculation
  - When a store is executed, an associative up against the active advanced load add table is performed. If there is a match, advanced load address entry is marked invalid (it is cleared).

Very Long Instruction Word — VLIW

- Data speculation
  - Later, when the chk.a is executed, hardv checks the advanced load address table for entry installed by its corresponding advar load.
    - If an entry is found, the speculation was succe and nothing will happen.
    - If no entry is found, there may have been a colli and the check instruction branches to a fix-up to reexecute the code.

Very Long Instruction Word — VLIW

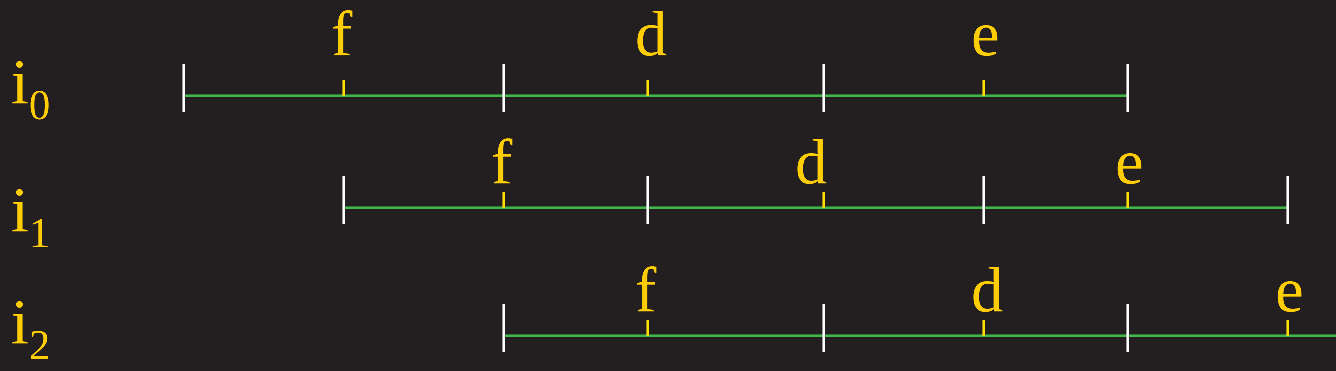
- Questions
  - Compare and contrast VLIW architec against multiprocessor and vector proce (you need to discuss about issues such as flow of control, inter-proce communications, memory organization programming requirements).
  - Within the scope of VLIW architecture, dis the major source of problems.

Very Long Instruction Word — VLIW

- VLIW machines are not software compatible with general purpose machine. Even they are compatible with themselves.
- Density of long instructions depend on the instruc level parallelism detected during the compila This could effect space utilization drastically.
- VLIW offers performance improvement.
- There is no need for extra hardware in order to de parallelism.

Super Pipelined Processor

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



2-Stage Super Pipelined Processor

Super Pipelined Processor

- Naturally, in a super Pipelined Proces sub-stages are clocked at a higher freque than the major stages.
- Reducing processor cycle time, he higher performance, relies on instruct parallelism to prevent pipeline stalls in sub-stages.

Super Pipelined Processor

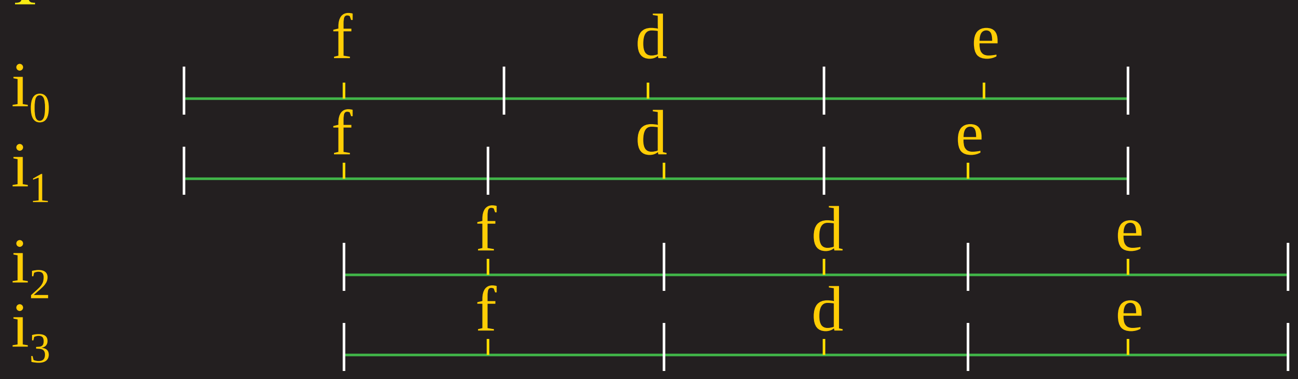
- In comparison with Super Scalar:
  - For a given set of operations, the su pipelined processor takes longer to generat results than the super scalar processor.
  - Simple operations take longer time to exe in a super scalar than super pipelined, s there are no clock with finer resolution.

Super Pipelined Processor

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

Super Pipelined Superscalar Process

- Since the number of instructions issued cycle and the cycle time are theoretic orthogonal, we could have a super pipeli superscalar machine.



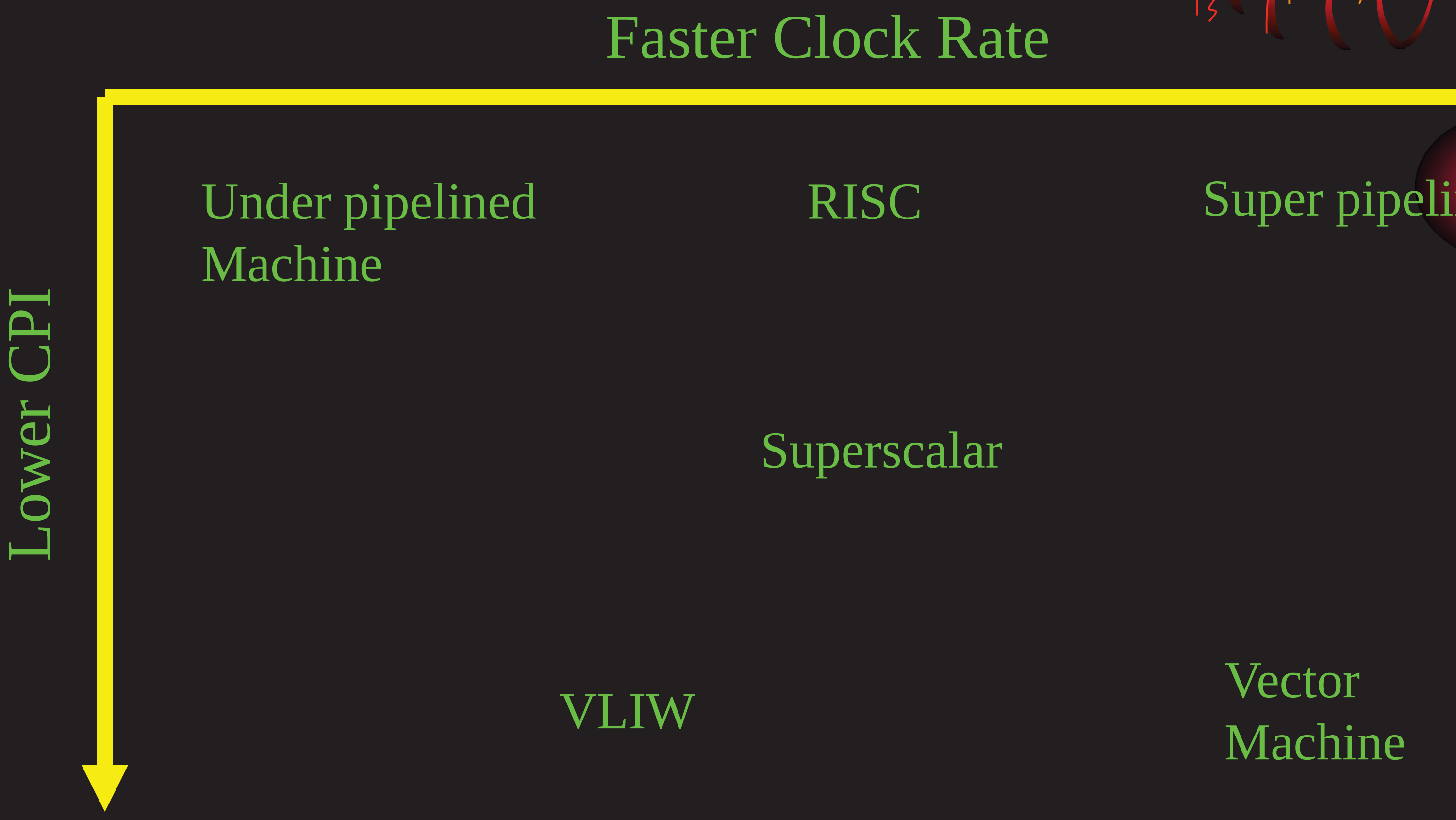
2-Stage 2-issue Super Pipelined Superscalar Proc

Fall 2010

Beyond RISC

- As noted before, achieving a higher performa means processing a given task in a smaller amo of time. To reduce the time to execute a seque of instructions, one can:
  - Reduce individual instruction latencies, or
  - Execute more instructions concurrently.
- Superscalar processors exploit the sec alternative.

Beyond RISC



Faster Clock Rate

Lower CPI

Under pipelined Machine

RISC

Super pipeli

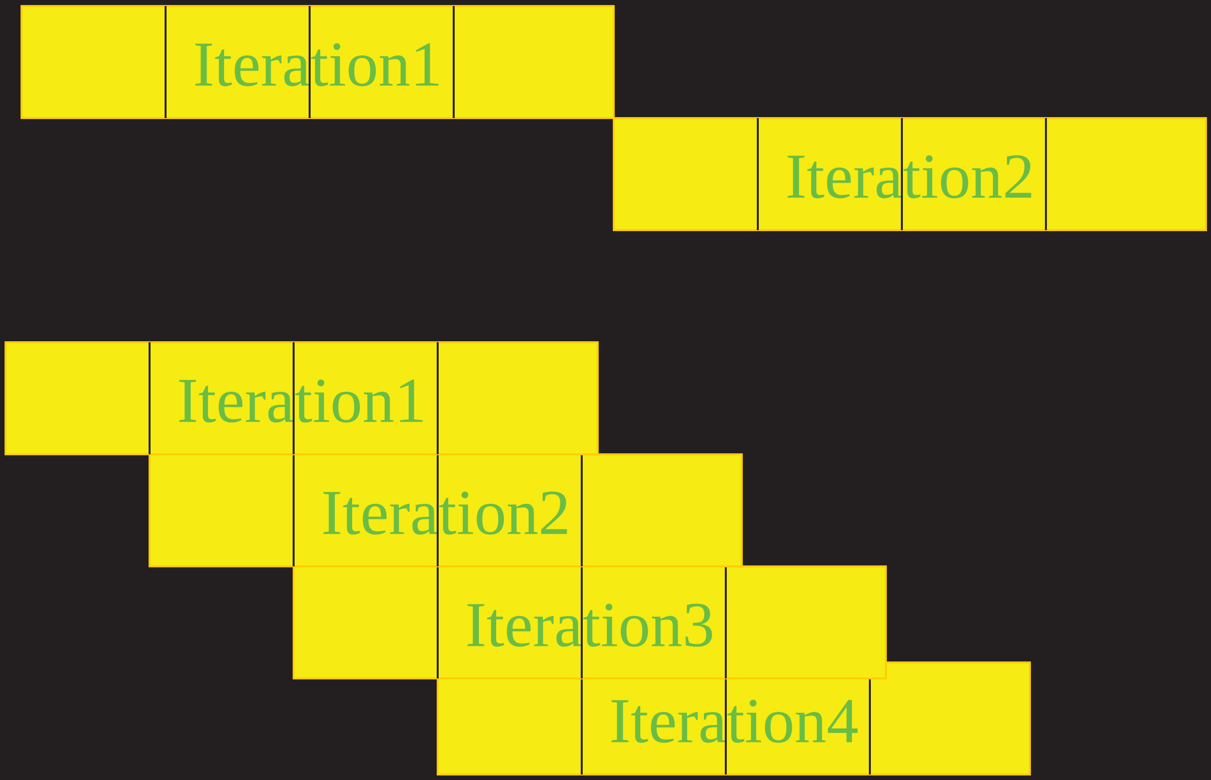
Superscalar

VLIW

Vector Machine

Beyond RISC

- Software pipelining



Iteration1

Iteration2

Iteration1

Iteration2

Iteration3

Iteration4

Beyond RISC

- Software pipelining
  - Software pipelining requires
    - Managing the loop count,
    - Handling renaming the registers for the pipeline
    - Finishing the work in progress when the loo ended.
    - Starting the pipeline when the loop is entered, ar
    - Unrolling to expose cross-iteration parallelism.

Beyond RISC

- Software pipelining
  - Software pipelining is a technique
    reorganizes loops such that each iteration in
    software-pipelined code is made f
    instructions chosen from different iteration
    the original loop — it interleaves instruct
    from different iterations without unrolling
    loop.

Beyond RISC

- Software Pipelining

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
BNZ	R_1, Loop	Branch if not zero

Beyond RISC

Software Pipelining

Iteration i:	LD	F_0, 0(R_1)
	ADD	F_4, F_0, F_2
	SD	F_4, 0(R_1)
Iteration i+1	LD	F_0, 0(R_1)
	ADD	F_4, F_0, F_2
	SD	F_4, 0(R_1)
Iteration i+2	LD	F_0, 0(R_1)
	ADD	F_4, F_0, F_2
	SD	F_4, 0(R_1)

Beyond RISC

Software Pipelining

LOOP:

SD	F_4, 16(R_1)	Stores into M[i]
ADD	F_4, F_0, F_2	Adds to M[i-1]
LD	F_0, 0(R_1)	Loads M[i-2]
SUB	R_1, R_1, #8	Decrement by 8
BNZ	R_1, Loop	Branch if not zero