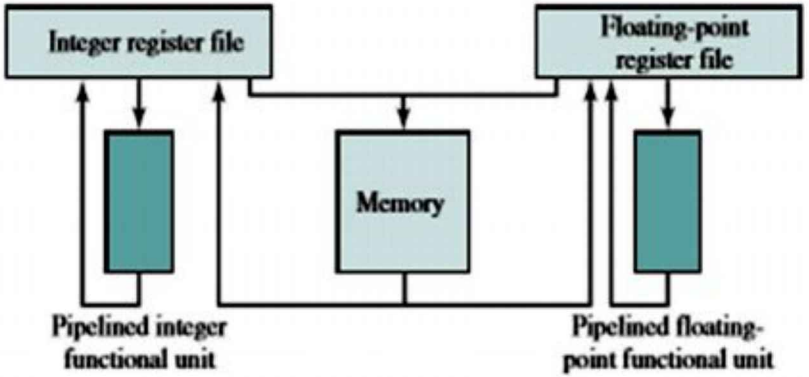
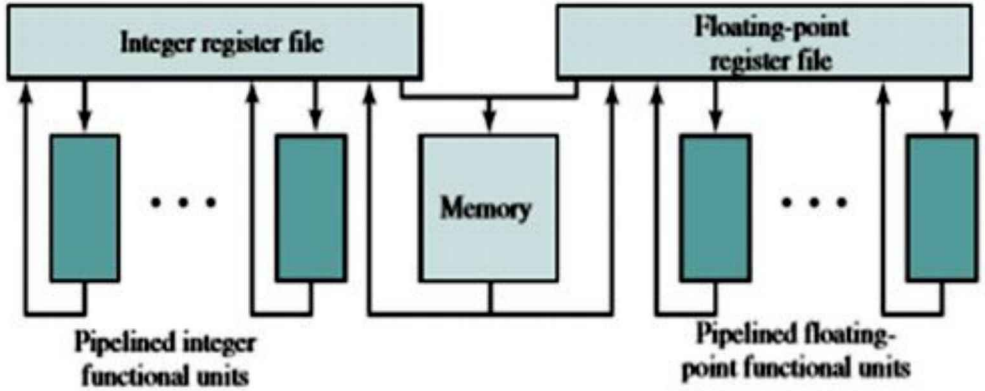
Superscalar Processors

- Superscalar is a computer designed to improve the performance of the execution of scalar instructions.
- A scalar is a variable that can hold only one atomic value at a time, e.g., an integer or a real (float).
- A scalar architecture processes one data item at a time.
- Examples of non-scalar variables:
  - Arrays
  - Matrices
  - Records
- In a superscalar architecture (SSA), several scalar instructions can be initiated simultaneously and executed independently.
- Pipelining allows also several instructions to be executed at the same time, but they have to be in different pipeline stages at a given moment.
- SSA includes all features of pipelining but, in addition, there can be several instructions executing simultaneously in the same pipeline stage.
- SSA introduces therefore a new level of parallelism, called **instruction-level parallelism**.

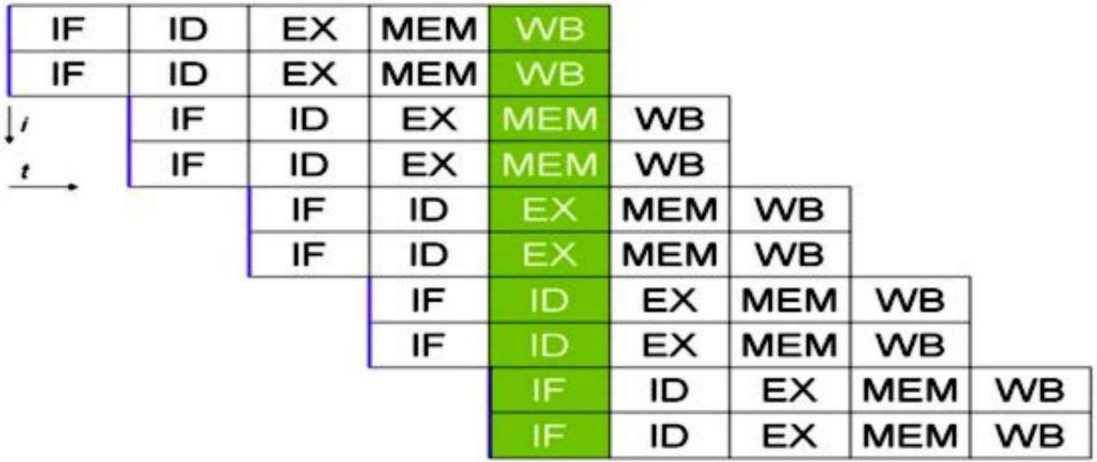


Scalar Organization



Superscalar Organization

Instruction-level parallelism



Superscalar machine

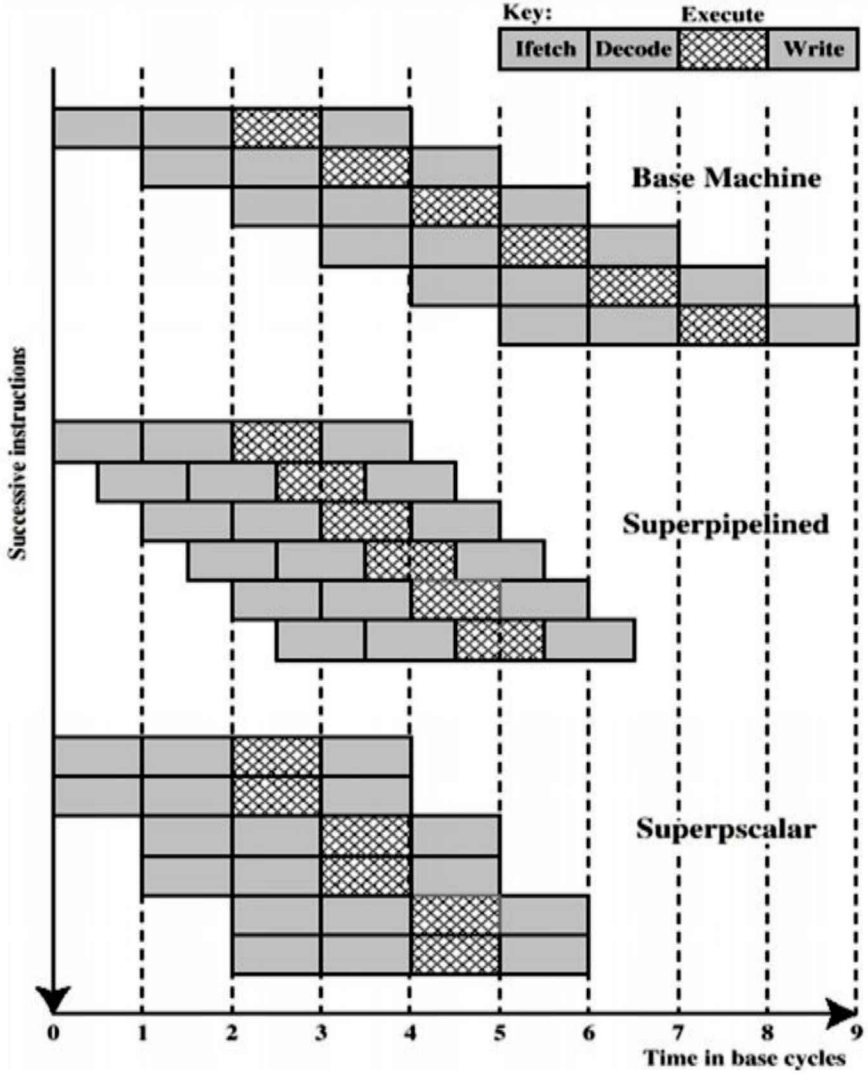
- A Superscalar machine executes multiple independent instructions in parallel.
- They are pipelined as well:
  - “Common” instructions (arithmetic, load/store, conditional branch) can be executed independently.
  - Equally applicable to RISC & CISC, but more straightforward in RISC machines.
  - The order of execution is usually assisted by the compiler.
- Limitations of Superscalar:
  - Data dependency
  - Procedural dependency
  - Resource conflicts

Superpipelining

- Superpipelining is based on dividing the stages of a pipeline into several sub-stages, and thus increasing the number of instructions which are handled by the pipeline at the same time.
- For example, by dividing each stage into two sub-stages, a pipeline can perform at twice the speed in the ideal situation.
- Many pipeline stages may perform tasks that require less than half a clock cycle.
- No duplication of hardware is needed for these stages.
- For a given architecture and the corresponding instruction set there is an optimal number of pipeline stages/sub-stages. Increasing the number of stages/sub-stages over this limit reduces the overall performance.
    - Overhead of data buffering between the stages.
    - Not all stages can be divided into (equal-length) sub-stages.
    - The hazards will be more difficult to resolved.
    - More complex hardware.

Superscalar vs. Superpipelining

- Base machine: 4-stage pipeline
  - Instruction fetch
  - Operation decode
  - Operation execution
  - Result write back
- **Superpipeline** of degree 2
  - A sub-stage often takes half a clock cycle to finish.
- **Superscalar** of degree 2
  - Two instructions are executed concurrently in each pipeline stage.
  - Duplication of hardware is required by definition.



Instruction Level Parallelism

- Instruction Level Parallelism (ILP) is used to refer to the architecture in which multiple operations can be performed parallelly in a particular process, with its own set of resources – address space, registers, identifiers, state, program counters.
- It refers to the compiler design techniques and processors designed to execute operations, like memory load and store, integer addition, float multiplication, in parallel to improve the performance of the processors.
- Examples of architectures that exploit ILP are VLIWs, Superscalar Architecture.
- A typical ILP allows multiple-cycle operations to be pipelined.
- *Note: Here again we have hazards like: Data dependencies, Name Dependencies, Data Hazards (RAW, WAW, WAR) and Control Dependencies.*

Example of ILP

- Suppose, 4 operations can be carried out in single clock cycle.
- So there will be 4 functional units, each attached to one of the operations, branch unit, and common register file in the ILP execution hardware.
- The sub-operations that can be performed by the functional units are Integer ALU, Integer Multiplication, Floating Point Operations, Load, Store.
- Let the respective latencies be 1, 2, 3, 2.
- Let the sequence of instructions be:
  - y1 = x1*1010
  - y2 = x2*1100
  - z1 = y1+0010
  - z2 = y2+0101
  - t1 = t1+1
  - p = q*1000
  - clr = clr+0010
  - r = r+0001

CYCLE	INT ALU	INT ALU	FLOAT ALU	FLOAT ALU
1	t1 = t1+1	clr = clr+0010	y1 = x1*1010	y2 = x2*1100
2	r = r+0001		p = q*1000	
3	nop			
4	z1 = y1+0010	z2 = y2+0101		

CYCLE	OPERATION
1	y1 = x1*1010
2	nop
3	nop
4	y2 = x2*1100
5	nop
6	nop
7	z1 = y1+0010
8	z2 = y2+0101
9	t1 = t1+1
10	p = q*1000
11	clr = clr+0010
12	r = r+0001

Implementation of ILP and overcoming hazards or dependencies

- Score boarding
- Tomasulo's solution for dynamic scheduling.
- Branch prediction.

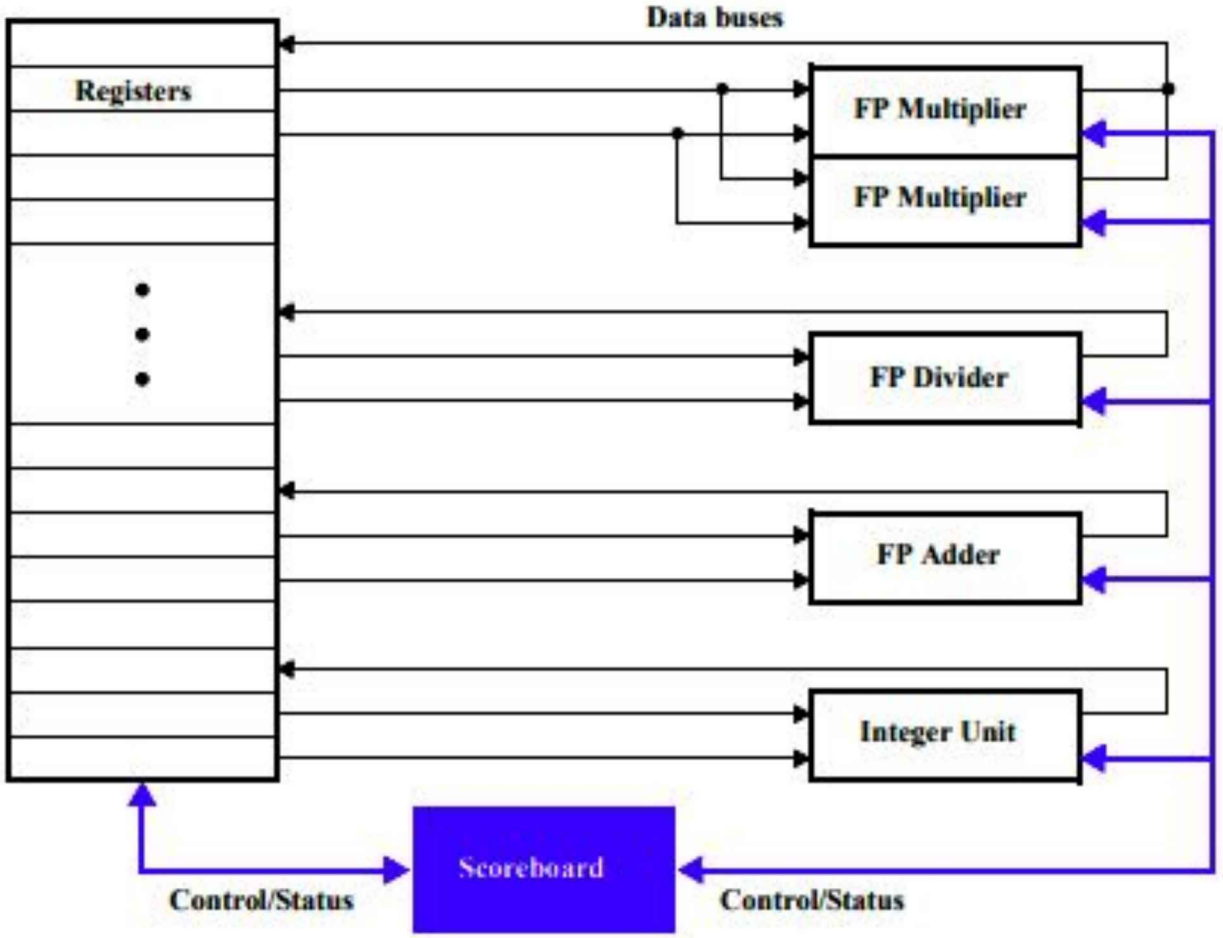
Score Boarding

- Instructions to be issued when they are ready, not necessarily in order, hence out of order execution.
- To implement out-of-order issue we need to split the instruction decode phase into two:
  - **Issue**—decode instructions and check for structural hazards;
  - **Read operands**—wait until no data hazards obtain then read the operands and start executing.
- It dynamically schedules the pipeline.
- Instructions must pass through the issue phase in order.
- This method can stall or bypass each other, in the read operands phase and enter, or even, complete execution in out of order manner.

Four Stages in Scoreboarding

- **Issue: resolve structural and WAW hazards**
  - The functional unit is free (structural hazards)
  - No other active instructions have the same destination register (WAW hazards)
- **Read operands: resolve RAW hazards**
  - Check if the source operands are available
  - If no earlier issued active instruction is going to write
  - If no functional unit is writing
- **Execution complete**
- **Write result: resolve WAR hazards**
  - if there is an instruction that has not read its operand that precedes the completing instruction
  - one of the operands is the same reg as the result of the completing instruction.

Basic Structure of Scoreboard



Three Parts of the Scoreboard

- **Instruction status—**which of 4 steps the instruction is in
- **Functional unit status—**Indicates the state of the functional unit (FU). 9 fields for each functional unit:
  - Busy—Indicates whether the unit is busy or not
  - Op—Operation to perform in the unit (e.g., + or –)
  - Fi—Destination register
  - Fj, Fk—Source-register numbers
  - Qj, Qk—Functional units producing source registers Fj, Fk
  - Rj, Rk—FIags indicating when Fj, Fk are ready
- **Register result status—**Indicates which functional unit will write each register, if one exists. Blank when no pending instructions will write that register

Scoreboard

Instruction	Instruction Status			
	Issue	Read Operands	Execution Complete	Write Result
LOAD F6,34(R2)	x	x	x	x
LOAD F2,45(R3)	x	x	x	
MULT F0,F2,F4	x			
SUB F8,F6,F2	x			
DIV F10,F0,F6	x			
ADD F6,F8,F2	x			

Name	Functional Unit Status								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2	R3				No	
Multiplier 1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Multiplier 2	No								
Adder	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divider	Yes	Div	F10	F0	F6	Mult1		No	Yes

Name	Register Result Status								
	F0	F2	F4	F6	F8	F10	F12	...	F30
Integer	Mult1	Integer			Sub	Divide			

Scoreboard Limitations

- No forwarding (First write register then read it)
- Limited to instructions in basic block (small window)
- Number of functional units(structural hazards)
- Wait for WAR hazards
- Prevent WAW hazards

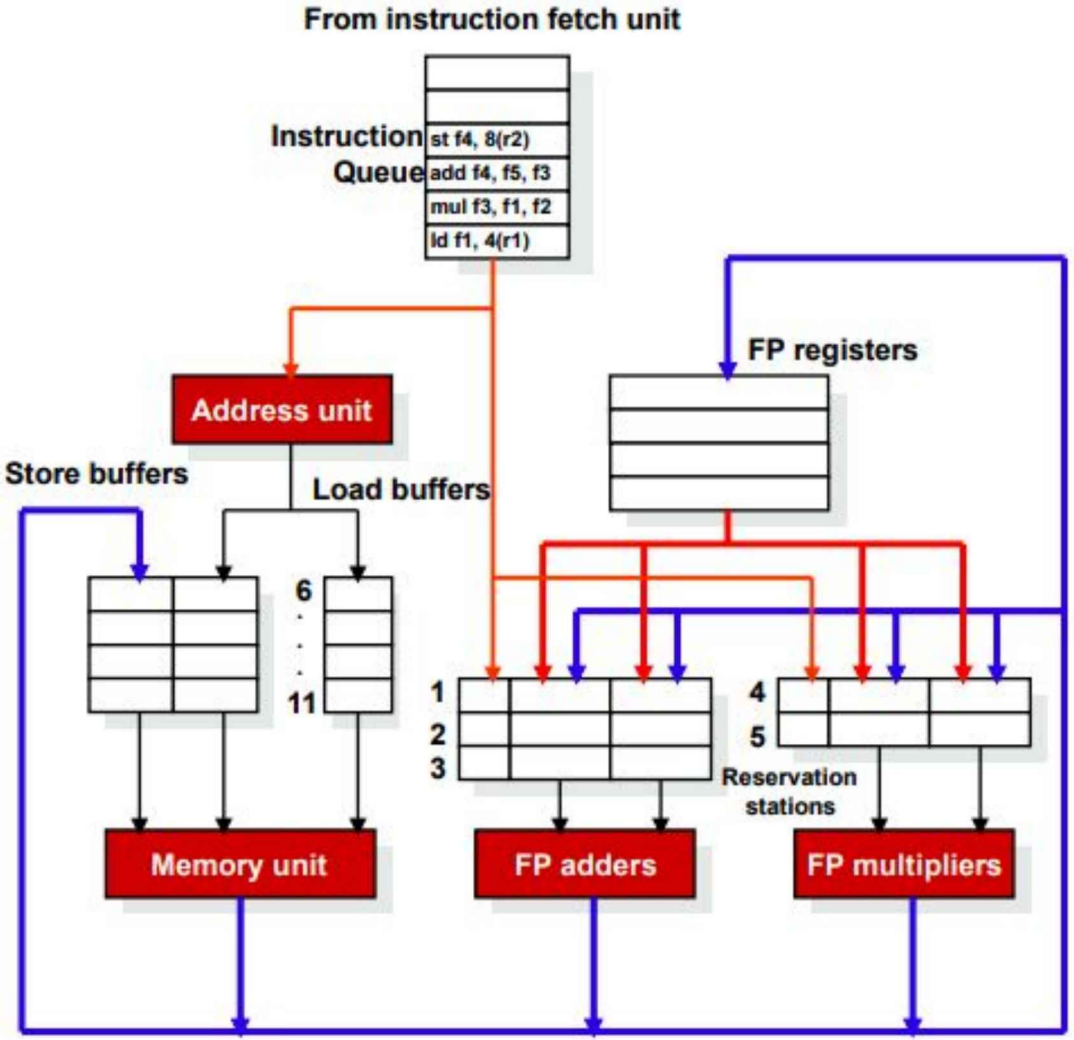
Tomasulo's solution for dynamic scheduling

- Executing instructions only when operands are available, waiting instruction is stored in a **reservation station**.
- Reservation stations keep track of pending instructions (RAW). WAW can be avoided using **Register renaming**.
- Tomasulo architecture executes instructions in three phases; each phase may take more than one clock cycle:
  - **Issue**
    - Get next instruction from FIFO queue
    - If available RS, issue the instruction to the RS with operand values if available
    - If operand values not available, stall the instruction
  - **Execute**
    - When operand becomes available, store it in any reservation stations waiting for it
    - When all operands are ready, issue the instruction
    - Loads and store maintained in program order through effective address
    - No instruction allowed to initiate execution until all branches that proceed it in program order have completed
  - **Write result**
    - Write result on CDB (common data bus) into reservation stations and store buffers
    - (Stores must wait until address and value are received)

Reservation Station Components

- **Op—**Operation to perform in the unit (e.g., + or –)
- **Qj, Qk—**Reservation stations producing source registers (value to be written)
- **Vj, Vk—**Value of Source operands
- **Rj, Rk—**Flags indicating when Vj, Vk are ready
- **Busy—**Indicates reservation station and FU is busy
- **Register result status—**Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Tomasulo Organization



Example

- LDs: 2 cycles; ADDs and SUBDs: 2 cycles; MULTDs: 10 cycles; DIVDs: 40 cycles.
- Tomasulo at Cycle 0:

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU									

Example Contd…

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU	Load1								

Example Contd…

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1		
LD	F2	45+	R3	2		
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU		Load2		Load1					

Example Contd…

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	
LD	F2	45+	R3	2		
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1					

Example Contd…

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	Yes	SUBD	M(A1)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	…	*F30*
4	*FU*	Mult1	Load2		M(A1)	Add1				

Example Contd…

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
2	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	…	F30
5	FU	Mult1	M(A2)		M(A1)	Add1	Mult2			

Example Contd…

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
1	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
9	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	M(A2)		Add2	Add1	Mult2			

Example Contd…

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
0	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	M(A2)		Add2	Add1	Mult2			

Example Contd…

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
2	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1	M(A2)		Add2	(M-M)	Mult2			

Example Contd…

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
1	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
6	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1	M(A2)		Add2	(M-M)	Mult2			

Tomasulo Summary

- Prevents Register as bottleneck
- Avoids WAR, WAW hazards of Scoreboard
- Allows loop unrolling in HW
- Not limited to basic blocks (provided branch prediction)
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- 360/91 descendants are PowerPC 604, 620; MIPS R10000; HP-PA 8000; Intel Pentium Pro

Branch Prediction

- It uses predictor is a simple saturating n-bit counter.
- Each time a particular branch is taken its entry is incremented otherwise it is decremented.
- If the most significant bit in the counter is set then predict that the branch is taken.
- Instructions executed on the basis of a predicted branch, before the actual branch result is known, are said to involve **Speculative** execution.
  - If a branch prediction tums out to be correct, the corresponding speculatively executed instructions must be committed.
  - If the prediction tums out to be wrong, the effects of corresponding speculative operations carried out within the processor must be cleaned up, and instructions from another branch of the program must instead be executed.

Loop-Unrolling: Compiler-Detected Instruction Level Parallelism

- Loop unrolling is a technique by which independent instructions from multiple successive iterations of a loop can be made to execute in parallel.
- Example:

```
for i = 0 to 58 do
    c[i] = a[i]*b[i] - p*d[i];
```

```
for j = 0 to 52 step 4 do
{
  c[j] = a[j]*b[j] - p*d[j];
  c[j+1] = a[j+1]*b[j+1] - p*d[j+1];
  c[j+2] = a[j+2]*b[j+2] - p*d[j+2];
  c[j+3] = a[j+3]*b[j+3] - p*d[j+3];
}
```

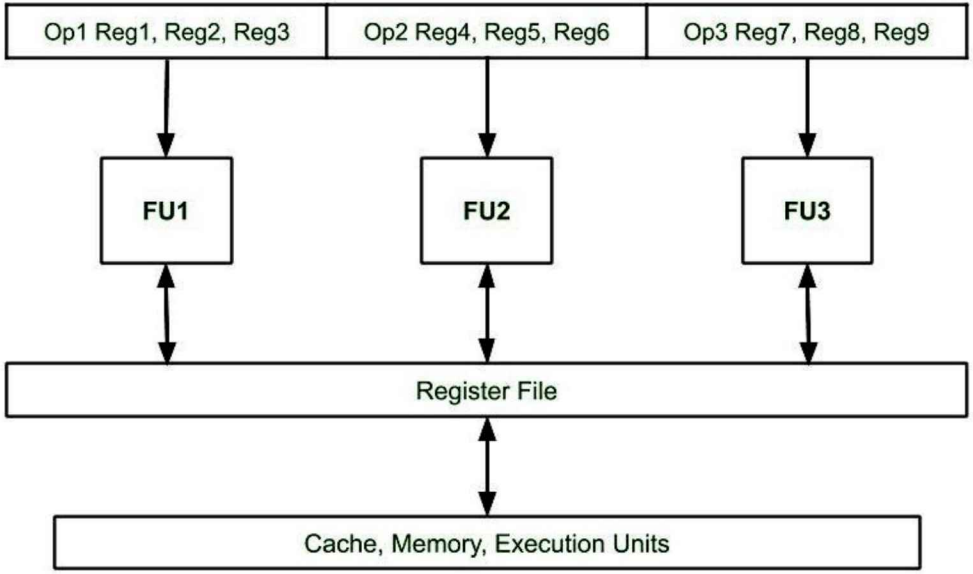
- While implementing Loop unrolling, control variable and any dependencies have to checked.

VLIW processors

- The limitations of the Superscalar processor are prominent as the difficulty of scheduling instruction becomes complex.
- The intrinsic parallelism in the instruction stream, complexity, cost, and the branch instruction issue get resolved by a higher instruction set architecture called the **Very Long Instruction Word (VLIW) or VLIW Machines**.
- VLIW uses Instruction Level Parallelism, i.e. it has programs to control the parallel execution of the instructions.
- In other architectures, the performance of the processor is improved by using either of the following methods: pipelining (break the instruction into subparts), superscalar processor (independently execute the instructions in different parts of the processor), out-of-order-execution (execute orders differently to the program) but each of these methods add to the complexity of the hardware very much.
- VLIW Architecture deals with it by depending on the compiler. The programs decide the parallel flow of the instructions and to resolve conflicts.
- This increases compiler complexity but decreases hardware complexity by a lot.

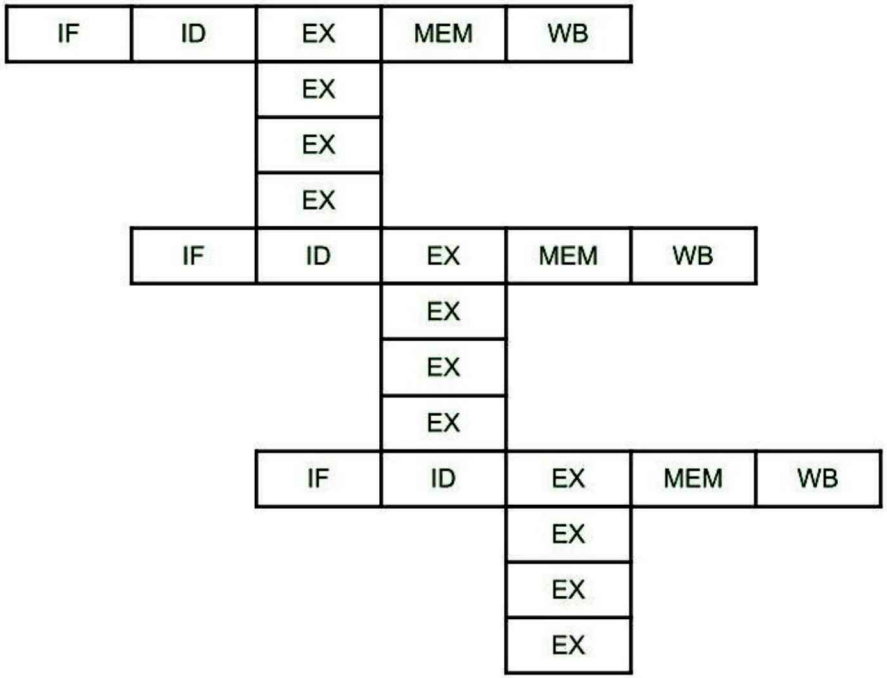
VLIW processors Features

- The processors in this architecture have multiple functional units, fetch from the Instruction cache that have the Very Long Instruction Word.
- Multiple independent operations are grouped together in a single VLIW Instruction. They are initialized in the same clock cycle.
- Each operation is assigned an independent functional unit.
- All the functional units share a common register file.
- Instruction words are typically of the length 64-1024 bits depending on the number of execution unit and the code length required to control each unit.
- Instruction scheduling and parallel dispatch of the word is done statically by the compiler.
- The compiler checks for dependencies before scheduling parallel execution of the instructions.



Block Diagram of VLIW Architecture

Time Space Diagram of VLIW Processor where 4 instructions are executed in parallel in a single instruction word



IF: Instruction Fetch ID: Instruction Decode EX: Execute MEM: Memory WB: Write Back

VLIW processors

- **Advantages:**
  - Reduces hardware complexity.
  - Reduces power consumption because of reduction of hardware complexity.
  - Since compiler takes care of data dependency check, decoding, instruction issues, it becomes a lot simpler.
  - Increases potential clock rate.
  - Functional units are positioned corresponding to the instruction pocket by compiler.
- **Disadvantages :**
  - Complex compilers are required which are hard to design.
  - Increased program code size.
  - Larger memory bandwidth and register-file bandwidth.
  - Unscheduled events, for example a cache miss could lead to a stall which will stall the entire processor.
  - In case of un-filled opcodes in a VLIW, there is waste of memory space and instruction bandwidth.

What is Vector(Array) Processing?

- There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items, that will take a conventional computer (with scalar processor) days or even weeks to complete.
- Such complex instructions, which operates on multiple data at the same time, requires a better way of instruction execution, which was achieved by Vector processors.
- A vector instruction involves a large array of operands.
- In other words, the same operation will be performed over an array or a string of data. Specialized vector processors are generally used in supercomputers.
- A vector processor can assume either a register-to-register architecture (uses shorter instructions and vector register files) or a memory-to-memory architecture (memory-based instructions which are longer in length, including memory addresses).

Challenges in parallel processing

- Connecting your CPUs
  - Dynamic vs Static—connections can change from one communication to next
  - Blocking vs Nonblocking—can simultaneous connections be present?
  - Connections can be complete, linear, star, grid, tree, hypercube, etc.
- Bus-based routing
  - Crossbar switching—impractical for all but the most expensive supercomputers
  - 2X2 switch—can route inputs to different destinations
- Dealing with memory
  - Various options:
    - Global Shared Memory
    - Distributed Shared Memory
    - Global shared memory with separate cache for processors
- Potential Hazards:
  - Individual CPU caches or memories can become out of synch with each other. “Cache Coherence”
  - Solutions:
    - UMA/NUMA machines
    - Snoopy cache controllers
    - Write-through protocols

Topics discussed in Unit I and II

- Delays in pipelining
- Mechanisms to tackle pipeline stalls
- Multiprocessor architectures: shared memory symmetric multiprocessing, clusters and grids.
- Interconnection networks: characteristics and routing mechanisms.
- Parallel algorithms on realistic architectures.