Concepts Introduced

- exceptions in a pipeline
- multicycle operations
- multiple issue pipelines

Exceptions

- An *exception* or an *interrupt* is an event other than regular transfers of control (branches, jumps, calls, returns) that changes the normal flow of instruction execution.
- An *exception* refers to any unexpected change in control flow without distinguishing if the cause is internal or external.
- An *interrupt* means that the event is externally caused.

type of event	from where?	MIPS terminology
I/O device request	external	interrupt
syscall	internal	exception
arithmetic overflow	internal	exception
page fault	internal	exception
undefined instruction	internal	exception
hardware malfunction	either	exception or interrupt

Multiple Exceptions

- Exceptions can occur in different pipeline stages on different instructions.
- Multiple exceptions can occur in the same clock cycle. The LW could have a page fault in the MEM stage and the ADD could have an integer overflow in the EX stage (both in cycle 4).
- Exceptions could occur out of order. The AND could have a page fault in the IF stage (cycle 3) and the LW could have a page fault in the MEM stage (cycle 4).

cycle	1	2	3	4	5	6	7	8
LW	IF	ID	EX	MEM	WB			
ADD		IF	ID	EX	MEM	WB		
AND			IF	ID	EX	MEM	WB	
SUB				IF	ID	EX	MEM	WB

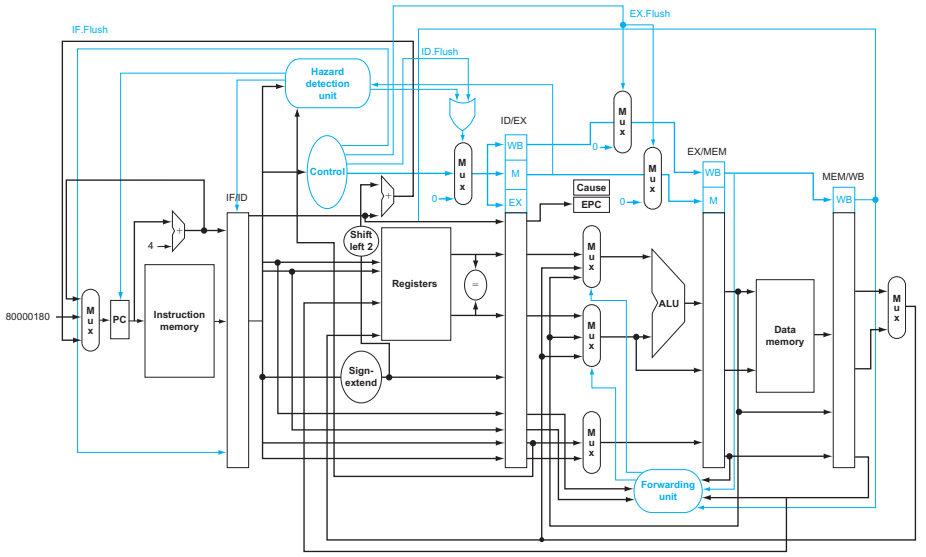
Precise Exceptions

- Supporting precise exceptions means that:
  - The exception addressed first is the one associated with the instruction that entered the pipeline first.
  - The instructions that entered the pipeline previously are allowed to complete.
  - The instruction with the exception and the ones that entered the pipeline afterwards are flushed.
  - The appropriate instruction can be restarted after the exception is handled or the program can be terminated.

Handling Exceptions

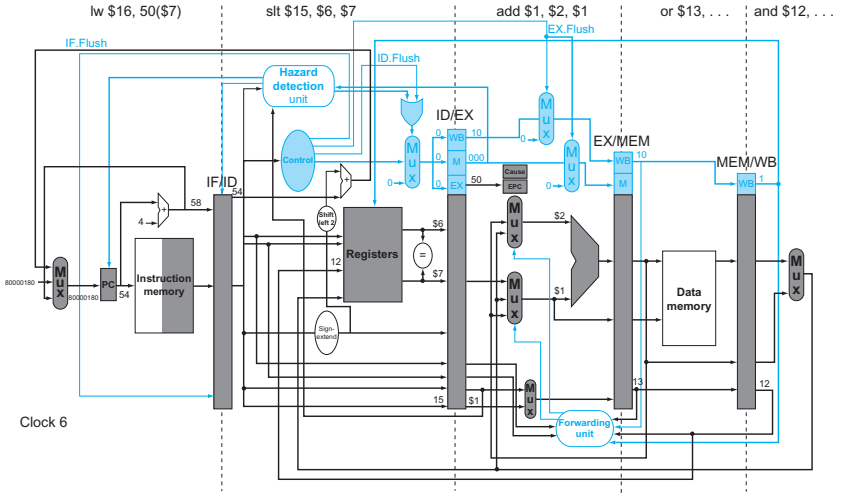
- When an exception is detected, the machine:
  - Flushes the instructions from the pipeline that include the instruction causing the exception and the ones that entered the pipeline afterwards.
  - Stores the address of the instruction causing the exception in the EPC (Exception Program Counter).
  - Begins fetching instructions at the address of the exception handler routine.

Datapath with Control to Handle Exceptions



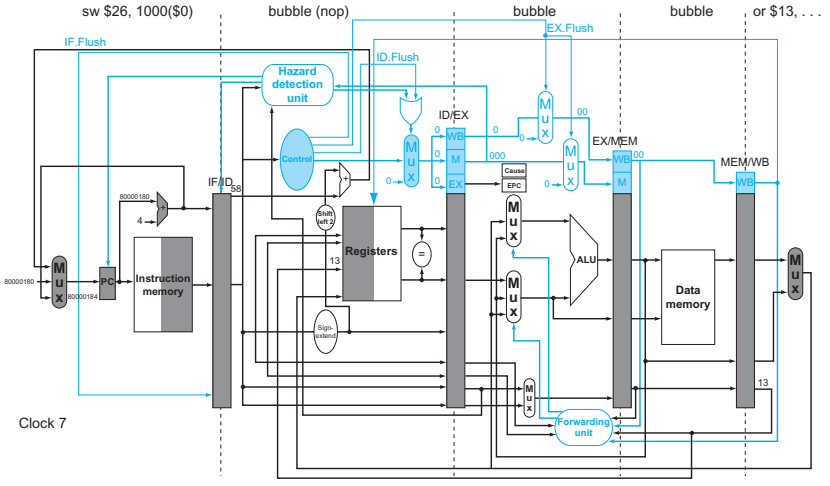
Handling an Arithmetic Exception in the Pipeline

- The address after the add is saved in the EPC and flush signals cause control values in the pipeline registers to be cleared.



Handling an Arithmetic Exception in the Pipeline (cont.)

- Instructions are now converted to bubbles in the pipeline.

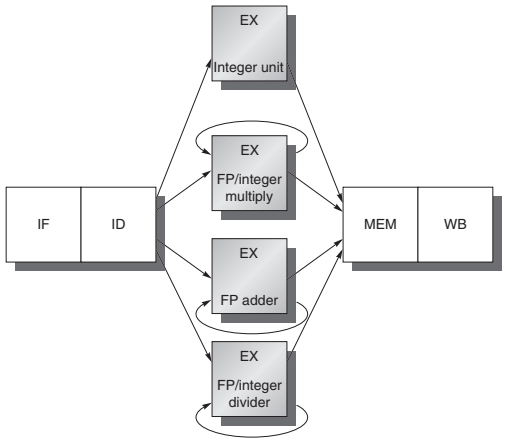


Multiple Cycle Operations

- Many arithmetic operations are traditionally performed in multiple cycles.
  - integer and floating-point multiplies
  - integer and floating-point divides
  - floating-point adds, subtracts, and conversions
- Completing these operations in a single cycle would require a longer clock cycle and/or much more logic in the units that perform these operations.

MIPS Pipeline with Three Additional FP Units

- In this datapath the multicycle operations loop when they reach the EX stage as these multicycle units are not pipelined.
- Unpipelined multicycle units can lead to structural hazards.



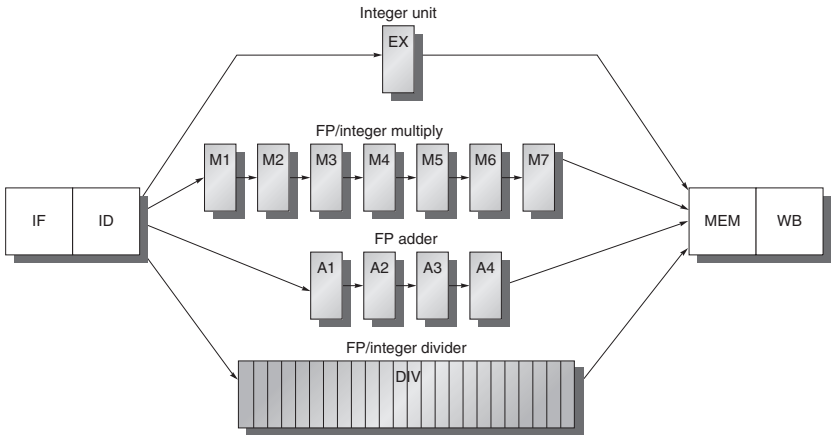
Example Functional Unit Latencies and Initiation Intervals

- The latency is the minimum number of intervening cycles between an instruction that produces a result and an instruction that uses the result.
- The initiation interval is the number of cycles that must elapse between issuing two operations of a given type.

Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

A Pipeline Supporting Multiple Outstanding FP Operations

- The multiplies, FP adds, and FP subtracts are pipelined.
- Divides are not pipelined since this operation is used less often.



Example Pipelining of Independent Multicycle Operations

- There are no dependences, so there are no stalls.
- The states in *italics* show where data is needed and the stages in **bold** show where data is available.

MUL.D	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADD.D		IF	ID	*A1*	A2	A3	**A4**	**MEM**	WB		
L.D			IF	ID	*EX*	**MEM**	WB				
S.D				IF	ID	*EX*	*MEM*	WB			

Complications Due to Multiple Cycle Operations

- Stalls for RAW hazards will be more frequent.
- The longer the pipeline, the more complicated the stall and forwarding logic becomes.
- Structural hazards can occur when multicycle operations are not fully pipelined.
- Multiple instructions can attempt to write to the FP register file in a single cycle.
- WAW hazards are possible since instructions may not reach the WB stage in order.
- Out of order completion may cause problems with exceptions.

FP Code Sequence Showing Stalls from RAW Hazards

- The multiply is stalled due to a load delay.
- The add and store are stalled due to RAW FP hazards.

		Clock cycle number																
Instruction		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
L.D	F4,0(R2)	IF	ID	EX	MEM	WB												
MUL.D	F0,F4,F6		IF	ID	Stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
ADD.D	F2,F0,F8			IF	Stall	ID	Stall	Stall	Stall	Stall	Stall	Stall	A1	A2	A3	A4	MEM	WB
S.D	F2,0(R2)					IF	Stall	Stall	Stall	Stall	Stall	Stall	ID	EX	Stall	Stall	Stall	MEM

Multicycle Instructions Can Lead to WAW Hazards

- In this example three instructions attempt to simultaneously perform a write-back to the FP register file in clock cycle 11, which causes a WAW hazard due to a single FP register file write port.
- Out of order completion can also lead to imprecise exceptions.

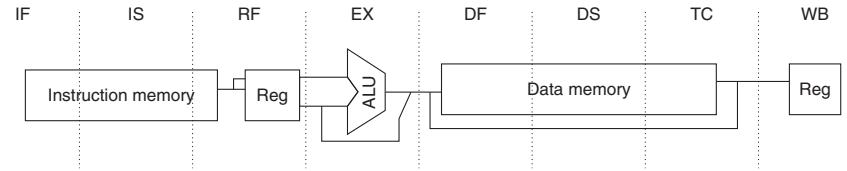
	Clock cycle number										
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
L.D F2,0(R2)							IF	ID	EX	MEM	WB

Faster Scalar Processors

- superpipelining
  - Means more stages in the pipeline.
  - Lowers the cycle time.
  - Increases the number of pipeline stalls.
- multiple issue
  - Means multiple instructions can simultaneously enter the pipeline and advance to each stage during each cycle.
  - Lowers the cycles per instruction (CPI).
  - Increases the number of pipeline stalls.
- dynamic scheduling
  - Allows instructions to be executed out of order when instructions that previously entered the pipeline are stalled or require additional cycles.
  - Allows for useful work during some instruction stalls.
  - Often increases cycle time and energy usage.

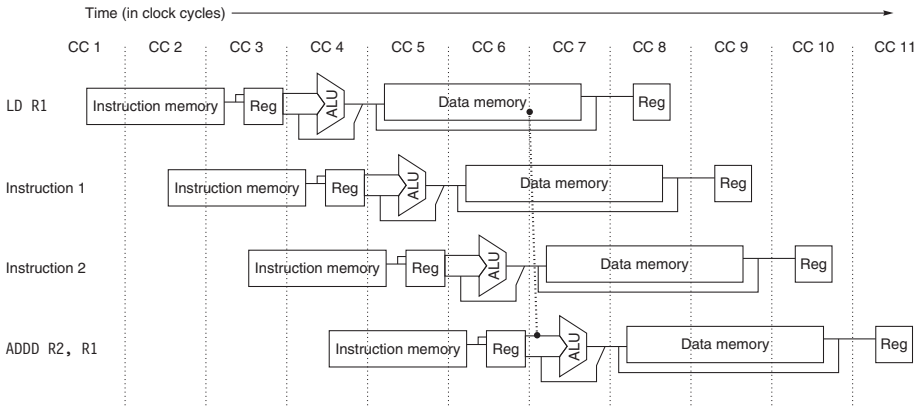
MIPS R4000 Integer Pipeline

- Below are the stages for the MIPS R4000 integer pipeline.
  - IF - first half of instruction fetch; PC selection occurs here with the initiation of the IC access
  - IS - second half of instruction fetch; complete IC access
  - RF - instruction decode, register fetch, hazard checking, IC hit detection
  - EX - effective address calculation, ALU operation, branch target address calculation and condition evaluation
  - DF - first half of data cache access
  - DS - second half of data cache access
  - TC - tag check to determine if DC access was a hit
  - WB - write back for loads and register-register operations



MIPS R4000 Integer Pipeline Load Delay

- A two delay cycle is possible because the loaded value is available at the end of the DS stage and can be forwarded.
- If the tag check in the TC stage indicates a miss, then the pipeline is backed up a cycle and the L1 DC miss is serviced.



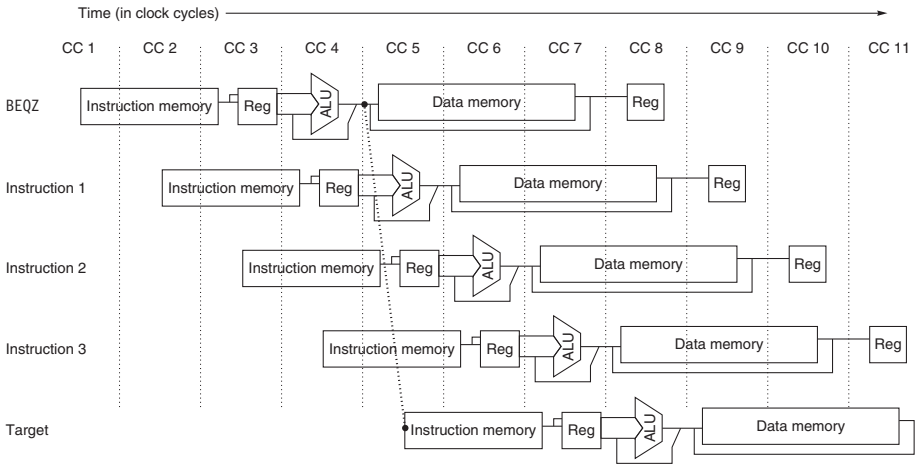
MIPS R4000 Integer Pipeline Load Delay (cont.)

- A load instruction followed by an immediate use of the loaded value results in a 2 cycle stall.

	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
LD R1,...	IF	IS	RF	EX	DF	DS	TC	WB	
DADD R2,R1,...		IF	IS	RF	Stall	Stall	EX	DF	DS
DSUB R3,R1,...			IF	IS	Stall	Stall	RF	EX	DF
OR R4,R1,...				IF	Stall	Stall	IS	RF	EX

MIPS R4000 Integer Pipeline Branch Delay

- The branch delay is 3 cycles since the condition evaluation is performed during the EX stage.



MIPS R4000 Integer Pipeline Branch Delay (cont.)

- A taken branch on the MIPS R4000 has a 1 cycle delay slot followed by a 2 cycle stall.

instruction	1	2	3	4	5	6	7	8	9	10	11	12
branch instruction	IF	IS	RF	EX	DF	DS	TC	WB				
delay slot		IF	IS	RF	EX	DF	DS	TC	WB			
branch instruction + 2			IF	IS	stall	stall	stall	stall	stall	stall		
branch instruction + 3				IF	stall	stall	stall	stall	stall	stall	stall	
branch target					IF	IS	RF	EX	DF	DS	TC	WB

MIPS R4000 Integer Pipeline Branch Delay (cont.)

- A not taken branch on the MIPS R4000 has just a 1 cycle delay slot.

instruction	1	2	3	4	5	6	7	8	9	10	11
branch instruction	IF	IS	RF	EX	DF	DS	TC	WB			
delay slot		IF	IS	RF	EX	DF	DS	TC	WB		
branch instruction + 2			IF	IS	RF	EX	DF	DS	TC	WB	
branch instruction + 3				IF	IS	RF	EX	DF	DS	TC	WB

Superpipelined Example

- Simple example using the MIPS pipeline.

cycle	1	2	3	4	5	6	7	8
lw $1,0($4)	IF	ID	EX	MEM	WB			
add $3,$1,$3		IF	ID	stall	**EX**	MEM	WB	
sub $7,$2,$3			IF	stall	ID	**EX**	MEM	WB

- How would these instructions proceed through the superpipeline with the following pipeline stages?

cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
lw $1,0($4)	IF1	IF2	ID	RF	EX1	EX2	DC1	DC2	WB								
add $3,$1,$3																	
sub $7,$2,$3																	

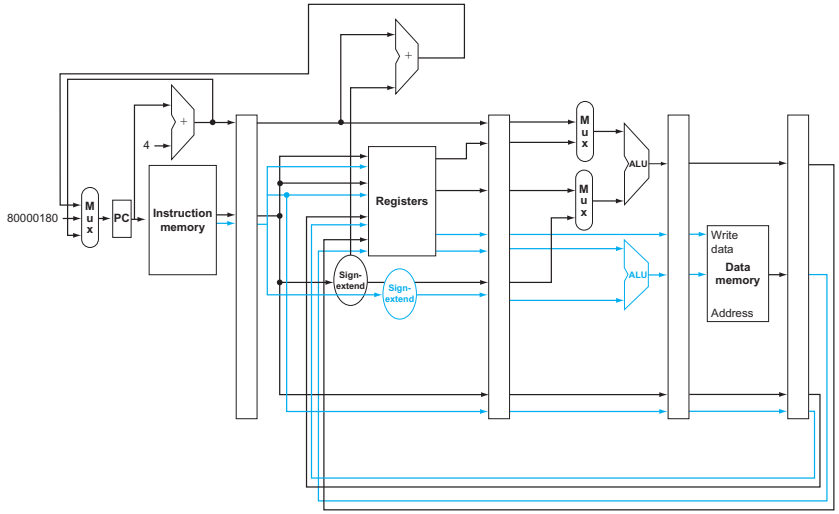
Static Multiple Issue

- In a static multiple-issue processor, the compiler has the responsibility for arranging the sets of instructions that are independent and can be fetched, decoded, and executed together.
- A static multiple-issue processor that simultaneously issues several independent operations in a single wide instruction is called a Very Long Instruction Word (VLIW) processor.
- Below is an example static two-issue pipeline in operation.

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

A Static Two-Issue Datapath

- The additions needed for double-issue are highlighted in blue.



Example of Scheduling Code for a Static Two-Issue MIPS

- Original loop in C:

```
for (i = n-1; i != 0; i = i-1)
    a[i] += s;
```

- Original loop in MIPS assembly:

```
Loop: lw   $t0,0($s1)      # $t0 = a[i];
      addu $t0,$t0,$s2     # $t0 += s;
      sw   $t0,0($s1)      # a[i] = $t0;
      addi $s1,$s1,-4      # i = i-1;
      bne  $s1,$zero,Loop # if (i!=0) goto Loop;
```

- Code scheduled for a static two-issue MIPS processor.

	ALU or branch instruction	Data transfer instruction	Clock cycle
Loop:		lw $t0, 0($s1)	1
	addi $s1,$s1,-4		2
	addu $t0,$t0,$s2		3
	bne $s1,$zero,Loop	sw $t0, 4($s1)	4

Loop Unrolling

- Loop unrolling is a compiler optimization that makes multiple copies of the body of a loop to reduce loop overhead and provide more scheduling opportunities.
- Original loop in C:

```
for (i = 0; i < n; i++)
    a[i] += s;
```

- Unrolled loop in C:

```
for (i = 0; i < n % 4; i++)
    a[i] = b[i] + c[i];
for (; i < n; i++) {
    a[i] = b[i] + c[i]; i++;
    a[i] = b[i] + c[i]; i++;
    a[i] = b[i] + c[i]; i++;
    a[i] = b[i] + c[i];
}
```

Loop Unrolling Example at the Assembly Level

```
# Original Loop
Loop: lw   $t0,0($s1)
      addu $t0,$t0,$s2
      sw   $t0,0($s1)
      addi $s1,$s1,-4
      bne  $s1,$zero,Loop
```

```
# Unrolled Loop
Loop: lw   $t0,0($s1)
      addu $t0,$t0,$s2
      sw   $t0,0($s1)
      addi $s1,$s1,-4
      lw   $t0,0($s1)
      addu $t0,$t0,$s2
      sw   $t0,0($s1)
      addi $s1,$s1,-4
      lw   $t0,0($s1)
      addu $t0,$t0,$s2
      sw   $t0,0($s1)
      addi $s1,$s1,-4
      lw   $t0,0($s1)
      addu $t0,$t0,$s2
      sw   $t0,0($s1)
      addi $s1,$s1,-4
      bne  $s1,$zero,Loop
```


Loop Unrolling Example after Adjusting Offsets

```
Loop: lw   $t0,0($s1)
      addi $s1,$s1,-4
      addi $s1,$s1,-4
      addi $s1,$s1,-4
      addi $s1,$s1,-4
      addu $t0,$t0,$s2
      sw   $t0,16($s1)
      lw   $t0,12($s1)
      addu $t0,$t0,$s2
      sw   $t0,12($s1)
      lw   $t0,8($s1)
      addu $t0,$t0,$s2
      sw   $t0,8($s1)
      lw   $t0,4($s1)
      addu $t0,$t0,$s2
      sw   $t0,4($s1)
      bne  $s1,$zero,Loop
```


Loop Unrolling Example after Combining Add Instructions

```
Loop: lw   $t0,0($s1)
      addi $s1,$s1,-16
      addu $t0,$t0,$s2
      sw   $t0,16($s1)
      lw   $t0,12($s1)
      addu $t0,$t0,$s2
      sw   $t0,12($s1)
      lw   $t0,8($s1)
      addu $t0,$t0,$s2
      sw   $t0,8($s1)
      lw   $t0,4($s1)
      addu $t0,$t0,$s2
      sw   $t0,4($s1)
      bne  $s1,$zero,Loop
```


Loop Unrolling Example after Renaming Registers

```
Loop: lw   $t0,0($s1)
      addi $s1,$s1,-16
      addu $t0,$t0,$s2
      sw   $t0,16($s1)
      lw   $t1,12($s1)
      addu $t1,$t1,$s2
      sw   $t1,12($s1)
      lw   $t2,8($s1)
      addu $t2,$t2,$s2
      sw   $t2,8($s1)
      lw   $t3,4($s1)
      addu $t3,$t3,$s2
      sw   $t3,4($s1)
      bne  $s1,$zero,Loop
```

Loop Unrolling Example after Scheduling Instructions

```
Loop: lw   $t0,0($s1)
      addi $s1,$s1,-16
      lw   $t1,12($s1)
      lw   $t2,8($s1)
      lw   $t3,4($s1)
      addu $t0,$t0,$s2
      sw   $t0,16($s1)
      addu $t1,$t1,$s2
      sw   $t1,12($s1)
      addu $t2,$t2,$s2
      sw   $t2,8($s1)
      addu $t3,$t3,$s2
      sw   $t3,4($s1)
      bne  $s1,$zero,Loop
```


Loop Unrolling Example after Two-Issue Scheduling

- Below is the code for a two-issue MIPS, where the second operation is used only for data transfer (load/store) instructions.
- Empty slots are noops.

	ALU or branch instruction		Data transfer instruction		Clock cycle
Loop:	addi	$s1,$s1,-16	lw	$t0, 0($s1)	1
			lw	$t1,12($s1)	2
	addu	$t0,$t0,$s2	lw	$t2, 8($s1)	3
	addu	$t1,$t1,$s2	lw	$t3, 4($s1)	4
	addu	$t2,$t2,$s2	sw	$t0, 16($s1)	5
	addu	$t3,$t3,$s2	sw	$t1,12($s1)	6
			sw	$t2, 8($s1)	7
	bne	$s1,$zero,Loop	sw	$t3, 4($s1)	8

Dynamic Multiple-Issue Processors

- Dynamic multiple-issue processors dynamically detect if sequential instructions can be simultaneously issued in the same cycle.
  - no data hazards (dependences)
  - no structural hazards
  - no control hazards
- This type of processors are also known as superscalar.
- One advantage of superscalar over static multiple-issue is that code compiled for single issue will still be able to execute.

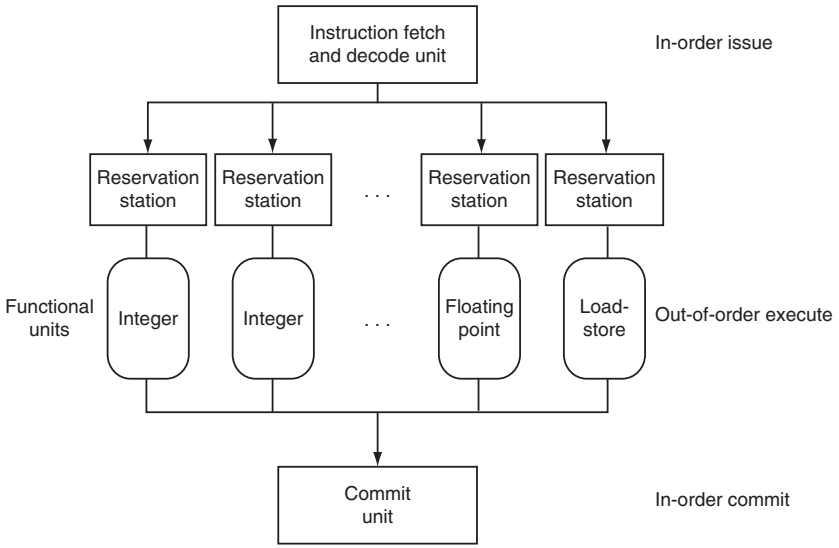
Out-of-Order Execution Processors

- Some processors are designed to execute instructions out of order to perform useful work when a given instruction is stalled.
- The *add* is dependent on the *lw*, but the *sub* is independent.

```
lw  $1,0($2)
add $3,$4,$1
sub $6,$4,$5
```

- Out-of-order (OoO) or dynamically scheduled processors:
  - fetch and issue instructions in order
  - execute instructions out of order
  - commit results in order
- Many OoO processors also support multi-issue to further improve performance.

Primary Units of a Dynamically Scheduled Pipeline



Changes in Intel Microprocessors

- Due to thermal limitations, the clock rate has not increased in recent years, which has led to fewer pipeline stages and the adoption of multi-core processors.

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power	
Intel 486	1989	25 MHz	5	1	No	1	5	W
Intel Pentium	1993	66 MHz	5	2	No	1	10	W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29	W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75	W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103	W
Intel Core	2006	2930 MHz	14	4	Yes	2	75	W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	1	87	W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	77	W

Specifications of an Embedded and Server Processor

Processor	ARM A8	Intel Core i7 920
Market	Personal Mobile Device	Server, Cloud
Thermal design power	2 Watts	130 Watts
Clock rate	1 GHz	2.66 GHz
Cores/Chip	1	4
Floating point?	No	Yes
Multiple Issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline Stages	14	14
Pipeline schedule	Static In-order	Dynamic Out-of-order with Speculation
Branch prediction	2-level	2-level
1st level caches / core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
2nd level cache / core	128–1024 KiB	256 KiB
3rd level cache (shared)	–	2–8 MiB

Fallacies and Pitfalls

- *Fallacy: Pipelining is easy.*
  - There are a lot of issues (forwarding, hazards, exceptions) to handle.
- *Fallacy: Pipelining ideas can be implemented independent of technology.*
  - The delayed branch only made sense with a short pipeline.
  - Dynamic pipelining became more feasible as logic became much faster than memory.
- *Pitfall: Failure to consider instruction set design can adversely impact pipelining.*
  - Variable length instructions and different running times can lead to imbalance among pipeline stages and complicate hazard detection.
  - Complicated addressing modes complicates pipeline control.