Instruction-Level Parallelism (ILP)

ILP: refers to the overlap execution of instructions.

Pipelined CPI = Ideal pipeline CPI + structural stalls + RAW stalls + WAR stalls + WAW stalls + Control stalls.

Latencies of FP operations used in chapter 4.

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Loop Unrolling Example

We examine how compiler can increase ILP by unrolling loops.

```
for (i=1; i<=1000; i++)
    x[i] = x[i] + s;
```

The correspnoding DLX assembly code:

```
Loop: LD    F0, 0(R1); F0=array element
      ADDD  F4, F0, F2; add scalar in F2
      SD    0(R1), F4; store result
      SUBI  R1, R1, #8; decrement pointer
                ; 8 bytes (per DW)
      BNEZR1, Loop; branch R1 != zero
```

R1: initially contains the address of the element in the array with the highest addr.

What is the value of R1?

What is the type of X arrary element?

F2 contains the scalar vlaue, s.

Increasing Instruction-Level Parallelism

Loop without scheduling (9cycles)Loop after pipeline scheduling(6cycles)

	clockcycle issued	clockcycle issued
Loop: LD F0, 0(R1)	1	Loop: LD F0, 0(R1) 1
stall	2	stall 2
ADDD F4, F0, F23		ADDD F4, F0, F23
stall	4	SUBIR1, R1, #84
stall	5	BNEZR1, Loop5
SD 0(R1), F4	6	SD 8(R1), F46
SUBI R1, R1, #87		
BNEZR1, Loop	8	
stall	9	

SD moves to the branch delay slot.

Loop Unrolling

Unroll loop once yield two copies of the loop body.

One copy of code will remain the same.

Register renaming and address offset modification are performed on other copies to maintain the right semantics.

For example, when the above loop is unrolled three times, the following modification is done on the second copies of the loop body, which implements x[i+1] = x[i+1]+s;

original loop body	second loop body after modification
==========	==========
LD F0, 0(R1)	LD F6, -8(R1); allocate F6 for x[i+1], its address is R1-8
ADDD F4, F0, F2	ADDD F8, F6, F2; rename registers to F8, F6.
SD 0(R1), F4	SD -8(R1), F8; address offset modification0→-8, F4→F8

Loop Unrolling

Loop unrolled three times (6.8cycles)Unrolled Loop after scheduling(3.5cycles)

```
Loop: LDF0, 0(R1)Loop:LDF0, 0(R1)
      ADDD F4, F0, F2LD  F6, -8(R1)
      SD 0(R1), F4    LD  F10, -16(R1)
      LDF6, -8(R1)    LD  F14, -24(R1)
      ADDD F8, F6, F2ADDDF4, F0, F2
      SD -8(R1), F8   ADDDF8, F6, F2
      LDF10, -16(R1)  ADDDF12, F10, F2
      ADDD F12, F10, F2ADDDF16, F14, F2
      SD -16(R1), F12 SD 0(R1), F4
      LDF14, -24(R1)  SD -8(R1), F8
      ADDD F16, F14, F2SD-16(R1), F12
      SD -24(R1), F16 SUBIR1,R1,#32
      SUBI R1, R1, #32BNEZR1, Loop
      BNEZ R1, Loop   SD 8(R1), F16; 8-32=-24
```

LD 2cycles; ADDD 3 cycles; Branch 2cycle14/4=3.5cycles (no stall)

4*(2+3+1)+1(SUBI)+2=27; 27/4=6.75

Dynamic Scheduling in Pipelines

Static Scheduling

- stall instruction and cease fetching and issuing until data dependence is cleared
- use software pipeline scheduling to minimize the stalls.

Dynamic Scheduling

- use hardware to rearrange the instruction execution to reduce the stalls.
- capable of handling cases where dependences unknown at compiler time.
- significantly increase hardware complexity.
- out of order execution

```
DIVF    F0, F2, F4
ADDF    F10, F0, F8
SUBF    F6, F6, F14
```

Scoreboarding

A *dynamic scheduling* advanced pipelining technique implemented in CDC 6600.

Use a scoreboard unit to control/coordinate multiple functional units in a pipeline.

Scoreboard unit is responsible for instruction issue and hazard detection.

For DLX scoreboard, ID stage is split into two steps—Issue and Read Operands; EX and WR stages are modified; IF stage unchanged.

Issue—If *the functional unit is free* and *no other active instruction has the same destination register* (**WAW**), then issue the instruction to the functional unit and update scoreboard tables; otherwise stall until the **structure hazard** is cleared.

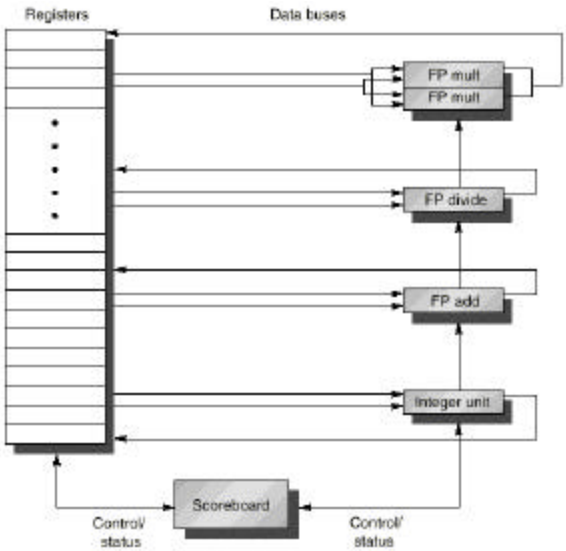
Read operands—Scoreboard monitors the availability of the source operands. When the source operands are available, scoreboard tells the functional unit to read the operands and begin execution.
A operand is available if no active instruction is going to write it or being actively written. (**RAW**)

Execution—notify scoreboard when finished.

Write result—Completing instruction cannot write while another instruction has not read the (to be written) operand. (**WAR**)

DLX Scoreboard

We have one integer unit, two FP multipliers, one FP adder, and one FP divider.



Pipeline Execution Pattern

Instruction\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14...17	18..
LF F6,34(R2)															
LF F2,45(R3)															
MULTF F0,F2,F4															
SUBF F8,F6,F2															
DIVF F10,F0,F6															
ADDF F6,F8,F2															

Scoreboard/Instruction Status Table

Instruction	Issued	Operand Read	Execution Complete	Result Written
LF F6,34(R2)				
LF F2,45(R3)				
MULTF F0,F2,F4				
SUBF F8,F6,F2				
DIVF F10,F0,F6				
ADDF F6,F8,F2				

Functional Unit Status Table

FU no.	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
1	Integer									
2	FP-mult1									
3	FP-mult2									
4	FP-add									
5	FP-div									

Register Result Status Table

FP register	F0	F2	F4	F6	F8	F10	F12...F30
FU no.							

Simple DLX Scoreboard

The DLX scoreboard example in our text book assumes that the integer functional unit is non-pipelined and at any one time only one integer instruction can use the unit.

The following are the pipeline execution pattern and the history of the scoreboard changes for the simple DLX Scoreboard in our text book.

Note that here we only have one integer unit. Exercise 8? assumes 3 integer units and therefore the execution of integer instructions can be overlapped.

Bookkeeping in DLX Scoreboard Unit

FU: the functional unit used the instruction.

op: the operation of the instruction.

D: the destination register of the instruction.

S1 and S2: the source registers of the instruction.

IS, OR, EC, RW: are the fields in the Instruction Status Table.

Busy(f), Op(f), Fi(f), Fj(f), Fk(f), Qj(f), Qk(f), Rj(f), Rk(f): are the fields in Functional Unit Status Table.

Result(reg): the FU no. field of register reg in the Register Result Status Table.

Issue: If (not(Busy(FU)=No and Result(D)=clear)) then stall the issuing
Wait until (Busy(FU)=No and Result(D)=clear)
DO {If (issuing is stalled) then restart the instruction issue and fetch;
Busy(FU)←Yes; Op(FU)←op; Fi(FU)←D; Fj(FU)←S1; Fk(FU)←S2;
Qj(FU)←Result(S1); Qk(FU)←Result(S2);
Rj(FU)←if Qj(FU)=clear then Yes else No;
Rj(FU)←if Qj(FU)=clear then Yes else No;
Result(D)←FU; Issue the instruction to FU; IS←Ð}

Bookkeeping in DLX Scoreboard Unit

Read Operands: Wait Until (Rj(FU)=Yes and Rk(FU)=Yes)
Do {Rj(FU)←No; Rk(FU)←No; Notify FU to read operands;
OR←Ð}
Execution Complete: Wait Until FU done Do EC←Ð;
Write Result: Wait Until (∀f((Fj(f)¦D or Rj(f)=No) & (Fk(f)¦D or Rk(f)=No)))
Do {∀f(if (Qj(f)=FU) then Rj(f)←Yes;
∀f(if (Qk(f)=FU) then Rk(f)←Yes;
Result(D)←clear; Busy(FU)←No;
Notify FU to write result to register;
RW←Ð}

Pipeline Execution Pattern

Instruction\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LF F6,34(R2)	IF	IS	RO	EX	EX	WR											
LF F2,45(R3)		IF	ISw	ISw	ISw	ISw	IS	RO	EX	EX	WR						
MULTF F0,F2,F4			IF	S	S	S	IS	Rw	Rw	Rw	Rw	RO	EX	EX	EX	EX	EX
SUBF F8,F6,F2							IF	IS	Rw	Rw	Rw	RO	EX	EX	WR		
DIVF F10,F0,F6								IF	IS	Rw	Rw	Rw	Rw	Rw	Rw	Rw	Rw
ADDF F6,F8,F2									IF	ISw	ISw	ISw	ISw	ISw	ISw	IS	RO

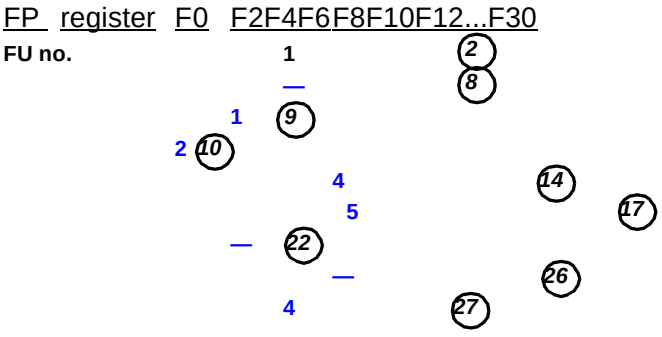
Scoreboard/Instruction Status Table

Instruction	Issued	Operand Read	Execution Completed	Result Written
LF F6,34(R2) *1*	*Ð2*	*Ð4*	*Ð7*	*Ð8*
LF F2,45(R3) *3*	*Ð9*	*Ð12*	*Ð21*	*Ð22*
MULTF F0,F2,F4 *6*	*Ð10*	*Ð23*		
SUBF F8,F6,F2 *11*	*Ð14*	*Ð24*	*Ð25*	*Ð26*
DIVF F10,F0,F6 *15*	*Ð17*			
ADDF F6,F8,F2 *18*	*Ð27*	*Ð28*		

History of Functional Unit Status Table

FU no.	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
1	Integer	*2*Yes	LF	F6	R2				Yes	
									*4*No	No
		*8*No								
		*9*Yes	LF	F2	R3				Yes	
									*12*No	No
2	FP-mult1	*10*Yes	MULTF	F0	F2	F4	1		No	Yes
									*22*Yes	
									*23*No	No
3	FP-mult2									
4	FP-add	*14*Yes	SUBF	F8	F6	F2		1	Yes	No
										*22*Yes
									*24*No	No
		*26*No								
		*27*Yes	ADDF	F6	F8	F2			Yes	Yes
									*28*No	No
5	FP-divide	*17*Yes	DIVF	F10	F0	F6			Yes	No

History of Register Result Status Table



Execution Detail in DLX Scoreboard Unit

1. Fetch instruction LF F6,34(R2), enter it in IST.

2. Issue: (Busy(1)=No and Result(F6)=clear)) is true; proceed
If (not(Busy(FU)=No and Result(D)=clear)) then stall the issuing
Wait until (Busy(FU)=No and Result(D)=clear)
DO {If (issuing is stalled) then restart the instruction issue and fetch;
Busy(FU)←Yes; Op(FU)←op; Fi(FU)←D; Fj(FU)←S1; Fk(FU)←S2;
Qj(FU)←Result(S1); Qk(FU)←Result(S2);
Rj(FU)←if Qj(FU)=clear then Yes else No;
Rj(FU)←if Qj(FU)=clear then Yes else No;
Result(D)←FU; Issue the instruction to FU; IS←Ð}

3. Fetch instruction LF F2,45(R3), enter it in IST.

4. RO: (Rj(1)=Yes and Rk(1)=Yes) is true; proceed.
Wait Until (Rj(FU)=Yes and Rk(FU)=Yes)
Do {Rj(FU)←No; Rk(FU)←No; Notify FU to read operands;
OR←Ð}

5. Issue: Busy(1)=Yes,
therefore (Busy(1)=No and Result(F6)=clear)) is not true;
stall the instruction issuing. and wait until condition becomes true.

6. Execution Complete: Notified by integer unit; set EC←Ð.

8. Write Result: D=F6; FU=1; none of Fj or Fk has value therefore
(Fj(f)¦D or Rj(f)=No) & (Fk(f)¦D or Rk(f)=No) is true for each functional unit f, therefore
∀f((Fj(f)¦D or Rj(f)=No) & (Fk(f)¦D or Rk(f)=No)) is true.

Do {∀f(if (Qj(f)=FU) then Rj(f)←Yes;
∀f(if (Qk(f)=FU) then Rk(f)←Yes;
Result(D)←clear; Busy(FU)←No;
Notify FU to write result to register;
RW←Ð}

9. With Busy(1)←No set at Event 8, the stall instruction LF F2,45(R3) now
with condition (Busy(1)=No and Result(F6)=clear)) = true and therefore it can proceed;
It will restart the instruction issue and set the scoreboard table.

13. RO: Since Rj(3) is No (F2 is not ready), (Rj(1)=Yes and Rk(1)=Yes) is false and MULTF will have to wait in RO step. same for 16

20. IS: the FP-add is still Busy, ADDF has to wait and stall issue.

22. D=F2; FU=1; for f=1, (Fj(1)=R3¦D & Fk(1)=—¦D) is true;
for f=2, (Fj(2)=F2¦D *false* or Rj(2)=No *true*)&(Fk(2)=F4¦D *true*) is true;
for f=3, (Fj and Fk are clear) therefore
(Fj(3)¦D or Rj(3)=No) & (Fk(3)¦D or Rk(3)=No) is true
for f=4, (Fk(4)=F2¦D *false* or Rj(2)=No *true*) &(Fj(2)=F6¦D *true*) is true;
for f=5, (Fj(5)=F0¦D & Fk(5)=F6¦D) is true.

Therefore, ∀f((Fj(f)¦D or Rj(f)=No) & (Fk(f)¦D or Rk(f)=No)) is true.
Write Result step can proceed with Do block.
Do {∀f(if (Qj(f)=FU) then Rj(f)←Yes;
∀f(if (Qk(f)=FU) then Rk(f)←Yes;
Result(D)←clear; Busy(FU)←No;
Notify FU to write result to register;
RW←Ð}

Here Qj(2)=1→ Rj(2)←Yes; Qk(4)=1→ Rj(4)←Yes;
Result(F2)←clear; Busy(1)←No; Notify FU to write result to register;
RW←Ð.

23. RO: With Rj(2) set to Yes at Event 22, MULTF now can proceed to read operands. Same for SUBF at Event 24.

26. Similar to 22, try to verify the condition is true yourself.

27. Busy(4) is set to No at Event 26, therefore ADDF can proceed with instruction issue.

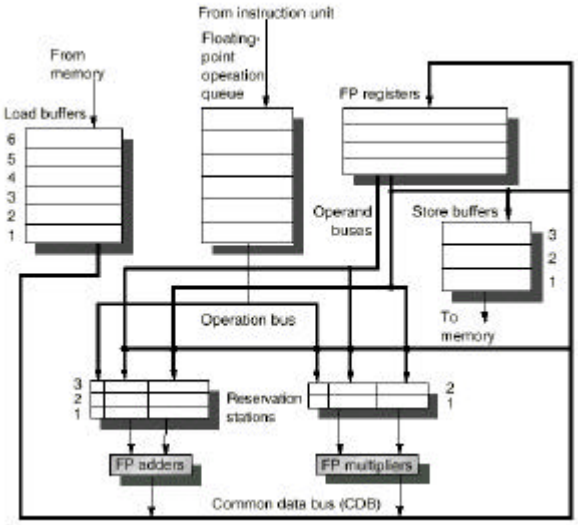
Tomasulo Algorithm

A *distributed dynamic scheduling* advanced pipelining technique invented by R. Tomasulo and implemented in IBM 360/91.

- Hazard detection and execution control are **distributed** among the pipeline execution control unit, the reservation station in each FP unit, and the control unit in each load or store buffer.
- The source operands are copied to the reservation station at instruction issue step if they are available → avoid the WAR hazard.
- Result are broadcast to all functional units via a common data bus, CDB. → reduce the contention of the bus access among the waiting function units.
- The broadcast result consists of the data and a tag which indicates the source of the broadcasting. Each waiting functional unit watches the tag to see if this is the waiting result.
- Use **dynamic register renaming** to avoid the WAW and WAR hazard.

Tomasulo-based FP Unit for DLX (TFP)

3 FP-adders, 2 FP-multipliers, 6 Load Buffers, 3 Store Buffers.



Pipeline Execution Pattern

F: Fetch, I: Issue, D:Decode, E: Execution, X: wait at execution stage, M:MEM,W: write result

	FP load & store	FP add & sub	FP multiplication	FP division
clock cycles	2	4	10	20

DLX	cycles operands	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18		37	38	39
LD	F6, 34(R2)	F	I	E	E	W																	
LD	F2, 45(R3)		F	I	E	E	W																
MULTD	F0, F2, F4			F	I	X	X	**E**	E	E	E	E	E	E	E	E	E	W					
SUBD	F8, F6, F2				F	I	X	**E**	E	E	E	W											
DIVD	F10, F0, F6					F	I	X	X	X	X	X	X	X	X	X	X	X	E		E	W	
ADDD	F6, F8, F2						F	I	X	X	X	X	E	E	E	E	W						

Reservation Stations (RS)

RS Name	Busy	Op	Vj	Vk	Qj	Qk
Add1						
Add2						
Add3						
Mult1						
Mult2						

When a function unit is reserved, the pipeline control unit

- copies the op code to Op field in the reservation station,
- copies the operand value to V field from the register files if available there.
- if the operand is being generated by some other function unit or load buffers, put the name of the function unit in the Q field.

The reservation stations observe data on Common Data Bus (CDB) and grab in needed data and save in corresponding V field.

When all operands for the operation are ready, the reservation station launch op.

Pipeline Execution Pattern

F: Fetch, I: Issue,D:Decode, E: Execution, X: wait at execution stage, M:MEM,W: write result

	FP load & store	FP add & sub	FP multiplication	FP division
clock cycles	2	4	10	20

DLX	cycles operands	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		24	25	26	27
L1: LD	F0, 0(R1)	F	D	E	M	W																	
MULTD	F4, F0, F2		F	D	S	E	E	E	E	E	E	E	E	E	E	M	W						
SUBI	R1, R1, #8			F	S	D	E	**M**	W														
BNEZ	R1, L1					F	D	**E**	M	W													
SD	0(R1), F4						F	D	S	S	S	S	S	S	S	E	M	W					
L1: LD	F0, 0(R1)							F	D	E	M	W											
MULTD	F4, F0, F2								F	D	S	S	S	S	S	E	E	E		E	M	W	
SUBI	R1, R1, #8									F	S	S	S	S	S	D	E	M	W				
BNEZ	R1, L1															F	D	E	M				
SD	0(R1), F4																F	D	S	S	E	M	W

Pipeline Execution Pattern

F: Fetch, I: Issue,D:Decode, E: Execution, X: wait at execution stage, M:MEM,W: write result

	FP load & store	FP add & sub	FP multiplication	FP division
clock cycles	2	4	10	20

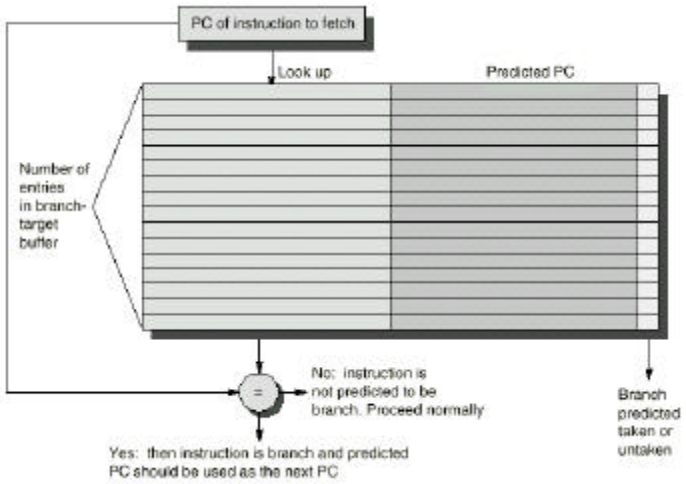
DLX	cycles operands	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
L1: LD	F0, 0(R1)	F	I	E	E	W																	
MULTD	F4, F0, F2		F	I	X	X	E	E	E	E	E	E	E	E	E	E	W						
SUBI	R1, R1, #8			F	D	E	M	**W**															
BNEZ	R1, L1				F	D	E	**M**	W														
SD	0(R1), F4					F	I	X	X	X	X	X	X	X	X	X	X	E	E	W			
L1: LD	F0, 0(R1)						F	I	E	E	W												
MULTD	F4, F0, F2							F	I	X	X	E	E	E	E	E	E	E	E	E	E	W	
SUBI	R1, R1, #8								F	D	E	M	W										
BNEZ	R1, L1									F	D	E	M	W									
SD	0(R1), F4										F	I	X	X	X	X	X	X	X	E	E	W	

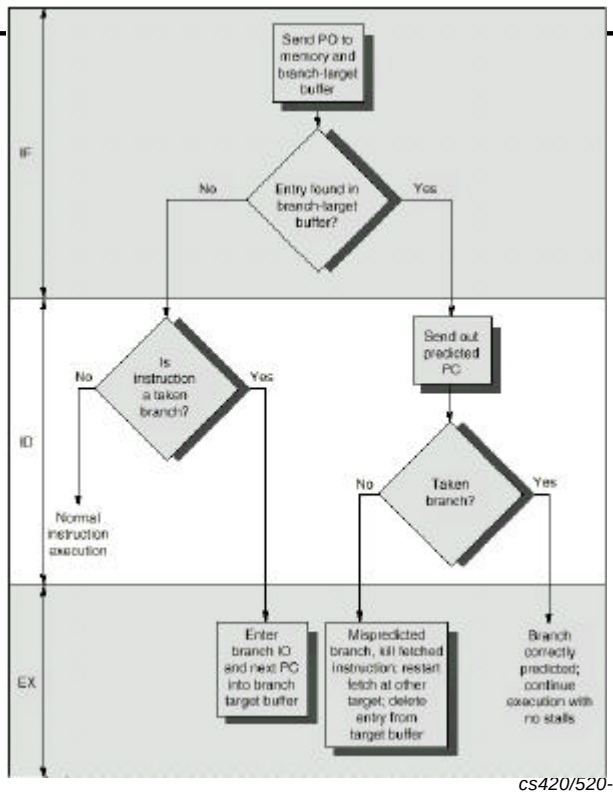
History of TFP tables

RS Name	Busy	Op	Vj	Vk	Qj	Qk
Mult1	Yes	MULTD		Regs[F2]	load1	
Mult2	Yes	MULTD		Regs[F2]	load1	

Reducing Branch Penalty with Branch Target Buffer

PC of the instruction to be fetched is sent to a branch target buffer (like a cache). If match, the predicted PC field of the cache entry contains the prediction of next PC after the branch. Fetching immediately begin at that address.





Achieving Zero Cycle Branch by Branch Folding

- instead of storing predicted PC (target address), storing the target instruction(s).
- replace the branch instruction with target instruction at the end of IF stage.
  → **zero cycle is spent on branch instruction!**
- no prediction penalty on uncoditional jump.

Superscalar:

Machines that issue multiple independent instructions per clock cycle

Instruction type	Pipe stages							
Integer instruction	IF	ID	EX	MEM	WB			
FP instruction	IF	ID	EX	MEM	WB			
Integer instruction		IF	ID	EX	MEM	WB		
FP instruction		IF	ID	EX	MEM	WB		
Integer instruction			IF	ID	EX	MEM	WB	
FP instruction			IF	ID	EX	MEM	WB	
Integer instruction				IF	ID	EX	MEM	WB
FP instruction				IF	ID	EX	MEM	WB

FIGURE 4.26 Superscalar pipeline in operation.

The unrolled and schedule code on a Superscalar DLX (2.4 cycles per element)

	Integer instruction	FP instruction	Clock cycle
Loop:	LD F0,0(R1)		1
	LD F6,-8(R1)		2
	LD F10,-16(R1)	ADDD F4,F0,F2	3
	LD F14,-24(R1)	ADDD F8,F6,F2	4
	LD F18,-32(R1)	ADDD F12,F10,F2	5
	SD 0(R1),F4	ADDD F16,F14,F2	6
	SD -8(R1),F8	ADDD F20,F18,F2	7
	SD -16(R1),F12		8
	SD -24(R1),F16		9
	SUBI R1,R1,#40		10
	BNEZ R1,LOOP		11
	SD -32(R1),F20		12

FIGURE 4.27 The unrolled and scheduled code as it would look on a superscalar DLX.

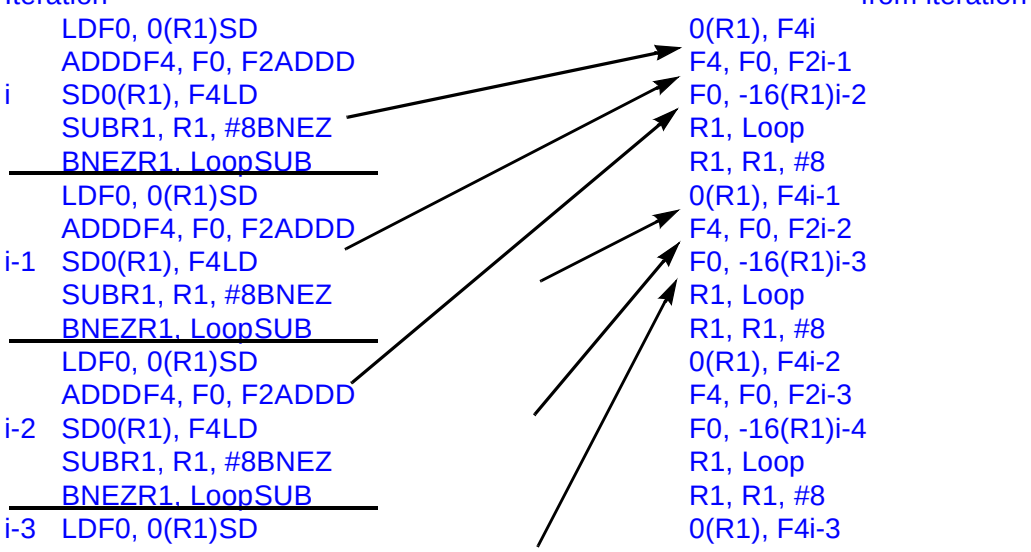
VLIW (Very Long Instruction Word)

Memory reference 1	Memory reference 2	FP operation 1	FP operation 2	Integer operation/branch
LD F0,0(R1)	LD F6,-8(R1)			
LD F10,-16(R1)	LD F14,-24(R1)			
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2	
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2	
		ADDD F20,F18,F2	ADDD F24,F22,F2	
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2		
SD -16(R1),F12	SD -24(R1),F16			
SD -32(R1),F20	SD -40(R1),F24			SUBI R1,R1,#56
SD -0(R1),F28				BNEZ R1,LOOP

- 1.28 cycles per result.
- only 60% of slots contain operations.
- VLIW is best for the instruction sequence with large no. of parallelism.
- VLIW requires high memory and register bandwidth. (multi-port memory)
- VLIW code size is larger.

Software Pipelining

- Consume less code space than straight loop unrolling.



Hint for Exercise 6.15

For exercises 6.15, TFP pipeline execution pattern

Instruction\Cycle	1	2	3	4	5	6	7	8	9	10	11	...	16	17	18	19
LD F2, 0(R1)	IF	ID	IS	EX	EX	WR										
MULTD F4, F2, F0		IF	ID	IS	EXw	EXw	EX	EX	EX	EX	EX	...	EX	WR		
LD F6, 0(R2)			IF	ID	IS	EX	EX	WR								
ADDD F6, F4, F6				IF	ID	IS	EXw	EXw	EXw					EXw	EX	
SD 0(R2), F6					IF	ID	IS	EXw	EX	EX	WR					
ADDI R1, R1, #8						IF	ID	EX	MEM	WB						
ADDI R2, R2, #8							IF	ID	EX*	MEM	WB					
SGTI R3, R1, Done								IF	ID	EX	MEM	WB				

Compared with the following scoreboard pipeline execution

Scoreboard INSTRUCTION COUNT TABLE

Instruction\Cycle	IF	IS	RO	EX	WR
LD F2, 0(R1)	1	2	3	4,5	6
MULTD F4, F2, F0	2	3	7	8-16	18
LD F6, 0(R2)	3	4	5	6,7	8
ADDD F6, F4, F6	4	9	18	19-24	25
SD 0(R2), F6	9	10	26	27,28	29
ADDI R1, R1, #8	10	11	12	13	14
ADDI R2, R2, #8	11	12	13	14	27*