ADD R	Inst	Name	FMT	Opcode	funct3	funct7		7
SUB SUB R 0110011 0x4 0x00 imm[12]	add	ADD	R	_	0x0	0×00	fu	
NOR	sub	SUB	R	0110011	0x0	0x20		
and AND sl1 Shift Left Logical R 0110011 0x1 0x00 sr1 Shift Right Logical R 0110011 0x5 0x20 sr2 Shift Right Arith* R 0110011 0x5 0x20 slt Set Less Than R 0110011 0x5 0x20 slt Set Less Than R 0110011 0x2 0x00 addi ADD Immediate I 0010011 0x0 addi ADD Immediate I 0010011 0x6 andi AND Immediate I 0010011 0x6 andi AND Immediate I 0010011 0x7 slti Shift Left Logical Imm I 0010011 0x7 slii Shift Right Logical Imm I 0010011 0x7 sr1i Shift Right Logical Imm I 0010011 0x5 sr2i Shift Right Arith Imm I 0010011 0x5 sr3i Shift Right Arith Imm I 0010011 0x5 slti Set Less Than Imm I 0010011 0x2 slti Set Less Than Imm I 0010011 0x3 lb Load Byte I 0000011 0x3 lb Load Byte I 0000011 0x0 lh Load Half I 0000011 0x1 lh Load Half I 0000011 0x1 lb Load Byte (U) I 0000011 0x2 lbu Load Byte (U) I 0000011 0x5 sh Store Byte S 0100011 0x5 sh Store Half S 0100011 0x1 sh Store Half S 0100011 0x1 beq Branch == B 1100011 0x3 beq Branch == B 1100011 0x5 bltu Branch < B 1100011 0x5 bltu Branch < B 1100011 0x5 bltu Branch < CU) B 1100011 0x6 bgeu Branch ≥ I 1100111 0x7 lui Load Upper Imm U 0 110111 auipc Add Upper Imm U 0 0100111 auipc Add Upper Imm to PC U 0010111	xor	XOR	R	0110011	0x4	0x00		-
Shift Left Logical R	or	OR	R	0110011	0x6	0x00	imm	12
sr1	and	AND	R	0110011	0x7	0x00		
STA Shift Right Arith* R 0110011 0x5 0x20	sll	Shift Left Logical	R	0110011	0x1	0x00		
slt Set Less Than R 0110011 0x2 0x00 sltu Set Less Than (U) R 0110011 0x3 0x00 addi ADD Immediate I 0010011 0x0 xori XOR Immediate I 0010011 0x4 ori OR Immediate I 0010011 0x7 slii Shift Left Logical Imm I 0010011 0x7 srli Shift Right Logical Imm I 0010011 0x5 imm[5:11]=0x00 srai Shift Right Arith Imm I 0010011 0x5 imm[5:11]=0x20 sti Set Less Than Imm I 0010011 0x2 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 lb Load Byte I 0000011 0x3	srl	Shift Right Logical	R	0110011	0x5	0x00		
Set Less Than (U)	sra	Shift Right Arith*	R	0110011	0x5	0x20		
Addi	slt	Set Less Than	R	0110011	0x2	0x00		
xori XOR Immediate I 0010011 0x4 ori OR Immediate I 0010011 0x6 andi AND Immediate I 0010011 0x7 slli Shift Left Logical Imm I 0010011 0x1 imm[5:11]=0x00 srli Shift Right Logical Imm I 0010011 0x5 imm[5:11]=0x00 srai Shift Right Arith Imm I 0010011 0x5 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 lh Load Byte I 0000011 0x0 sb Store Byte S 0100011 0x1	sltu	Set Less Than (U)	R	0110011	0x3	0x00		
ori	addi	ADD Immediate	I	0010011	0x0			7
andi AND Immediate slli Shift Left Logical Imm srli Shift Right Logical Imm I 0010011 0x1 imm[5:11]=0x00 srai Shift Right Arith Imm I 0010011 0x5 imm[5:11]=0x00 srai Shift Right Arith Imm I 0010011 0x5 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 sltiu Set Less Than Imm (U) I 0010011 0x3 lb Load Byte II 0000011 0x0 lh Load Half II 0000011 0x1 lw Load Word II 0000011 0x2 lbu Load Byte (U) I 0000011 0x5 sb Store Byte Sh Store Byte Sh Store Half Sw Store Word Soll 000011 0x1 sw Store Word Soll 000011 0x2 beq Branch == B 1100011 0x0 bhe Branch != B 1100011 0x1 blt Branch < B 1100011 0x4 bge Branch ≥ B 1100011 0x5 bltu Branch < (U) B 1100011 0x7 jal Jump And Link plan I 100111 0x0 lui Load Upper Imm Add Upper Imm to PC ecall Environment Call I 1110011 0x0 imm=0x0	xori	XOR Immediate	I	0010011	0x4			
s1li Shift Left Logical Imm I 0010011 0x1 imm[5:11]=0x00 srli Shift Right Logical Imm I 0010011 0x5 imm[5:11]=0x00 srai Shift Right Arith Imm I 0010011 0x5 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 sltiu Set Less Than Imm I 0010011 0x3 lb Load Byte I 0000011 0x3 lh Load Half I 0000011 0x0 lbu Load Byte (U) I 0000011 0x2 lbu Load Byte (U) I 0000011 0x4 lbu Load Byte (U) I 0000011 0x5 sb Store Byte S 0100011 0x0 sh Store Half S 0100011 0x1 sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x4 blt Branch < B 1100011 0x5 bltu Bran	ori	OR Immediate	I	0010011	0x6			
srli Shift Right Logical Imm I 0010011 0x5 imm[5:11]=0x00 srai Shift Right Arith Imm I 0010011 0x5 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 sltiu Set Less Than Imm (U) I 0010011 0x3 lb Load Byte I 0000011 0x0 lh Load Half I 0000011 0x2 lbu Load Byte (U) I 0000011 0x2 lbu Load Byte (U) I 0000011 0x4 lbu Load Half (U) I 0000011 0x4 lbu Load Half (U) I 0000011 0x5 sb Store Byte S 0100011 0x0 sh Store Half S 0100011 0x2 beq Branch == B 1100011 0x0 bne Branch < B 1100011 0x4 bge Branch < B 1100011 0x5 bltu Branch < U	andi	AND Immediate	I	0010011	0x7			
srai Shift Right Arith Imm I 0010011 0x5 imm[5:11]=0x20 slti Set Less Than Imm I 0010011 0x2 imm[5:11]=0x20 sltiu Set Less Than Imm I 0010011 0x2 imm[5:11]=0x20 lb Load Byte I 0010011 0x3 lh Load Half I 0000011 0x1 lw Load Byte (U) I 0000011 0x2 lbu Load Byte (U) I 0000011 0x4 lbu Load Byte (U) I 0000011 0x4 lbu Load Byte (U) I 0000011 0x5 sb Store Byte S 0100011 0x0 sh Store Byte S 0100011 0x1 sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x4 bge Branch < B 1100011 0x5 bltu Branch < <td>slli</td> <td>Shift Left Logical Imm</td> <td>I</td> <td>0010011</td> <td>0x1</td> <td>imm[5:11</td> <td>]=0x00</td> <td></td>	slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	
slti Set Less Than Imm I 0010011 0x2 sltiu Set Less Than Imm (U) I 0010011 0x3 lb Load Byte I 0000011 0x0 lh Load Half I 0000011 0x2 lbu Load Byte (U) I 0000011 0x4 lhu Load Half (U) I 0000011 0x5 sb Store Byte S 0100011 0x0 sh Store Half S 0100011 0x1 sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x0 bne Branch != B 1100011 0x4 bge Branch < B 1100011 0x5 bltu Branch <<(U) B 1100011 0x6 bgeu Branch ≥ (U) B 1100011 0x7 jal Jump And Link J 1101111 0x0 lui Load Upper Imm U 0110111 0x0 imm=0x0	srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	
sltiu Set Less Than Imm (U) I 0010011 0x3 lb Load Byte I 0000011 0x0 lh Load Half I 0000011 0x1 lw Load Word I 0000011 0x2 lbu Load Byte (U) I 0000011 0x4 lhu Load Half (U) I 0000011 0x5 sb Store Byte S 0100011 0x0 sh Store Half S 0100011 0x1 sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x0 bne Branch != B 1100011 0x4 bge Branch ≥ B 1100011 0x5 bltu Branch ≥ (U) B 1100011 0x6 bgeu Branch ≥ (U) B 1100011 0x7 jal Jump And Link J 1101111 0x0 lui	srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	
Description Description Description Description	slti	Set Less Than Imm	I	0010011	0x2			
1h Load Half I 0000011 0x1 1w Load Word I 0000011 0x2 1bu Load Byte (U) I 0000011 0x4 1hu Load Half (U) I 0000011 0x5 sb Store Byte S 0100011 0x0 sh Store Half S 0100011 0x1 sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x0 bne Branch != B 1100011 0x1 blt Branch <	sltiu	Set Less Than Imm (U)	I	0010011	0x3			
lw Load Word I 0000011 0x2 lbu Load Byte (U) I 0000011 0x4 lhu Load Half (U) I 0000011 0x5 sb Store Byte S 0100011 0x0 sh Store Half S 0100011 0x1 sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x0 bne Branch != B 1100011 0x1 blt Branch <	1b	Load Byte	I	0000011	0x0			7
Diamond Dia	1h	Load Half	I	0000011	0x1			
Ihu Load Half (U) I 0000011 0x5 sb Store Byte S 0100011 0x0 sh Store Half S 0100011 0x1 sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x0 bne Branch != B 1100011 0x1 blt Branch < B 1100011 0x4 bge Branch < (U) B 1100011 0x5 bltu Branch ≥ (U) B 1100011 0x7 jal Jump And Link J 1101111 0x0 lui Load Upper Imm U 0110111 0x0 lui Load Upper Imm to PC U 0010111 0x0 imm=0x0	lw	Load Word	I	0000011	0x2			
sb Store Byte S 0100011 0x0 sh Store Half S 0100011 0x1 sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x0 bne Branch != B 1100011 0x1 blt Branch < B 1100011 0x4 bge Branch ≥ B 1100011 0x5 bltu Branch ≥ (U) B 1100011 0x6 bgeu Branch ≥ (U) B 1100011 0x7 jal Jump And Link J 1101111 jalr Jump And Link Reg I 1100111 0x0 lui Load Upper Imm U 0110111 0x0 lui Add Upper Imm to PC U 0010111 0x0 imm=0x0	1bu	Load Byte (U)	I	0000011	0x4			
sh Store Half S 0100011 0x1 sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x0 bne Branch != B 1100011 0x1 blt Branch < B 1100011 0x4 bge Branch < (U) B 1100011 0x5 bltu Branch < (U) B 1100011 0x6 bgeu Branch ≥ (U) B 1100011 0x7 jal Jump And Link J 1101111 0x0 lui Load Upper Imm U 0110111 0x0 lui Add Upper Imm to PC U 0010111 0x0 imm=0x0	1hu	Load Half (U)	I	0000011	0x5			
sw Store Word S 0100011 0x2 beq Branch == B 1100011 0x0 bne Branch != B 1100011 0x1 blt Branch <	sb	Store Byte	S	0100011	0x0			7
beq Branch == B 1100011 0x0 bne Branch != B 1100011 0x1 blt Branch <	sh	Store Half	S	0100011	0x1			
bne Branch != B 1100011 0x1 blt Branch <	SW	Store Word	S	0100011	0x2			
blt Branch < B 1100011 0x4 bge Branch ≥ B 1100011 0x5 bltu Branch < (U) B 1100011 0x6 bgeu Branch ≥ (U) B 1100011 0x7 jal Jump And Link J 1101111 jalr Jump And Link Reg I 1100111 0x0 lui Load Upper Imm U 0110111 auipc Add Upper Imm to PC U 0010111 ecall Environment Call I 1110011 0x0 imm=0x0	beq	Branch ==	В	1100011	0x0			7
bge Branch ≥ B 1100011 0x5 bltu Branch < (U) B 1100011 0x6 bgeu Branch ≥ (U) B 1100011 0x7 jal Jump And Link J 11011111 jalr Jump And Link Reg I 1100111 0x0 lui Load Upper Imm U 0110111 0x0 auipc Add Upper Imm to PC U 0010111 0x0 imm=0x0 ecall Environment Call I 1110011 0x0 imm=0x0	bne	Branch !=	В	1100011	0x1			
bltu Branch < (U) B 1100011 0x6 bgeu Branch ≥ (U) B 1100011 0x7 jal Jump And Link J 1101111 jalr Jump And Link Reg I 1100111 0x0 lui Load Upper Imm U 0110111 auipc Add Upper Imm to PC U 0010111 ecall Environment Call I 1110011 0x0 imm=0x0	blt	Branch <	В	1100011	0x4			
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	bge	Branch ≥	В	1100011	0x5			
bgeu Branch ≥ (U) B 1100011 0x7 jal Jump And Link J 1101111 1101111 jalr Jump And Link Reg I 1100111 0x0 lui Load Upper Imm U 0110111 0x0 auipc Add Upper Imm to PC U 0010111 0x0 ecall Environment Call I 1110011 0x0 imm=0x0	bltu	Branch < (U)	В	1100011	0x6			1
jalr Jump And Link Reg I 1100111 0x0 lui Load Upper Imm U 0110111 0x0 auipc Add Upper Imm to PC U 0010111 0x0 imm=0x0 ecall Environment Call I 1110011 0x0 imm=0x0	bgeu	Branch \geq (U)	В	1100011	0x7			_ ՝
lui Load Upper Imm U 0110111 auipc Add Upper Imm to PC U 0010111 ecall Environment Call I 1110011 0x0 imm=0x0	jal	Jump And Link	J	1101111				7
auipc Add Upper Imm to PC U 0010111 O ecall Environment Call I 1110011 0x0 imm=0x0	jalr	Jump And Link Reg	I	1100111	0x0			
ecall Environment Call I 1110011 0x0 imm=0x0	lui	Load Upper Imm	U	0110111				7
	auipc	Add Upper Imm to PC	U	0010111				(
ebreak Environment Break I 1110011 0x0 imm=0x1	ecall	Environment Call	I	1110011	0x0	imm=0x0		٦,
	ebreak	Environment Break	I	1110011	0x0	imm=0x1] '

31	27	26	25	24	20	19	15	14	12	11	7	6	0	
	func	t7		rs	:2	rs1		fun	ct3		rd	opc	ode	R-type
	ir	nm[:	11:0)]		rs1		fun	ct3		rd	opc	ode	I-type
ir	nm[1	1:5]		rs	2	rs1		fun	ct3	imı	m[4:0]	opc	ode	S-type
im	m[12	10:5	5]	rs	2	rs1		fun	ct3	imm	[4:1 11]	opc	ode	B-type
				im	m[31	:12]					rd	opc	ode	U-type
			imn	n[20	10:1	11 19:12	2]				rd	opc	ode	J-type

x0: constant value 0

x1: return address

x2: stack pointer

x3: global pointer

x4: thread pointer

x5 - x7, x28 - x31: temporaries (np)

x8: frame pointer

x9, x18 - x27: saved registers

x10 – x11: function arguments/results (np)

x12 - x17: function arguments (np)

$$x=(-1)^S imes (1+ ext{Frac}) imes 2^{(ext{Expo-Bias})}$$

	Ü	denormalized	tloat	intinity	NaN	
Expo	0	D	1-254	255	255	
Frac	0	nonzero	anything	0	nonzero	_

CPU time = CPU clock cycles / Clock frequency

- Clock frequency (rate): cycles per second (*Hz*)
- Clock period: duration of a clock cycle (s)

 $CPU\ clock\ cycles = Instruction\ count \times Cycles\ per\ instruction\ (CPI)$

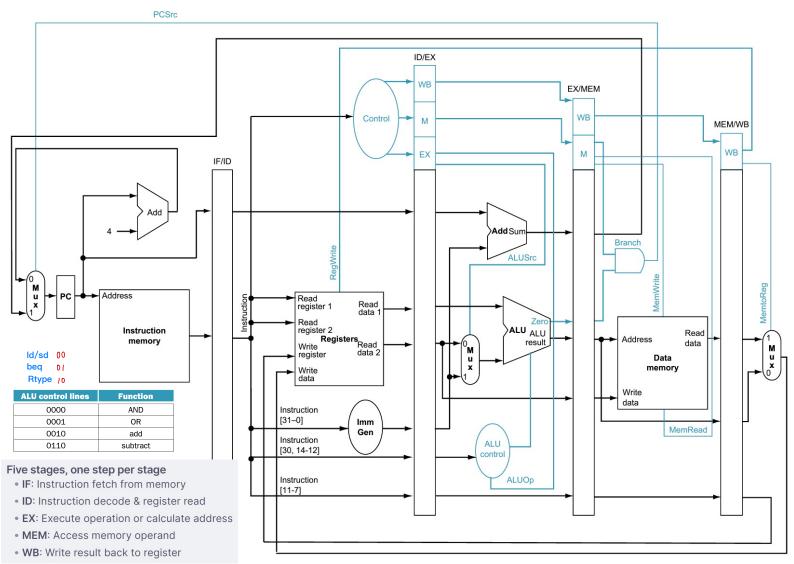
 $\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$

	Saving	registers
sort:	addi sp, sp, -20	# make room on stack for 5 registers
	sw x1, 16(sp)	# save return address on stack
	sw x22, 12(sp)	# save x22 on stack
	sw x21, 8(sp)	# save x21 on stack
	sw x20, 4(sp)	# save x20 on stack
	sw x19. $\Omega(sn)$	# save x19 on stack

	Proced	lure body
Move parameters	addi x21, x10, 0 addi x22, x11, 0	# copy parameter x10 into x21 # copy parameter x11 into x22
Outer loop	addi x19,x0, 0 for1tst:bge x19, x22, exit1	The state of the s
Inner loop	addi x20, x19, -1 for2tst:blt x20, x0, exit2 slli x5, x20, 2 add x5, x21, x5 lw x6, 0(x5) lw x7, 4(x5) ble x6, x7, exit2	# go to exit2 if j < 0
Pass parameters and call	addi x10, x21, 0 addi x11, x20, 0 jal x1, swap	# first swap parameter is v # second swap parameter is j # call swap
Inner loop	addi x20, x20, -1 jal, x0 for2tst	j for2tst ∦ go to for2tst
Outer loop	exit2: addi x19, x19, 1 jal, x0 for1tst	# i += 1 # go to for1tst

	Restoring	; registers
exit1:	lw x19, 0(sp)	# restore x19 from stack
	lw x20, 4(sp)	# restore x20 from stack
	lw x21, 8(sp)	# restore x21 from stack
	lw x22, 12(sp)	# restore x22 from stack
	lw x1, 16(sp)	# restore return address from stack
	addi sp, sp, 20	# restore stack pointer

	Procedure return
jalr x0, 0(x1)	# return to calling routine



azards

- Structural Hazard: conflict for use of a resource
- ightarrow pipelined datapaths need separate memories (caches)
- Data Hazard: instruction depends on previous instruction
- ex: add x19, x0, x1, sub x2, x19, x3
- \rightarrow forawarding: retrieving data from internal buffers (after ${\rm EX})$ instead of wait it to be stored in memory
- Load-Use Data Hazard: data loaded (at MEM) by load instruction have not yet become available when needed
- \rightarrow reorder code to avoid using load result in next instruction
- Control Hazards: fetching instruction depends on branch result
- ightarrow branch prediction: only stall if prediction is wrong
- \rightarrow static: based on typical branch behavior. ex: loop, if
- \rightarrow dynamic: record recent history of branch, when wrong, stall while re-fetching, and update history

Handle Exceptions

- Save PC of interrupted instruction → SEPC (can be use to return)
- \bullet Save indication of the problem \rightarrow SCAUSE
- Jump to handler \rightarrow 0000 0000 1C09 0000
- Vectored Interrupts: determine handler address by adding exception vector address to vector table base register
- when multiple exceptions:
 - precise: deal with exception from earliest instruction, flush subsequent
 - imprecise: stop pipeline and save state → complex handler software

Instruction-Level Parallelism

- pipelining: executing multiple instructions in parallel
- increase ILP: deeper pipeline → less work per stage → shorter cc
- static multiple issue: compiler group instructions into "issue slots" by pipeline resources required, hazards detected by compiler
 - RISC-V dual issue: ALU/branch then load/store, pad unused with nop
 - loop unrolling: replicate loop body with different registers
- dynamic multiple issue: CPU choose instructions to issue each cycle, compiler can help reorder instructions, hazards resolved by CPU at runtime
 - avoid structural & data hazards, compiler scheduling
 - dynamic scheduling: execute instructions out of order to avoid stalls
 - register renaming: by copy operand to reservation stations
 - speculation: guess the outcome of instructions then start corresponding operation, if guess is wrong then roll back

EX hazard

- if (EX/MEM.RegWrite
- and (EX/MEM.RegisterRd ≠ 0)
- and (EX/MEM.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 10
- if (EX/MEM.RegWrite
- and $(EX/MEM.RegisterRd \neq 0)$
- and (EX/MEM.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 10

MEM hazard

- if (MEM/WB.RegWrite
- and (MEM/WB.RegisterRd ≠ 0)
- and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
 - and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))
- and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01
- if (MEM/WB.RegWrite
- and $(MEM/WB.RegisterRd \neq 0)$
- and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd \neq 0)
 - and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))
- and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01

Load-use hazard

- if (ID/EX.MemRead and
 - ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
 - (ID/EX.RegisterRd = IF/ID.RegisterRs2)))
 - stall the pipeline (do nop)

	ALU or branch instruction	Data transfer instruction	Clock cycle
Loop:		lw x31, 0(x20)	1
	addi x20, x20, -4		2
	add x31, x31, x21		3
	blt x22, x20, Loop	sw x31, 4(x20)	4

	ALU or branch instruction	Data transfer instruction	Clock cycle
Loop:	addi x20, x20, -32	1w x28, 0(x20)	1
		lw x29, 12(x20)	2
	add x28, x28, x21	1w x30, 8(x20)	3
	add x29, x29, x21	lw x31, 4(x20)	4
	add x30, x30, x21	sw x28, 16(x20)	5
	add x31, x31, x21	sw x29, 12(x20)	6
		sw x30, 8(x20)	7
	blt x22, x20, Loop	sw x31, 4(x20)	8

_	21		J. D.	ompare to loop limit,	x22
or-array erement	dd scalar in x21	lt.	ecrement pointer	loop	<20 >
l ay	calar	tore result	nent p	re to	ranch if x20 >
p_Tc	og pp	tore	ecre	ompai	ranc

Mux control

ForwardA = 00

ForwardA = 10

ForwardA = 01

ForwardB = 00

ForwardB = 10

ForwardB = 01

ID/FX

ID/EX

EX/MEM

MEM/WB

FX/MFM

MEM/WB

:doo	3	x31,	0(x20)	_	_	x31
	add	x31,	x31, x2	1	_	add
	SW	x31,	0(x20)		_	sto
	addi	x20,	x20, -4		1	dec
	blt	x22,	x20, Loop	Loop /	_	COM
					1	bra