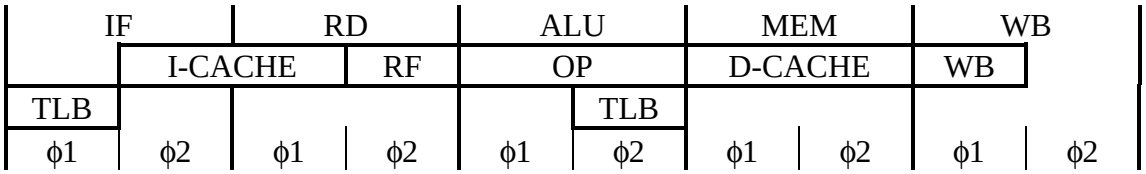
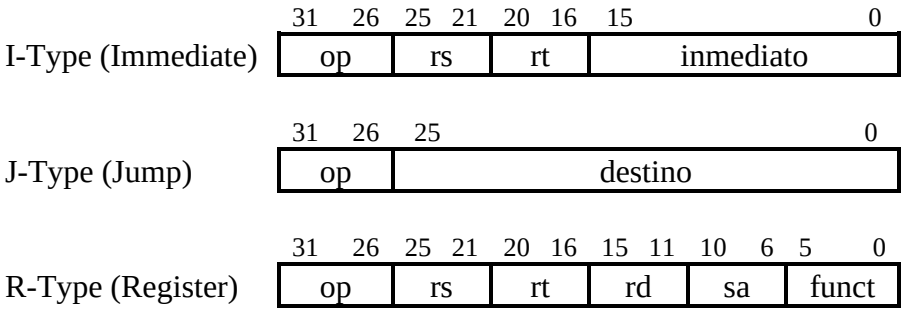
PIPELINE DEL R3000



Etapa	Fase	Acciones
IF	ϕ1	Traducción de dirección virtual a física (tras decisión de salto en ALU ϕ1).
	ϕ2	Envío de la dirección física a la caché de instrucciones.
RD	ϕ1	Lectura de la instrucción desde la caché. Comparación tags y paridad.
	ϕ2	Acceso al fichero de registros. Cálculo de dirección de destino en saltos.
ALU		Operación de la ALU (primera fase). Alineación de datos en stores (primera fase). Cálculo de la dirección efectiva en load y store. Determinación de la condición en saltos.
	ϕ2	Operación de la ALU (segunda fase). Alineación de datos en stores (segunda fase). Traducción de dirección virtual a física en load y store.
MEM	ϕ1	Envío de la dirección física a la caché de datos en load y store.
	ϕ2	Lectura de la caché de datos en load. Comparación de tags y paridad. Alineación de datos en loads.
WB	ϕ1	Escritura en el banco de registros.
	ϕ2	

UNIDADES FUNCIONALES		
BTA	IF ϕ1 - RD ϕ2	Incrementa PC - Calcula la dirección de destino en saltos.
TLB	IF ϕ1 - ALU ϕ2	Traduce direcciones virtuales a físicas.
I-CACHE	RD ϕ1 - IF ϕ2	Caché de instrucciones.
D-CACHE	MEM ϕ1 - MEM ϕ2	Caché de datos.
CACHE_LOGIC	RD ϕ1 - MEM ϕ2	Comapra tags y paridad.
ALU	ALU ϕ1 - ALU ϕ2	ALU - Calcula direcciones efectivas en loads y stores
SHIFTER	ALU ϕ1 - ALU ϕ2	Desplaza - Alinea operandos en stores
ALIGN	MEM ϕ2	Alinea operandos en loads

INSTRUCCIONES Y FORMATO



Instrucción	Load y Store. Tipo inmediato. rs = base, rt = destino
Load Byte	LB rt,offset(base) --- op = 100000 rt = sign_ext(MEM8[base + sign_ext(offset)])
Load Byte Unsigned	LBU rt,offset(base) --- op = 100100 rt = zero_ext(MEM8[base + sign_ext(offset)])
Load Halfword	LH rt,offset(base) --- op = 100001 rt = sign_ext(MEM16[base + sign_ext(offset)])
Load Halfword Unsigned	LHU rt,offset(base) --- op = 100101 rt = zero_ext(MEM16[base + sign_ext(offset)])
Load Word	LW rt,offset(base) --- op = 100011 rt = MEM32[base + sign_ext(offset)]
Load Word Left	LWL rt,offset(base) --- op = 100010 rt = merge(rt,left(MEM32[base + sign_ext(offset)]))
Load Word Right	LWR rt,offset(base) --- op = 100110 rt = merge(rt,right(MEM32[base + sign_ext(offset)]))
Store Byte	SB rt,offset(base) --- op = 101000 MEM8[base + sign_ext(offset)] = rt
Store Halfword	SH rt,offset(base) --- op = 101001 MEM16[base + sign_ext(offset)] = rt
Store Word	SW rt,offset(base) --- op = 101011 MEM32[base + sign_ext(offset)] = rt
Store Word Left	SWL rt,offset(base) --- op = 101010 MEM32[base + sign_ext(offset)] = merge(left(rt),MEM32[base + sign_ext(offset)])
Store Word Right	SWR rt,offset(base) --- op = 101110 MEM32[base + sign_ext(offset)] = merge(right(rt),MEM32[base + sign_ext(offset)])

Instrucción	ALU inmediato. Tipo inmediato. rs = fuente, rt = destino
Add Immediate	ADDI rt,rs,immediate --- op = 001000 rt = rs + sign_ext(immediate)
Add Immediate Unsigned	ADDIU rt,rs,immediate --- op = 001001 rt = rs + sign_ext(immediate)
Set on Less Than Immediate	SLTI rt,rs,immediate --- op = 001010 rt = (rs < sign_ext(immediate))
Set on Less Than Immediate Unsigned	SLTIU rt,rs,immediate --- op = 001011 rt = ((unsigned)rs < (unsigned)sign_ext(immediate))
And Immediate	ANDI rt,rs,immediate --- op = 001100 rt = rs & zero_ext(immediate)
Or Immediate	ORI rt,rs,immediate --- op = 001101 rt = rs | zero_ext(immediate))

Exclusive Or Immediate	XORI rt,rs,immediate --- op = 001110 rt = rs ^ zero_ext(immediate)
Load Upper Immediate	LUI rtimmediate --- op = 001111 rt = immediate<<16

Instrucción	ALU registros. Tipo registro. rs = fuente, rt = fuente, rd = destino. op = 0, sa = 0
Add	ADD rd,rs,rt --- funct = 100000 rd = rs + rt
Add Unsigned	ADDU rd,rs,rt --- funct = 100001 rd = rs + rt
Substract	SUB rd,rs,rt --- funct = 100010 rd = rs - rt
Substract Unsigned	SUBU rd,rs,rt --- funct = 100011 rd = rs - rt
Set on Less Than	SLT rd,rs,rt --- funct = 101010 rd = (rs < rt)
Set on Less Than Unsigned	SLTU rd,rs,rt --- funct = 101011 rd = ((unsigned)rs < (unsigned)rt)
And	AND rd,rs,rt --- funct = 100100 rd = rs & rt
Or	OR rd,rs,rt --- funct = 100101 rd = rs | rt
Exclusive Or	XOR rd,rs,rt --- funct = 100110 rd = rs ^ rt
Nor	NOR rd,rs,rt --- funct = 100111 rd = !(rs | rt)

Instrucción	Desplazamientos. Tipo registro. rt = fuente, rd = destino. op = 0
Shift Left Logical	SLL rd,rt,sa --- funct = 000000 rd = rt << sa
Shift Right Logical	SRL rd,rt,sa --- funct = 000010 rd = rt >> sa
Shift Right Arithmetic	SRA rd,rt,sa --- funct = 000011 rd = sign_ext(rt >> sa)
Shift Left Logical Variable	SLLV rd,rt,rs --- funct = 000100, sa = 00000 rd = rt << (rs & 1F)
Shift Right Logical Variable	SRLV rd,rt,rs --- funct = 000110, sa = 00000 rd = rt >> (rs & 1F)
Shift Right Arithmetic Variable	SRAV rd,rt,rs --- funct = 000111, sa = 00000 rd = sign_ext(rt >> (rs & 1F))

Instrucción	Saltos absolutos. Tipo salto.
Jump	J destino --- op = 000010 PC = PC & (F0000000 + destino<<2)
Jump And Link	JAL destino --- op = 000011 r31 = address(next(slot)), PC = PC & (F0000000 + destino<<2)

Instrucción	Saltos absolutos a registros. Tipo registro. rs = direccion destino del salto op = 0
Jump Register	JR rs --- funct = 001000 PC = rs
Jump And Link Register	JALR rs,rd --- funct = 001001 rd = address(next(slot)), PC = rs

Instrucción	Saltos relativos condicionales. Tipo inmediato.
Branch on Equal	BEQ rs,rt,offset --- op = 000100 si (rs = rt) PC = address(slot) + sign_ext(offset)
Branch on Not Equal	BNE rs,rt,offset --- op = 000101 si (rs != rt) PC = address(slot) + sign_ext(offset)
Branch on Less than or Equal Zero	BLEZ rs,offset --- op = 000110, rt = 00000 si (rs <= 0) PC = address(slot) + sign_ext(offset)
Branch on Greater Than Zero	BGTZ rs,offset --- op = 000111, rt = 00000 si (rs > 0) PC = address(slot) + sign_ext(offset)
Branch on Less Than Zero	BLTZ rs,offset --- op = 000001, rt = 00000 si (rs < 0) PC = address(slot) + sign_ext(offset)
Branch on Greater than or Equal Zero	BGEZ rs,offset --- op = 000001, rt = 00001 si (rs >= 0) PC = address(slot) + sign_ext(offset)
Branch on Less Than Zero And Link	BLTZAL rs,offset --- op = 000001, rt = 10000 r31 = address(next(slot)), si (rs < 0) PC = Address(slot) + sign_ext(offset)
Branch on Greater than or Equal Zero And Link	BGEZAL rs,offset --- op = 000001, rt = 10001 r31 = address(next(slot)), si (rs >= 0) PC = Address(slot) + sign_ext(offset)

Instrucción	Multiplicación y división. Tipo registro. rs, rt = fuente, rd = destino. op = 0, sa = 0
Multiply	MULT rs,rt --- funct = 011000 --- ciclos = 12 HI - LO = rs * rt
Multiply Unsigned	MULTU rs,rt --- funct = 011001 --- ciclos = 12 HI - LO = rs * rt (unsigned)
Divide	DIV rs,rt --- funct = 011010 --- ciclos = 35 LO = rs / rt, HI = rs % rt
Divide Unsigned	DIVU rs,rt --- funct = 011011 --- ciclos = 35 LO = rs / rt, HI = rs % rt (unsigned)
Move From HI	MFHI rd --- funct = 010000 rd = HI
Move From LO	MFLO rd --- funct = 010010 rd = LO
Move To HI	MTHI rd --- funct = 010001 HI = rd
Move To LO	MTLO rd --- funct = 010011 LO = rd

IF (Adquisición de Instrucción)	
ϕ1	**ϕ2**
PC_acceso <- PC PC <- PC+4	IMAR <- PC_acceso·¬cond + PC_bta·cond PC <- PC·¬cond + (PC_bta + 4)·cond

RD (Lectura)	
ϕ1	**ϕ2**
IR <- ICACHE[IMAR]	ALU_a <- RS·¬link + PC·link ALU_b <- RD·R-Type + IMM·I-Type·¬br + RT·I-Type·br2 .....+ R0·I-Type·br1 + R0·J-Type PC_bta <- RD·(op=R-Type) + PC+IMM·(op=I-Type) + (PC||destino)·(op=J-Type) SMDR_1 <- RT RS_a1 <- #RS·¬(op=link) + 0·(op=link) RS_b1 <- #RD·(op=R-Type) + 0·¬(op=R-Type) op_1 <- op func_1 <- func·(op=R-Type)·¬(op=link) + op·(op=I-Type)·¬(op=link) + #OR·(op=link) + branch

IF
(Adquisición de instrucción)

PC <- PC + 4 IR <- IMEM[PC]
ó (ALU_OUT & IR2)
PC_BTA1

RD
(Lectura de operandos)

PC1 <- PC PC_BTA <- Imm_L(IR)
ó (IR)
IR1 <- IR SIGN(Imm_C(IR))+PC

ALU_A <- Rs1
ALU_B <- Rs2

ALU
(Ejecución)

PC2 <- PC1

IR2 <- IR1

PC_BTA1 <- PC_BTA

SMDR <- ALU_B

DMAR <- ALU_A + SIGN(Imm_C(IR1))
ó (IR1)

ALU_OUT <- ALU_A op (IR1)
ALU_B
ó (IR1)
Imm_C(IR1)
ó (IR1)

SHIFT_OUT <- ALU_A op (IR1)
ALU_B
ó (IR1)
Imm_SA(IR1)