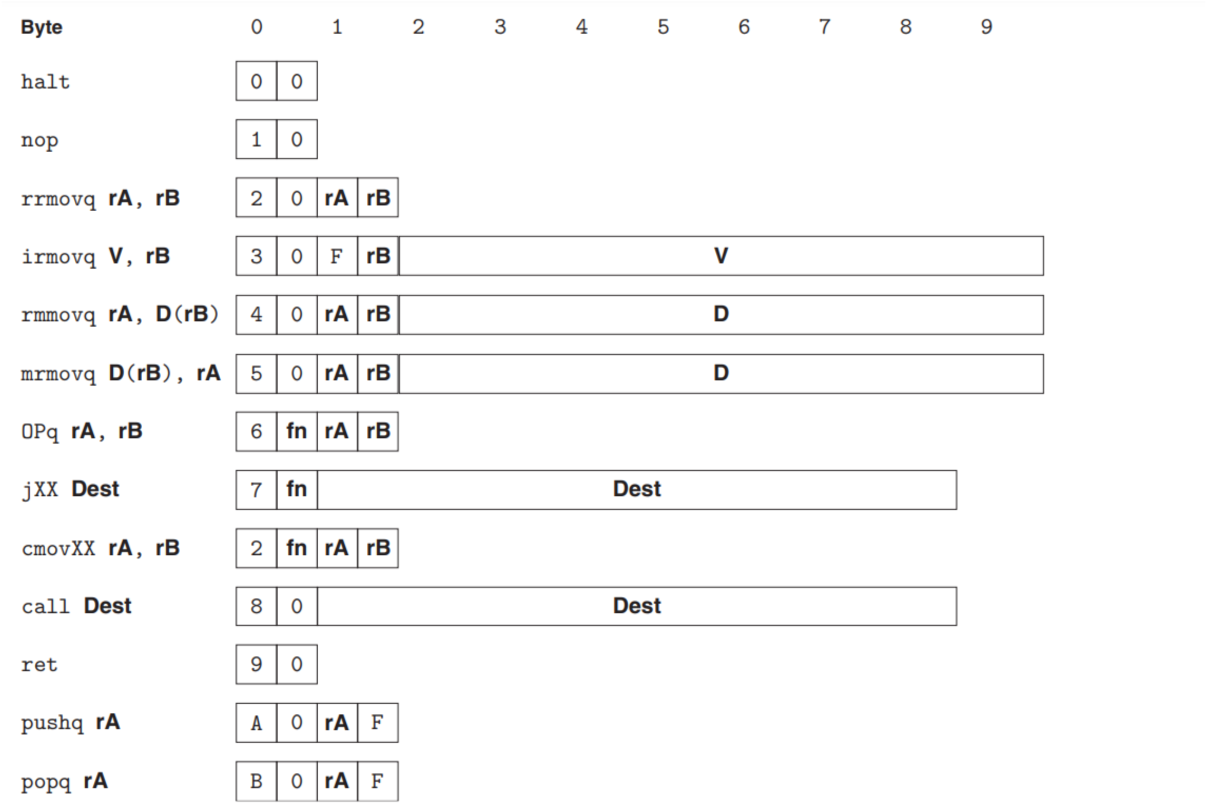
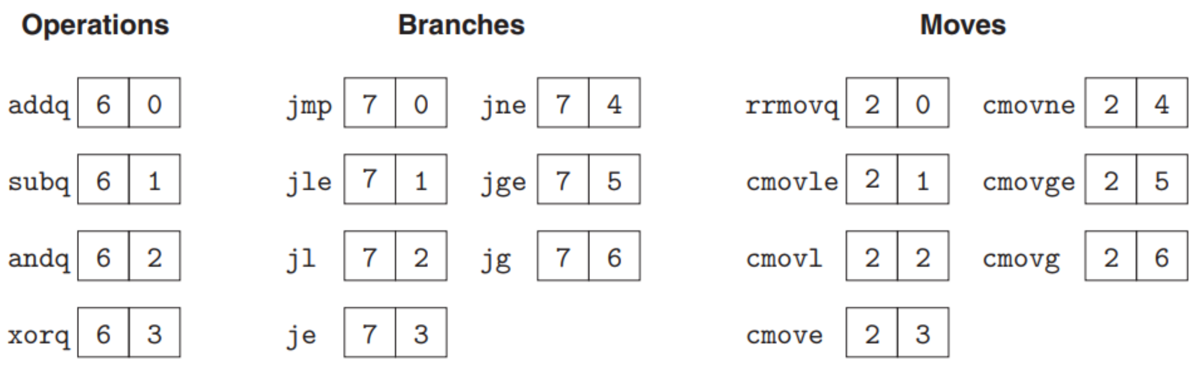


Byte	0	1	2	3	4	5	6	7	8	9
`halt`	0 0									
`nop`	1 0									
`rrmovq` **rA**, **rB**	2 0	rA rB								
`irmovq` **V**, **rB**	3 0	F rB	V							
`rmmovq` **rA**, **D**(**rB**)	4 0	rA rB	D							
`mrmovq` **D**(**rB**), **rA**	5 0	rA rB	D							
`OPq` **rA**, **rB**	6 fn	rA rB								
`jXX` **Dest**	7 fn	Dest								
`cmovXX` **rA**, **rB**	2 fn	rA rB								
`call` **Dest**	8 0	Dest								
`ret`	9 0									
`pushq` **rA**	A 0	rA F								
`popq` **rA**	B 0	rA F								



Operations			Branches						Moves					
`addq`	6	0	`jmp`	7	0	`jne`	7	4	`rrmovq`	2	0	`cmovne`	2	4
`subq`	6	1	`jle`	7	1	`jge`	7	5	`cmovle`	2	1	`cmovge`	2	5
`andq`	6	2	`jl`	7	2	`jg`	7	6	`cmovl`	2	2	`cmovg`	2	6
`xorq`	6	3	`je`	7	3				`cmove`	2	3			

This byte is split into two 4-bit parts: the high-order, or code, part, and the low-order, or function, part.

INSTRUCTIONS SETS OF Y86-64 PROCESSOR

INSTRUCTION	OPCODE	DESCRIPTION
halt	00	Stop program execution
nop	10	Do nothing for one cycle
rrmovq	20	Move register to regisrter
cmovle	21	Conditional move if less than or equal
cmov1	22	Conditional move if less than
cmove	23	Conditional move if equal
cmovne	24	Conditional move if not equal
cmovge	25	Conditional move if greater than or equal
cmovg	26	Conditional move if greater
irmovq	30	Move immediate to register
rmmovq	40	Move register to memory
mrmovq	50	Move memory to register
addq	60	Add rA to rB
subq	61	Subtract rA from rB
andq	62	Bitwise and rA and rB
xorq	63	Bitwise xor of rA and rB
Jmp	70	Unconditional jump to dest
Jle	71	Jump if less than or equL TO Dest
J1	72	Jump if less than dest
je	73	Jump if equal to dest
jne	74	Jump if not equal to dest
Jge	75	Jump if greater than or equal to dest
jg	76	Jump if greater than dest
call	80	Call subroutine at dest
ret	90	Return from subroutine
pushq	A0	Push register rA onto the stack
popq	B0	Pop top of stack into register rA

Y86 -64 Registers

Register Name	Register code	Description
%rax	0	General-purpose register ,often used for function return values (analogous to rax in x86-64)
%rcx	1	General -purpose register, often used as a counter or for loop iterations
%rdx	2	General -purpose register, typically used for input /output operations or as an additional counter
%rbx	3	General -purpose register, sometimes reserved for base pointers or other program data
%rsp	4	Stack pointer register, points to the top of the stack in memory. required for pushq and popq operations
%rbp	5	Base pointer register, used for stack frame references ,though not strictly required in Y86-64 programs
%rsi	6	General- purpose register often used to pass arguments to functions or for source indexing
%rdi	7	General- purpose register commonly used for functions arguments or destination indexing
%r8	8	Additional general -purpose registers like R8 in x 86-64
%r9	9	Additional general -purpose registers, similar to R9 in X 86-64
None	F	Represents no register (used in instructions where a register operand is not applicable)

Some examples

Instruction: rrmovq %r1, %r2

Encoding: 20 12

Instruction: irmovq $10, %r3

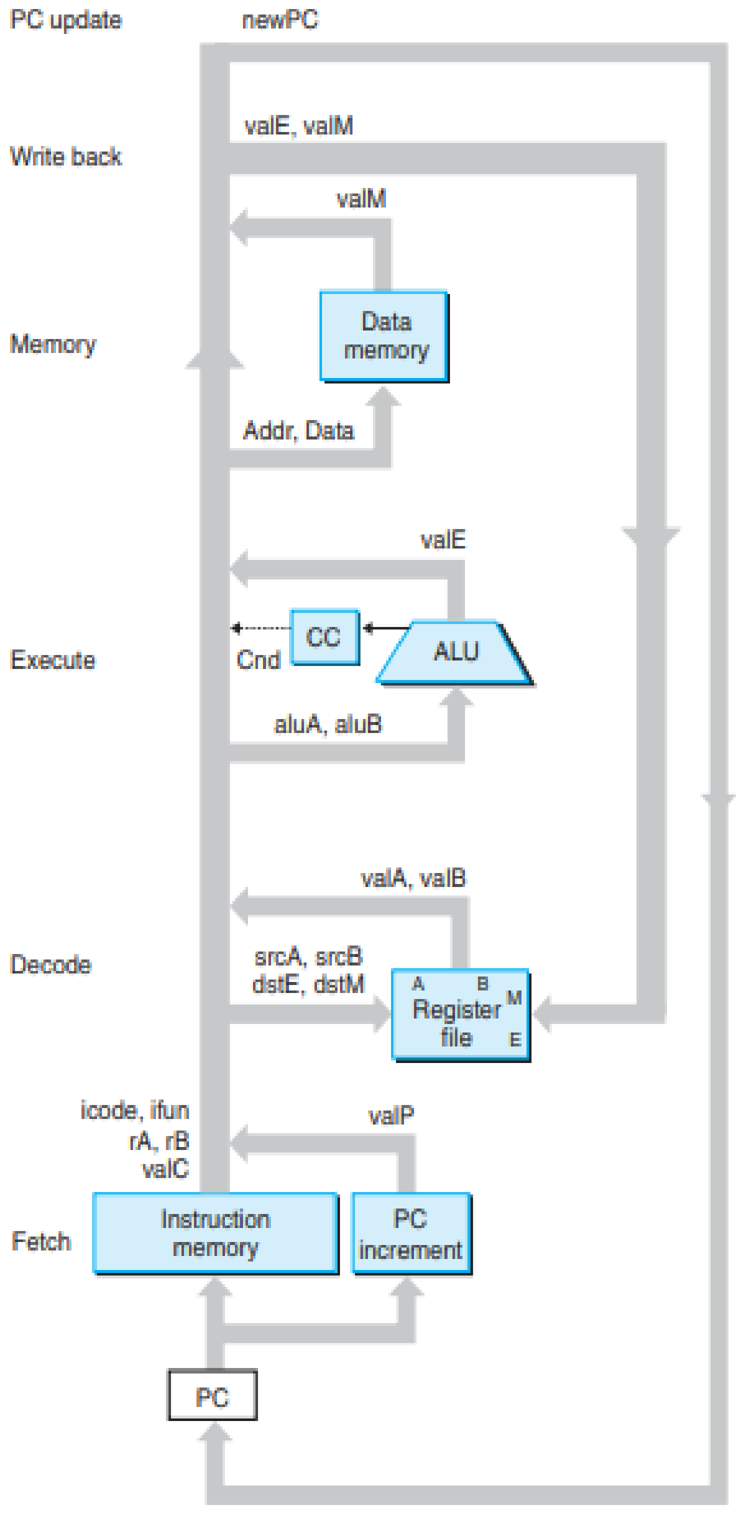
Encoding: 30 F3 000000000000000A

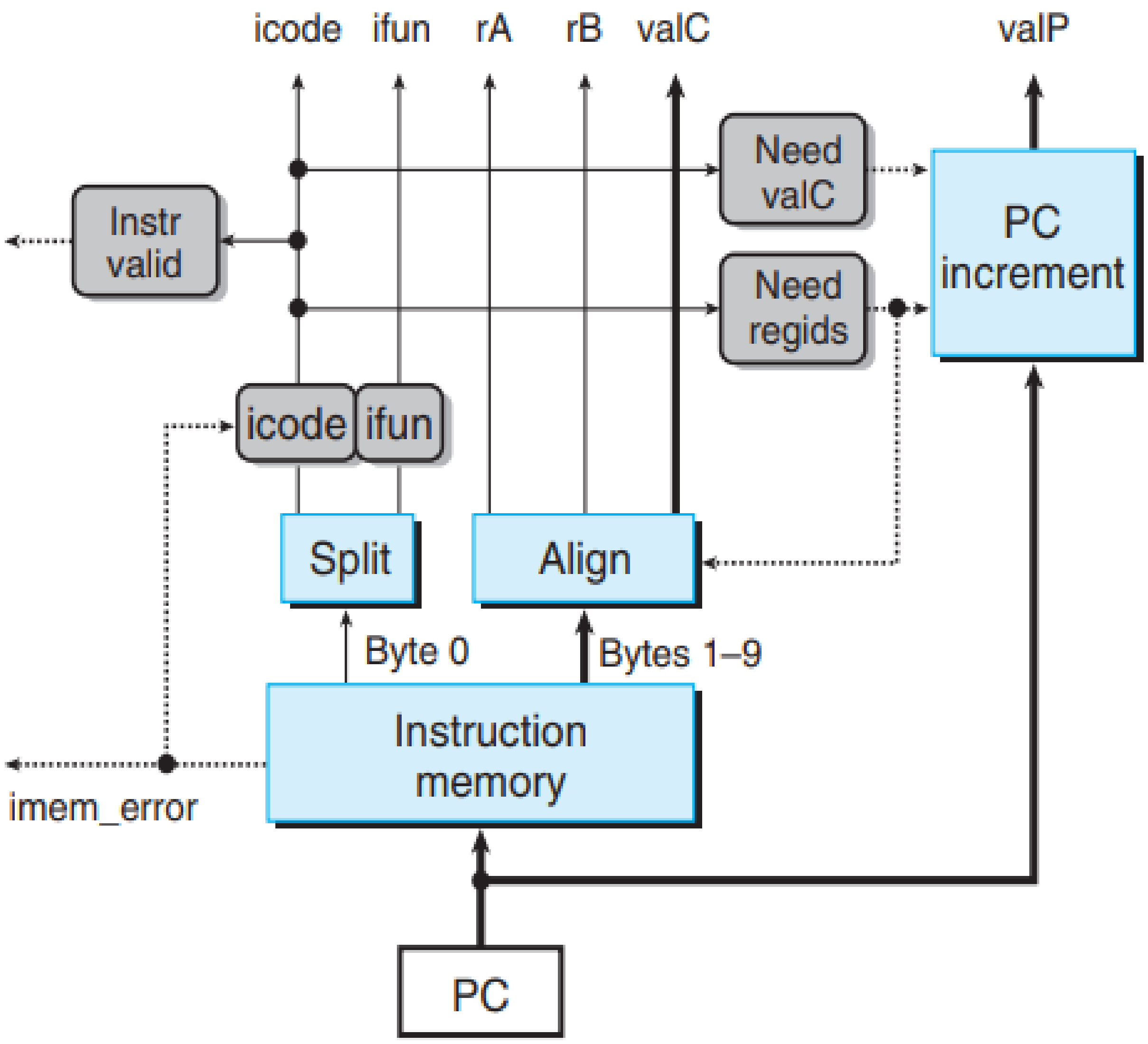
Instruction: rmmovq %r4, 8(%r5)

Encoding: 40 45 00000008

Instruction: cmovle %r10, %r11

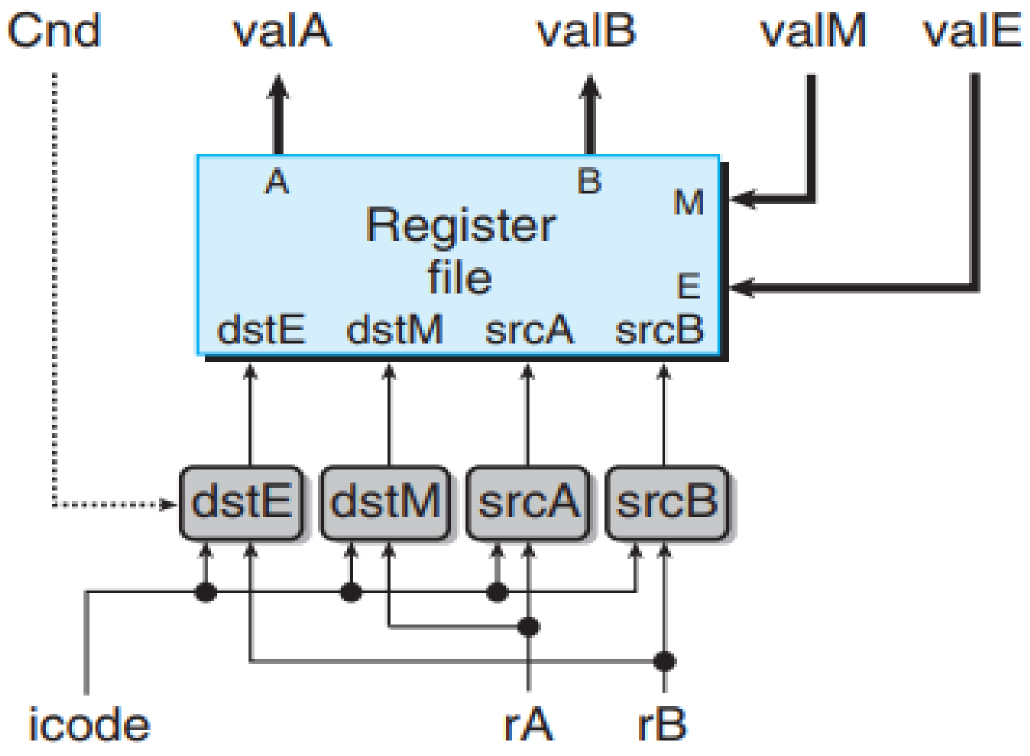
Encoding: 21 AB


PC update
newPC
Write back
valE, valM
valM
Memory
Data memory
Addr, Data
valE
Execute
CC
ALU
Cnd
aluA, aluB
valA, valB
Decode
srcA, srcB
dstE, dstM
A
B
M
Register file
E
icode, ifun
rA, rB
valC
valP
Fetch
Instruction memory
PC increment
PC



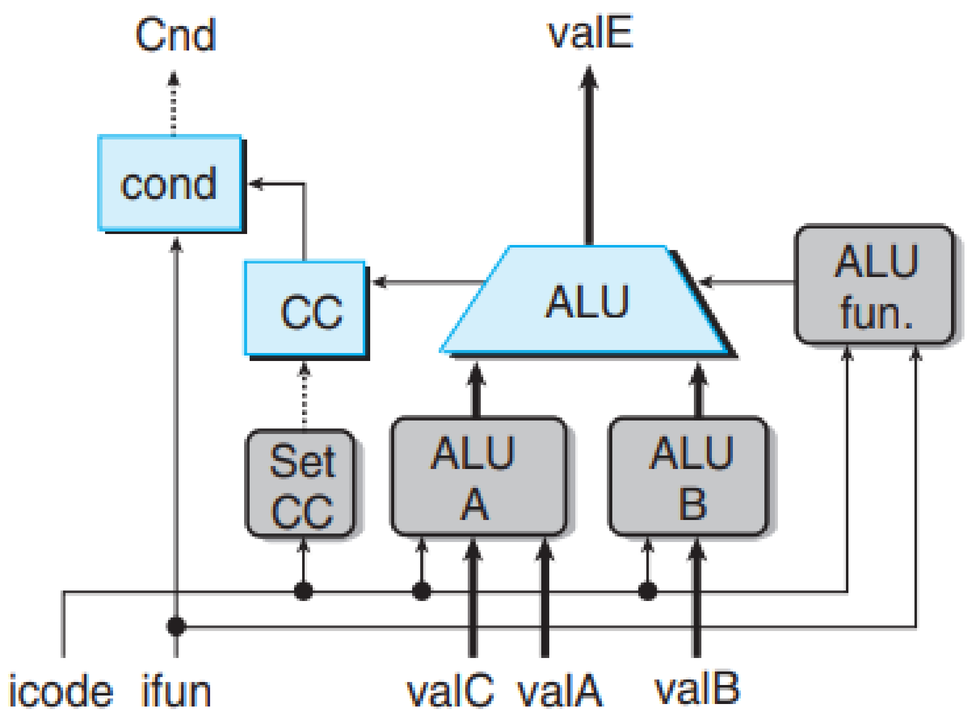
Figure

SEQ fetch stage



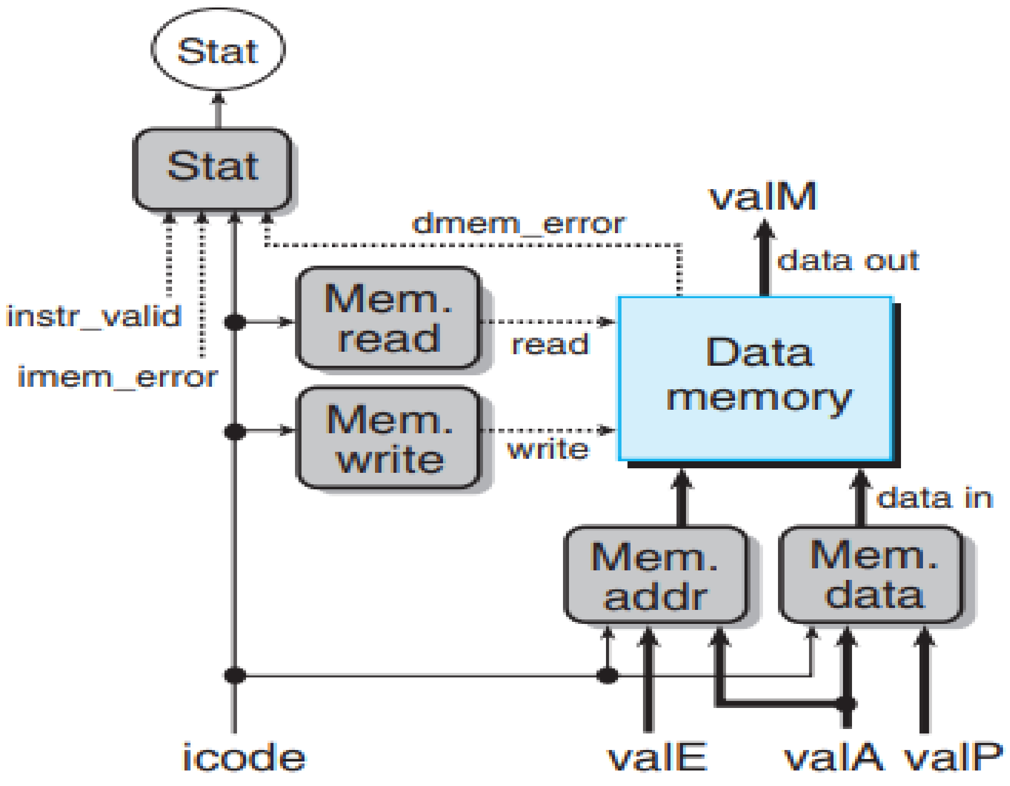
Figure

SEQ decode and write-back stage. The instruction fields are decoded to generate register identifiers for four addresses (two read and two write) used by the register file. The values read from the register file become the signals valA and valB. The two write-back values valE and valM serve as the data for the writes.



Figure

SEQ execute stage. The ALU either performs the operation for an integer operation instruction or acts as an adder. The condition code registers are set according to the ALU value. The condition code values are tested to determine whether a branch should be taken.



Figure

SEQ memory stage. The data memory can either write or read memory values. The value read from memory forms the signal valM.

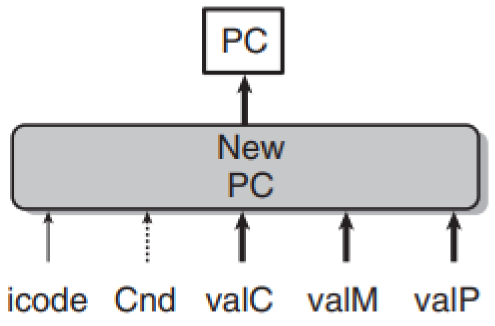


Figure 4.31 SEQ PC update stage. The next value of the PC is selected from among the signals valC, valM, and valP, depending on the instruction code and the branch flag.

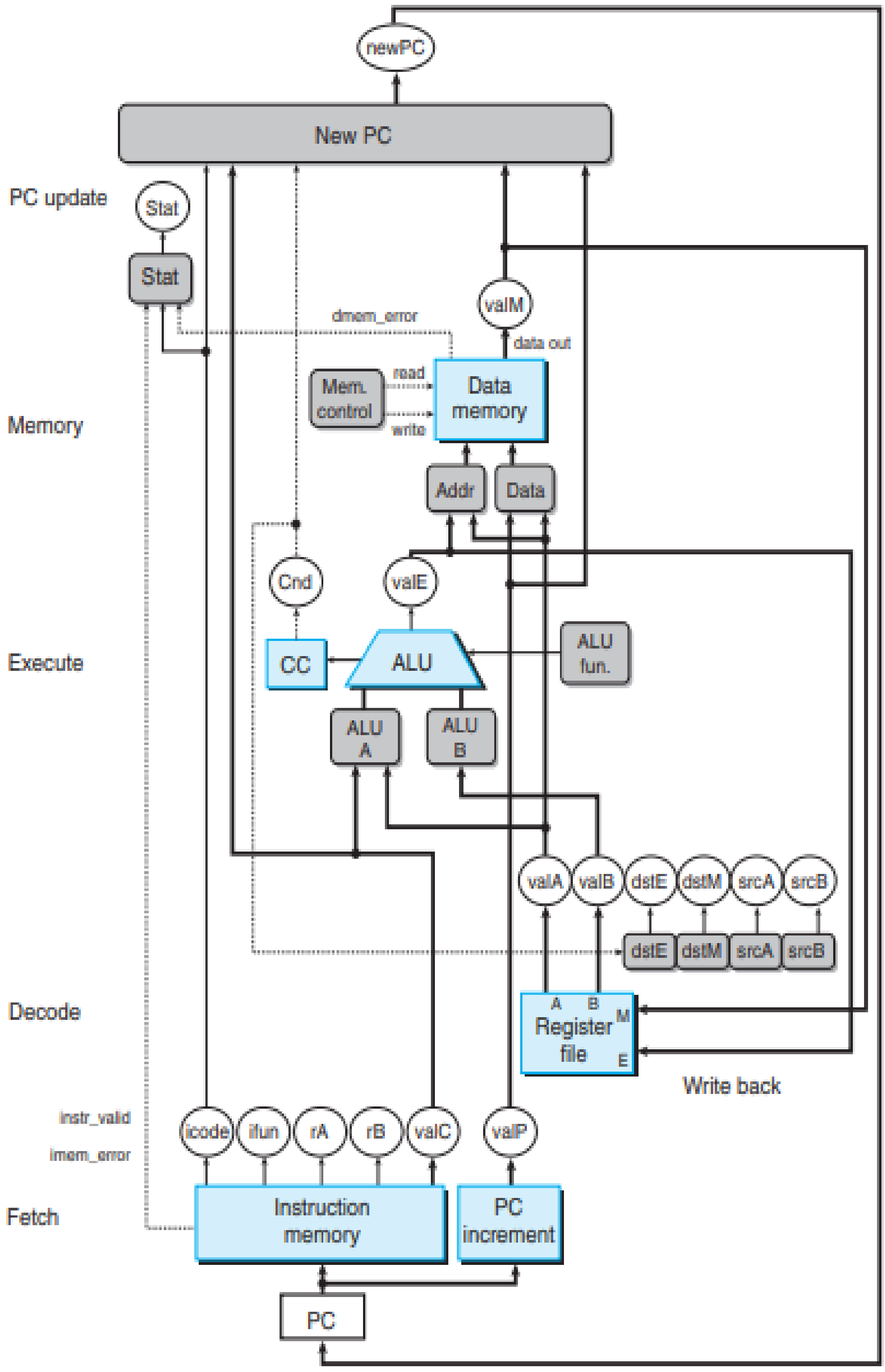
Example

Stage	`OPq rA, rB`	`rrmovq rA, rB`	`irmovq V, rB`

Fetch	icode:ifun ← M_1[PC]
	rA:rB ← M_1[PC + 1]
	
	valP ← PC + 2
		icode:ifun ← M_1[PC]
		rA:rB ← M_1[PC + 1]
		
		valP ← PC + 2
			icode:ifun ← M_1[PC]
			rA:rB ← M_1[PC + 1]
			valC ← M_8[PC + 2]
			valP ← PC + 10

Decode	valA ← R[rA]
	valB ← R[rB]
		valA ← R[rA]	

Execute	valE ← valB OP valA Set CC	valE ← 0 + valA	valE ← 0 + valC
Memory			
Write back	R[rB] ← valE	R[rB] ← valE	R[rB] ← valE
PC update	PC ← valP	PC ← valP	PC ← valP


newPC
New PC
PC update
Stat
Stat
dmem_error
valM
data out
Memory
Mem. control
read
write
Data memory
Addr
Data
Cnd
valE
Execute
CC
ALU
ALU fun.
ALU A
ALU B
valA
valB
dstE
dstM
srcA
srcB
dstE
dstM
srcA
srcB
Decode
A
B
M
Register file
E
Write back
instr_valid
imem_error
icode
ifun
rA
rB
valC
valP
Fetch
Instruction memory
PC increment
PC