Lawrence Naidu
HW #5
Collaborate with Sergio Zavala

1.

A

if_icmpeq1	MAR = SP = SP – 1; rd	Read in next-to-top world of stack
if_icmpeq2	MAR = SP = SP – 1	Set MAR to read in new top-of-stack
if_icmpeq3	H = MDR; rd	Copy second stack word to H
if_icmpeq4	OPC = TOS	Save TOS in OPC temporarily
if_icmpeq5	TOS = MDR	Put new top of stack in TOS
if_icmpeq6	Z = OPC - H	If top 2 words are equal, goto T, else F

B TOS will be replaced in the next cycle, so the content needs to be saved in a temporary cycle.
If note copied into cycle 6, we use TOS instead of OPC in cycle 6, which will give us a wrong calculation.

2.

invokevirtual1	PC = PC + 1; fetch	MBR = index byte 1; inc. PC, get 2nd byte
invokevirtual2	H = MBRU << 8	Shift and save first byte in H
invokevirtual3	H = MBRU OR H	H = offset of method pointer from CPP
invokevirtual4	MAR = CPP + H; rd	Get pointer to method from CPP area
invokevirtual5	OPC = PC + 1	Save Return PC in OPC temporarily
invokevirtual6	PC = MDR; fetch	PC points to new method; get param count
invokevirtual7	PC = PC + 1; fetch	Fetch 2nd byte of parameter count
invokevirtual8	H = MBRU << 8	Shift and save first byte in H
invokevirtual9	H = MBRU OR H	H = number of parameters
invokevirtual10	PC = PC + 1; fetch	Fetch first byte of # locals
invokevirtual11	TOS = SP – H	TOS = address of OBJREF – 1
invokevirtual12	TOS = MAR = TOS + 1	TOS = address of OBJREF (new LV)
invokevirtual13	PC = PC + 1; fetch	Fetch second byte of # locals
invokevirtual14	H = MBRU << 8	Shift and save first byte in H
invokevirtual15	H = MBRU OR H	H = # locals
invokevirtual16	MDR = SP + H + 1; wr	Overwrite OBJREF with link pointer
invokevirtual17	MAR = SP = MDR;	Set SP, MAR to location to hold old PC
invokevirtual18	MDR = OPC; wr	Save old PC above the local variables
invokevirtual19	MAR = SP = SP + 1	SP points to location to hold old LV
invokevirtual20	MDR = LV; wr	Save old LV above saved PC
invokevirtual21	PC = PC + 1; fetch	Fetch first opcode of new method.
invokevirtual22	LV = TOS; goto Main1	Set LV to point to LV Frame

Takes 21 instructions to call one integer, therefore we need a loop to execute 2 more loops. Total instructions needed to execute 3 parameters will be (21 * 3 = 63 instructions + 2 for loops instructions = 65 instructions).

$CPU_{time} = \frac{\#\ of\ instructions * CPI}{clock\ rate}$

However, CPI and clock rate is unknown so we can't determine CPU_{time}