

4 THE MICROARCHITECTURE LEVEL

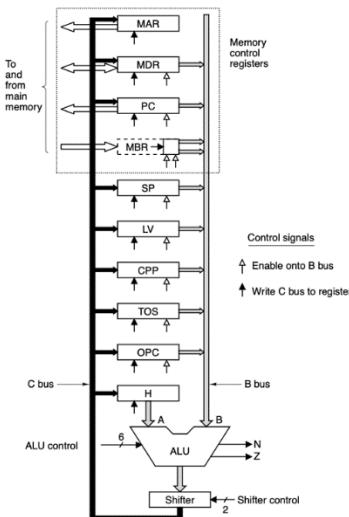


Figure 4-1. The data path of the example microarchitecture used in this chapter.

F ₀	F ₁	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	\bar{A}
1	0	1	1	0	0	\bar{B}
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B - A
1	1	0	1	1	0	B - 1
1	1	1	0	1	1	-A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
0	1	0	0	1	1	1
0	1	0	0	1	0	-1

Figure 4-2. Useful combinations of ALU signals and the function performed.

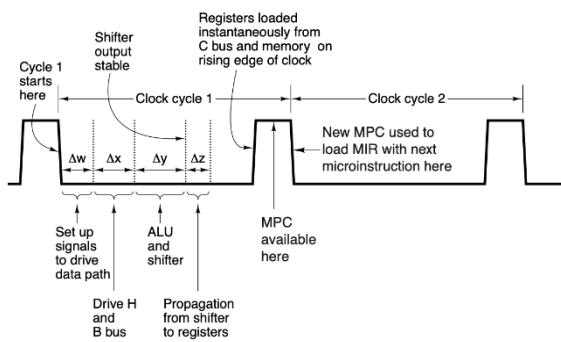


Figure 4-3. Timing diagram of one data path cycle.

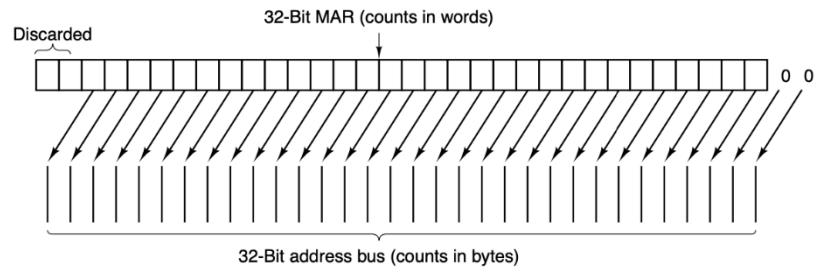


Figure 4-4. Mapping of the bits in MAR to the address bus.

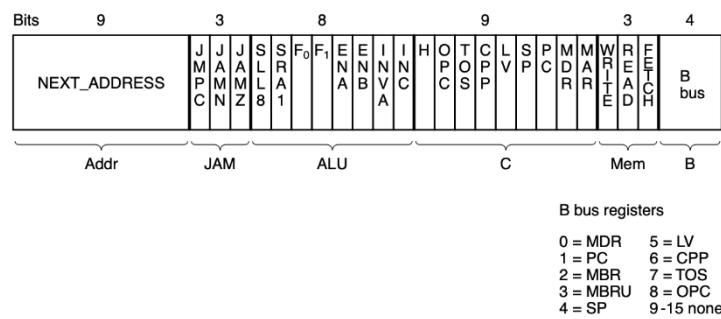


Figure 4-5. The microinstruction format for the Mic-1.

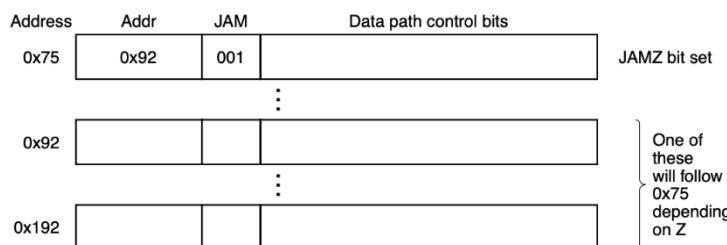


Figure 4-7. A microinstruction with JAMZ set to 1 has two potential successors.

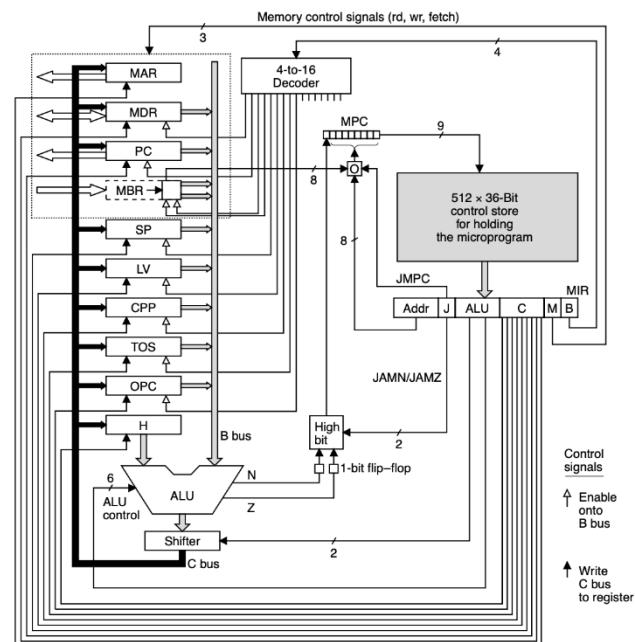


Figure 4-6. The complete block diagram of our example microarchitecture, the Mic-1.

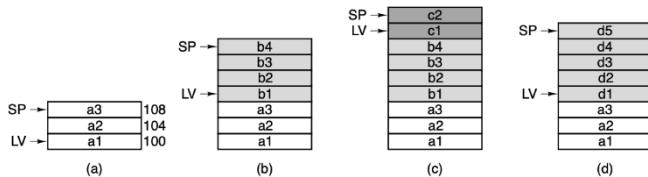


Figure 4-8. Use of a stack for storing local variables. (a) While A is active. (b) After A calls B. (c) After B calls C. (d) After C and B return and A calls D.

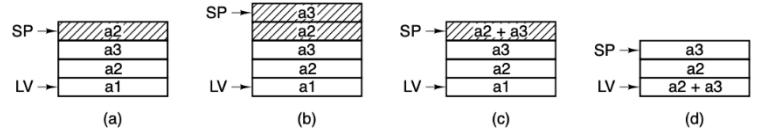


Figure 4-9. Use of an operand stack for doing an arithmetic computation.

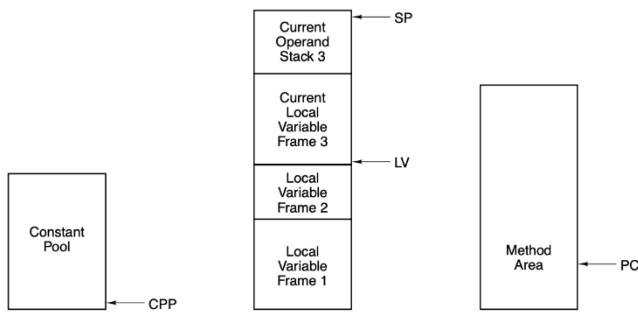


Figure 4-10. The various parts of the IJVM memory.

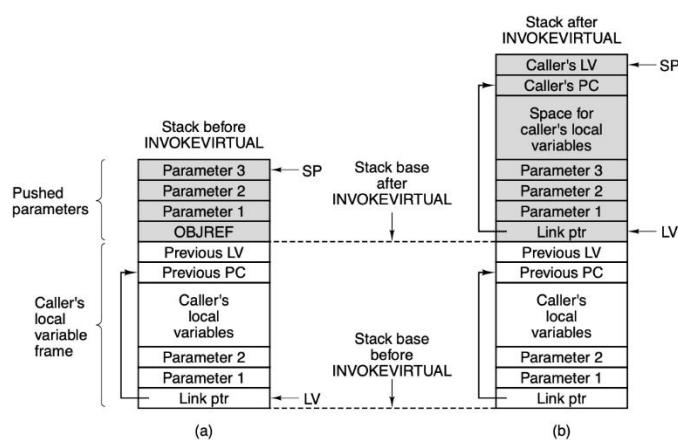


Figure 4-12. (a) Memory before executing `INVOKEVIRTUAL`. (b) After executing it.

Hex	Mnemonic	Meaning
0x10	BIPUSH byte	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO offset	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ offset	Pop word from stack and branch if it is zero
0x9B	IFLT offset	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ offset	Pop two words from stack; branch if equal
0x84	IINC varnum const	Add a constant to a local variable
0x15	ILOAD varnum	Push local variable onto stack
0xB6	INVOKEVIRTUAL disp	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE varnum	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W index	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

Figure 4-11. The IJVM instruction set. The operands *byte*, *const*, and *varnum* are 1 byte. The operands *disp*, *index*, and *offset* are 2 bytes.

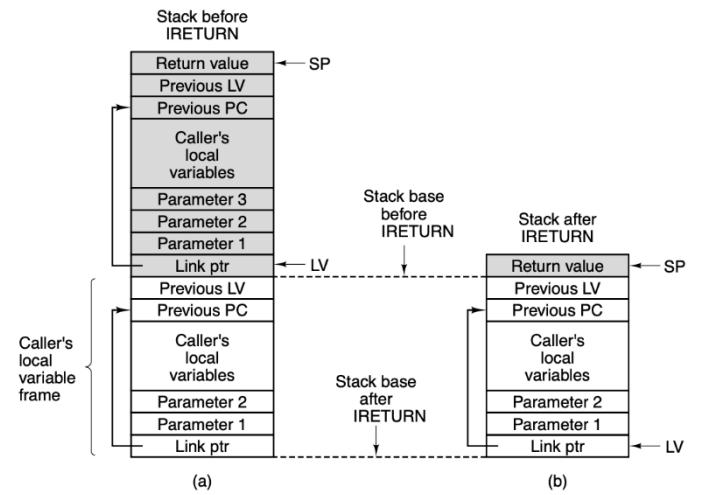


Figure 4-13. (a) Memory before executing `IRETURN`. (b) After executing it.

i = j + k;	1	ILOAD j // i = j + k	0x15 0x02
if (i == 3)	2	ILOAD k	0x15 0x03
k = 0;	3	IADD	0x60
else	4	ISTORE i	0x36 0x01
j = j - 1;	5	ILOAD i // if (i < 3)	0x15 0x01
	6	BIPUSH 3	0x10 0x03
	7	IF_ICMPEQ L1	0x9F 0x00 0x0D
	8	ILOAD j // j = j - 1	0x15 0x02
	9	BIPUSH 1	0x10 0x01
	10	ISUB	0x64
	11	ISTORE j	0x36 0x02
	12	GOTO L2	0xA7 0x00 0x07
	13 L1:	BIPUSH 0 // k = 0	0x10 0x00
	14	ISTORE k	0x36 0x03
	15 L2:		

(a) (b) (c)

Figure 4-14. (a) A Java fragment. (b) The corresponding Java assembly language. (c) The IJVM program in hexadecimal.

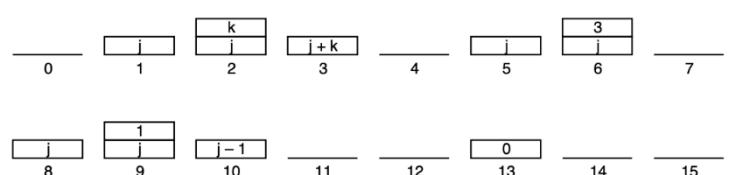


Figure 4-15. The stack after each instruction of Fig. 4-14(b).

Label	Operations	Comments	
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch	DEST = H
nop1	goto Main1	No nothing	DEST = SOURCE
iadd1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack	DEST = H
iadd2	H = TOS	H = top of stack	DEST = SOURCE
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack	DEST = H + SOURCE
isub1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack	DEST = H + SOURCE + 1
isub2	H = TOS	H = top of stack	DEST = H + 1
isub3	MDR = TOS = MDR - H; wr; goto Main1	Do subtraction; write to top of stack	DEST = SOURCE + 1
iand1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack	DEST = SOURCE - H
iand2	H = TOS	H = top of stack	DEST = SOURCE - 1
iand3	MDR = TOS = MDR AND H; wr; goto Main1	Do AND; write to new top of stack	DEST = -H
ior1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack	DEST = H AND SOURCE
ior2	H = TOS	H = top of stack	DEST = H OR SOURCE
ior3	MDR = TOS = MDR OR H; wr; goto Main1	Do OR; write to new top of stack	DEST = 0
dup1	MAR = SP = SP + 1	Increment SP and copy to MAR	DEST = 1
dup2	MDR = TOS; wr; goto Main1	Write new stack word	DEST = -1
pop1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack	
pop2		Wait for new TOS to be read from memory	
pop3	TOS = MDR; goto Main1	Copy new word to TOS	
swap1	MAR = SP - 1; rd	Set MAR to SP - 1; read 2nd word from stack	
swap2	MAR = SP	Set MAR to top word	
swap3	H = MDR; wr	Save TOS in H; write 2nd word to top of stack	
swap4	MDR = TOS	Copy old TOS to MDR	
swap5	MAR = SP - 1; wr	Set MAR to SP - 1; write as 2nd word on stack	
swap6	TOS = H; goto Main1	Update TOS	
bipush1	SP = MAR = SP + 1	MBR = the byte to push onto stack	
bipush2	PC = PC + 1; fetch	Increment PC, fetch next opcode	
bipush3	MDR = TOS = MBR; wr; goto Main1	Sign-extend constant and push on stack	
iload1	H = LV	MBR contains index; copy LV to H	
iload2	MAR = MBRU + H; rd	MAR = address of local variable to push	
iload3	MAR = SP + 1	SP points to new top of stack; prepare write	
iload4	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack	
iload5	TOS = MDR; goto Main1	Update TOS	
istore1	H = LV	MBR contains index; Copy LV to H	
istore2	MAR = MBRU + H	MAR = address of local variable to store into	
istore3	MDR = TOS; wr	Copy TOS to MDR; write word	
istore4	SP = MAR = SP - 1; rd	Read in next-to-top word on stack	
istore5	PC = PC + 1; fetch	Increment PC; fetch next opcode	
istore6	TOS = MDR; goto Main1	Update TOS	
wide1	PC = PC + 1; fetch; goto (MBR OR 0x100)	Multiway branch with high bit set	
wide_iload1	PC = PC + 1; fetch	MBR contains 1st index byte; fetch 2nd	
wide_iload2	H = MBRU << 8	H = 1st index byte shifted left 8 bits	
wide_iload3	H = MBRU OR H	H = 16-bit index of local variable	
wide_iload4	MAR = LV + H; rd; goto iload3	MAR = address of local variable to push	
wide_istore1	PC = PC + 1; fetch	MBR contains 1st index byte; fetch 2nd	
wide_istore2	H = MBRU << 8	H = 1st index byte shifted left 8 bits	
wide_istore3	H = MBRU OR H	H = 16-bit index of local variable	
wide_istore4	MAR = LV + H; goto istore3	MAR = address of local variable to store into	
ldc_w1	PC = PC + 1; fetch	MBR contains 1st index byte; fetch 2nd	
ldc_w2	H = MBRU << 8	H = 1st index byte << 8	
ldc_w3	H = MBRU OR H	H = 16-bit index into constant pool	
ldc_w4	MAR = H + CPP; rd; goto iload3	MAR = address of constant in pool	

Figure 4-17. The microgram for the Mic-1 (part 1 of 3).

Label	Operations	Comments
iinc1	H = LV	MBR contains index; Copy LV to H
iinc2	MAR = MBRU + H; rd	Copy LV + index to MAR; Read variable
iinc3	PC = PC + 1; fetch	Fetch constant
iinc4	H = MDR	Copy variable to H
iinc5	PC = PC + 1; fetch	Fetch next opcode
iinc6	MDR = MBR + H; wr; goto Main1	Put sum in MDR; update variable
goto1	OPC = PC - 1	Save address of opcode.
goto2	PC = PC + 1; fetch	MBR = 1st byte of offset; fetch 2nd byte
goto3	H = MBR << 8	Shift and save signed first byte in H
goto4	H = MBRU OR H	H = 16-bit branch offset
goto5	PC = OPC + H; fetch	Add offset to OPC
goto6	goto Main1	Wait for fetch of next opcode
ifl1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack
ifl2	OPC = TOS	Save TOS in OPC temporarily
ifl3	TOS = MDR	Put new top of stack in TOS
ifl4	N = OPC; if (N) goto T; else goto F	Branch on N bit
ifeq1	MAR = SP = SP - 1; rd	Read in next-to-top word of stack
ifeq2	OPC = TOS	Save TOS in OPC temporarily
ifeq3	TOS = MDR	Put new top of stack in TOS
ifeq4	Z = OPC; if (Z) goto T; else goto F	Branch on Z bit
if_icmpqe1	MAR = SP = SP - 1; rd	Read in next-to-top word of stack
if_icmpqe2	MAR = SP = SP - 1	Set MAR to read in new top-of-stack
if_icmpqe3	H = MDR; rd	Copy second stack word to H
if_icmpqe4	OPC = TOS	Save TOS in OPC temporarily
if_icmpqe5	TOS = MDR	Put new top of stack in TOS
if_icmpqe6	Z = OPC - H; if (Z) goto T; else goto F	If top 2 words are equal, goto T, else goto F
T	OPC = PC - 1; fetch; goto goto2	Same as goto1; needed for target address
F	PC = PC + 1	Skip first offset byte
F2	PC = PC + 1; fetch	PC now points to next opcode
F3	goto Main1	Wait for fetch of opcode

Figure 4-16. All permitted operations. Any of the above operations may be extended by adding “<< 8” to them to shift the result left by 1 byte. For example, a common operation is H = MBR << 8

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

Figure 4-17. The microprogram for the Mic-1 (part 2 of 3).

Label	Operations	Comments
ireturn1	MAR = SP = LV; rd	Reset SP, MAR to get link pointer
ireturn2		Wait for read
ireturn3	LV = MAR = MDR; rd	Set LV to link ptr; get old PC
ireturn4	MAR = LV + 1	Set MAR to read old LV
ireturn5	PC = MDR; rd; fetch	Restore PC; fetch next opcode
ireturn6	MAR = SP	Set MAR to write TOS
ireturn7	LV = MDR	Restore LV
ireturn8	MDR = TOS; wr; goto Main1	Save return value on original top of stack

Figure 4-17. The microprogram for the Mic-1 (part 3 of 3).

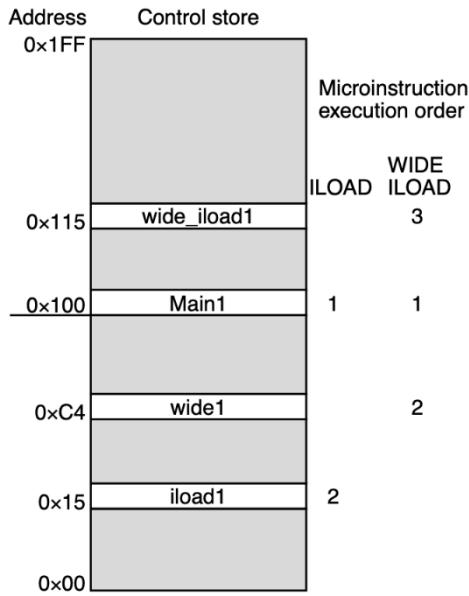


Figure 4-20. The initial microinstruction sequence for ILOAD and WIDE ILOAD. The addresses are examples.

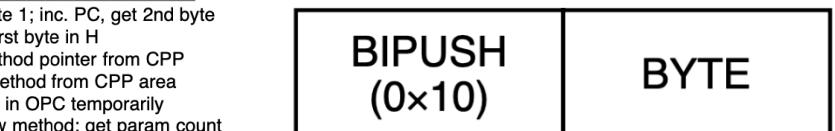


Figure 4-18. The BIPUSH instruction format.

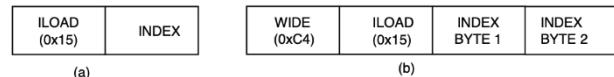


Figure 4-19. (a) ILOAD with a 1-byte index. (b) WIDE ILOAD with a 2-byte index.



Figure 4-21. The IINC instruction has two different operand fields.

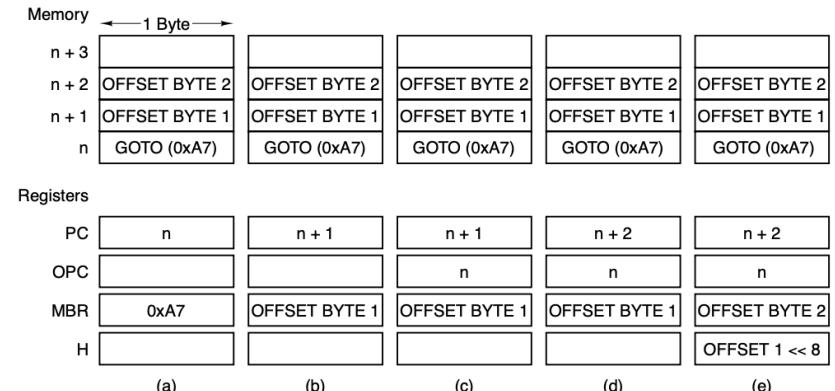


Figure 4-22. The situation at the start of various microinstructions. (a) Main1. (b) goto1. (c) goto2. (d) goto3. (e) goto4.

Label	Operations	Comments
pop1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack
pop2		Wait for new TOS to be read from memory
pop3	TOS = MDR; goto Main1	Copy new word to TOS
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch

Figure 4-23. New microprogram sequence for executing POP.

Label	Operations	Comments
pop1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack
Main1.pop	PC = PC + 1; fetch	MBR holds opcode; fetch next byte
pop3	TOS = MDR; goto (MBR)	Copy new word to TOS; dispatch on opcode

Figure 4-24. Enhanced microprogram sequence for executing POP.

Label	Operations	Comments
iload1	H = LV	MBR contains index; Copy LV to H
iload2	MAR = MBRU + H; rd	MAR = address of local variable to push
iload3	MAR = SP = SP + 1	SP points to new top of stack; prepare write
iload4	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack
iload5	TOS = MDR; goto Main1	Update TOS
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch

Figure 4-25. Mic-1 code for executing ILOAD.

Label	Operations	Comments
iload1	MAR = MBRU + LV; rd	MAR = address of local variable to push
iload2	MAR = SP = SP + 1	SP points to new top of stack; prepare write
iload3	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack
iload4	TOS = MDR	Update TOS
iload5	PC = PC + 1; fetch; goto (MBR)	MBR already holds opcode; fetch index byte

Figure 4-26. Three-bus code for executing ILOAD.

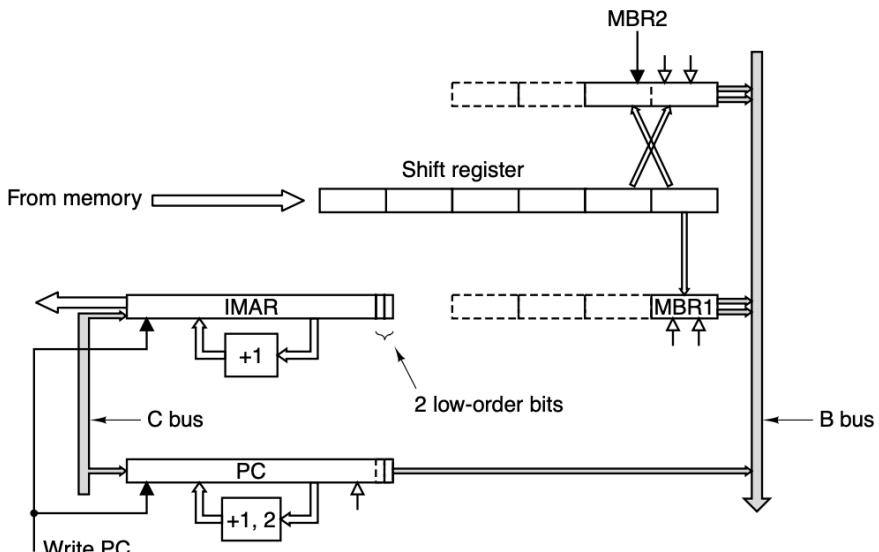
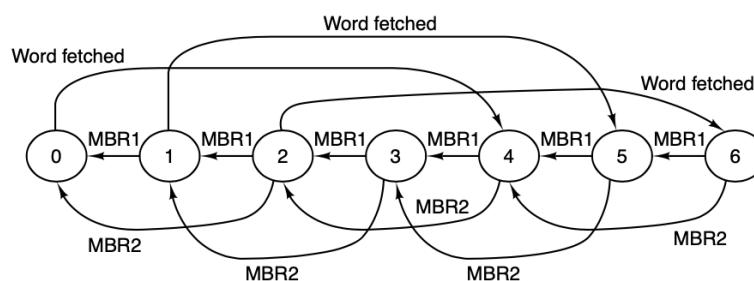


Figure 4-27. A fetch unit for the Mic-1.



Transitions

MBR1: Occurs when MBR1 is read

MBR2: Occurs when MBR2 is read

Word fetched: Occurs when a memory word is read and 4 bytes are put into the shift register

Figure 4-28. A finite state machine for implementing the IFU.

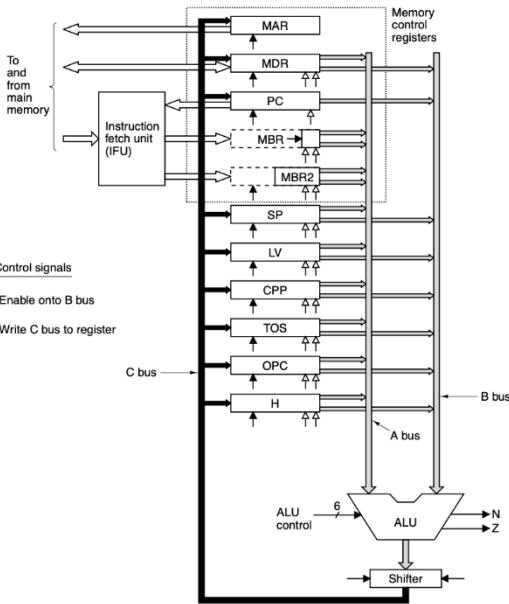


Figure 4-29. The datapath for Mic-2.

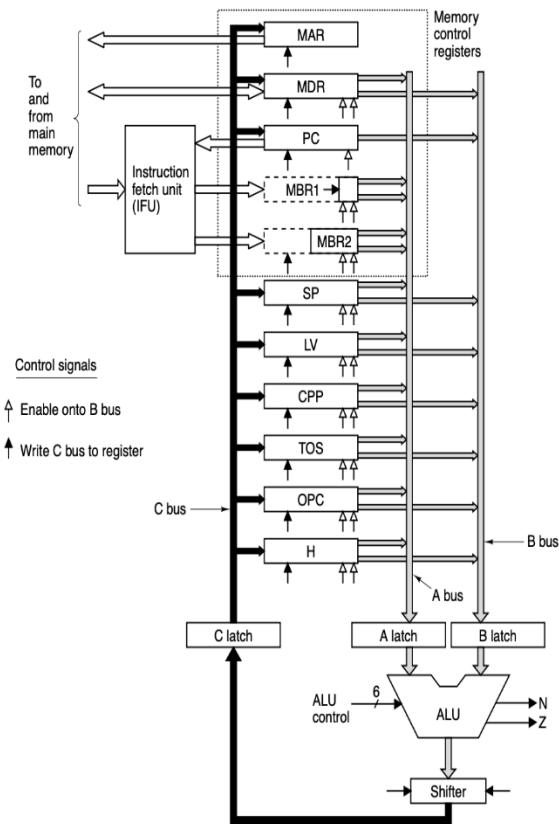


Figure 4-31. The three-bus data path used in the Mic-3.

Label	Operations	Comments
nop1	goto (MBR)	Branch to next instruction
iadd1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR+H; wr; goto (MBR1)	Add top two words; write to new top of stack
isub1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack
isub2	H = TOS	H = top of stack
isub3	MDR = TOS = MDR-H; wr; goto (MBR1)	Subtract TOS from Fetched TOS-1
land1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack
land2	H = TOS	H = top of stack
land3	MDR = TOS = MDR AND H; wr; goto (MBR1)	AND Fetched TOS-1 with TOS
ior1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack
ior2	H = TOS	H = top of stack
ior3	MDR = TOS = MDR OR H; wr; goto (MBR1)	OR Fetched TOS-1 with TOS
dup1	MAR = SP = SP + 1	Increment SP; copy to MAR
dup2	MDR = TOS; wr; goto (MBR1)	Write new stack word
pop1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack
pop2		Wait for read
pop3	TOS = MDR; goto (MBR1)	Copy new word to TOS
swap1	MAR = SP - 1; rd	Read 2nd word from stack; set MAR to SP
swap2	MAR = SP	Prepare to write new 2nd word
swap3	H = MDR; wr	Save new TOS; write 2nd word to stack
swap4	MDR = TOS	Copy old TOS to MDR
swap5	MAR = SP - 1; wr	Write old TOS to 2nd place on stack
swap6	TOS = H; goto (MBR1)	Update TOS
bipush1	SP = MAR = SP + 1	Set up MAR for writing to new top of stack
bipush2	MDR = TOS = MBR1; wr; goto (MBR1)	Update stack in TOS and memory
iload1	MAR = LV + MBR1U; rd	Move LV + index to MAR; read operand
iload2	MAR = SP = SP + 1	Increment SP; Move new SP to MAR
iload3	TOS = MDR; wr; goto (MBR1)	Update stack in TOS and memory
istore1	MAR = LV + MBR1U	Set MAR to LV + index
istore2	MDR = TOS; wr	Copy TOS for storing
istore3	MAR = SP = SP - 1; rd	Decrement SP; read new TOS
istore4		Wait for read
istore5	TOS = MDR; goto (MBR1)	Update TOS
wide1	goto (MBR1 OR 0x100)	Next address is 0x100 Ored with opcode
wide_iload1	MAR = LV + MBR2U; rd; goto iload2	Identical to iload1 but using 2-byte index
wide_istore1	MAR = LV + MBR2U; goto istore2	Identical to istore1 but using 2-byte index
ldc_w1	MAR = CPP + MBR2U; rd; goto iload2	Same as wide_iload1 but indexing off CPP
iinc1	MAR = LV + MBR1U; rd	Set MAR to LV + index for read
iinc2	H = MBR1	Set H to constant
iinc3	MDR = MDR + H; wr; goto (MBR1)	Increment by constant and update
goto1	H = PC - 1	Copy PC to H
goto2	PC = H + MBR2	Add offset and update PC
goto3		Have to wait for IFU to fetch new opcode
goto4	goto (MBR1)	Dispatch to next instruction
iflt1	MAR = SP = SP - 1; rd	Read in next-to-top word on stack
iflt2	OPC = TOS	Save TOS in OPC temporarily
iflt3	TOS = MDR	Put new top of stack in TOS
iflt4	N = OPC; if (N) goto T; else goto F	Branch on N bit

Figure 4-30. The microprogram for the Mic-2 (part 1 of 2).

Label	Operations	Comments
ifeq1	MAR = SP = SP - 1; rd	Read in next-to-top word of stack
ifeq2	OPC = TOS	Save TOS in OPC temporarily
ifeq3	TOS = MDR	Put new top of stack in TOS
ifeq4	Z = OPC; if (Z) goto T; else goto F	Branch on Z bit
if_icmpqe1	MAR = SP = SP - 1; rd	Read in next-to-top word of stack
if_icmpqe2	MAR = SP = SP - 1	Set MAR to read in new top-of-stack
if_icmpqe3	H = MDR; rd	Copy second stack word to H
if_icmpqe4	OPC = TOS	Save TOS in OPC temporarily
if_icmpqe5	TOS = MDR	Put new top of stack in TOS
if_icmpqe6	Z = H - OPC; if (Z) goto T; else goto F	If top 2 words are equal, goto T, else goto F
T	H = PC - 1; goto goto2	Same as goto1
F	H = MBR2	Touch bytes in MBR2 to discard
F2	goto (MBR1)	
invokevirtual1	MAR = CPP + MBR2U; rd	Put address of method pointer in MAR
invokevirtual2	OPC = PC	Save Return PC in OPC
invokevirtual3	PC = MDR	Set PC to 1st byte of method code.
invokevirtual4	TOS = SP - MBR2U	TOS = address of OBJREF - 1
invokevirtual5	TOS = MAR = H = TOS + 1	TOS = address of OBJREF
invokevirtual6	MDR = SP + MBR2U + 1; wr	Overwrite OBJREF with link pointer
invokevirtual7	MAR = SP = MDR	Set SP, MAR to location to hold old PC
invokevirtual8	MDR = OPC; wr	Prepare to save old PC
invokevirtual9	MAR = SP = SP + 1	Inc. SP to point to location to hold old LV
invokevirtual10	MDR = LV; wr	Save old LV
invokevirtual11	LV = TOS; goto (MBR1)	Set LV to point to zeroth parameter.
ireturn1	MAR = SP = LV; rd	Reset SP, MAR to read Link ptr
ireturn2		Wait for link ptr
ireturn3	LV = MAR = MDR; rd	Set LV, MAR to link ptr; read old PC
ireturn4	MAR = LV + 1	Set MAR to point to old LV; read old LV
ireturn5	PC = MDR; rd	Restore PC
ireturn6	MAR = SP	
ireturn7	LV = MDR	Restore LV
ireturn8	MDR = TOS; wr; goto (MBR1)	Save return value on original top of stack

Figure 4-30. The microprogram for the Mic-2 (part 2 of 2).

Label	Operations	Comments
swap1	MAR = SP - 1; rd	Read 2nd word from stack; set MAR to SP
swap2	MAR = SP	Prepare to write new 2nd word
swap3	H = MDR; wr	Save new TOS; write 2nd word to stack
swap4	MDR = TOS	Copy old TOS to MDR
swap5	MAR = SP - 1; wr	Write old TOS to 2nd place on stack
swap6	TOS = H; goto (MBR1)	Update TOS

Figure 4-32. The Mic-2 code for SWAP.

	Swap1	Swap2	Swap3	Swap4	Swap5	Swap6
Cy	MAR=SP-1;rd	MAR=SP	H=MDR;wr	MDR=TOS	MAR=SP-1;wr	TOS=H;goto (MBR1)
1	B=SP					
2	C=B-1	B=SP				
3	MAR=C; rd	C=B				
4	MDR=mem	MAR=C				
5		B=MDR				
6		C=B	B=TOS			
7		H=C; wr	C=B	B=SP		
8		Mem=MDR	MDR=C	C=B-1	B=H	
9				MAR=C; wr	C=B	
10				Mem=MDR	TOS=C	
11						goto (MBR1)

Figure 4-33. The implementation of SWAP on the Mic-3.

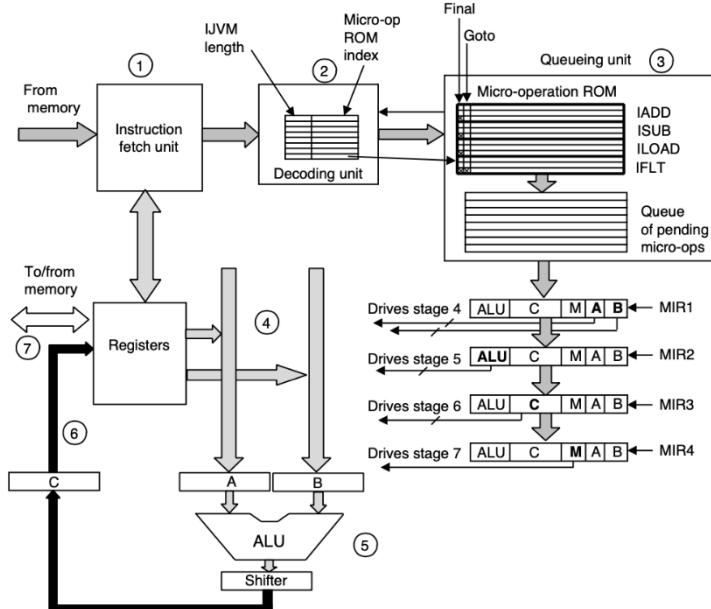


Figure 4-35. The main components of the Mic-4.

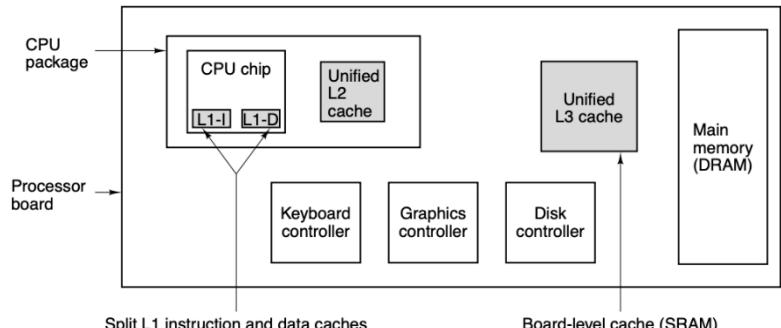


Figure 4-37. A system with three levels of cache.

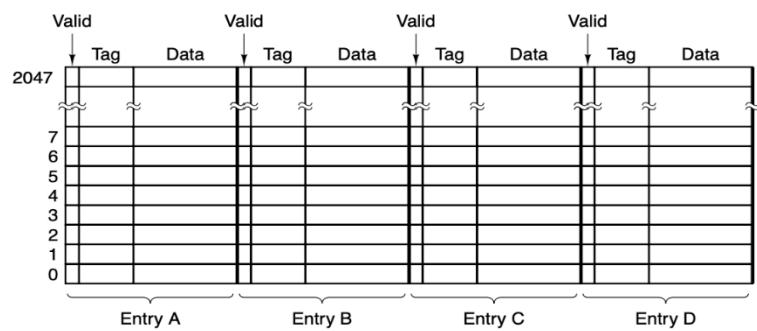


Figure 4-39. A four-way associative cache.

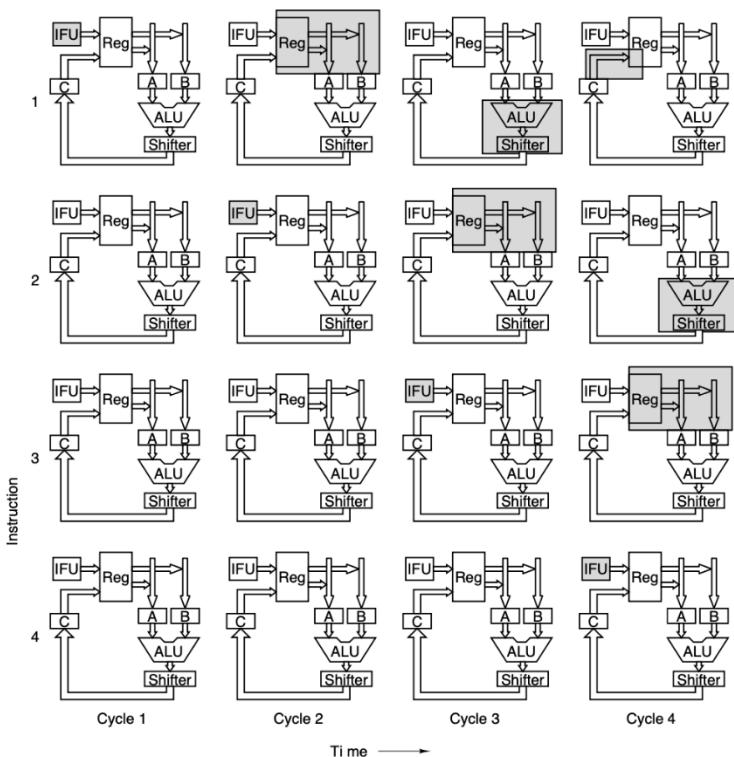


Figure 4-34. Graphical illustration of how a pipeline works.

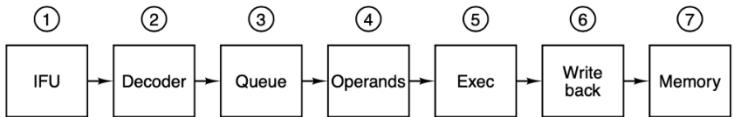
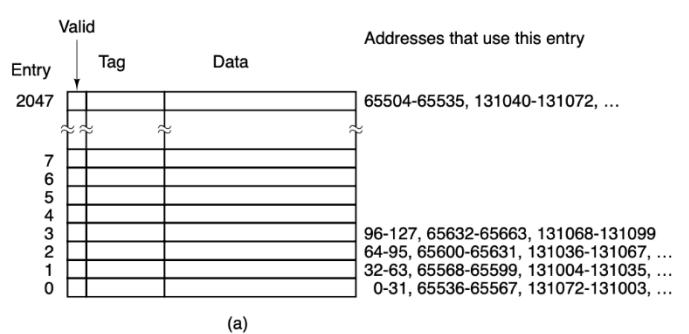
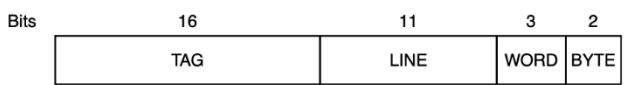


Figure 4-36. The Mic-4 pipeline.



(a)



(b)

Figure 4-38. (a) A direct-mapped cache. (b) A 32-bit virtual address.

```

if (i == 0)
    k = 1;
else
    k = 2;
    BR Next; unconditional branch to Next
    Else: MOV k,2; move 2 to k
    Next:

```

(a)

```

CMP i,0; compare i to 0
BNE Else; branch to Else if not equal
Then: MOV k,1; move 1 to k
BR Next; unconditional branch to Next
Else: MOV k,2; move 2 to k
Next:

```

(b)

Figure 4-40. (a) A program fragment. (b) Its translation to a generic assembly language.

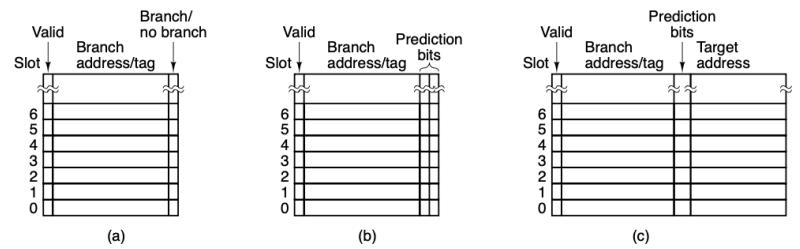


Figure 4-41. (a) A 1-bit branch history. (b) A 2-bit branch history. (c) A mapping between branch instruction address and target address.

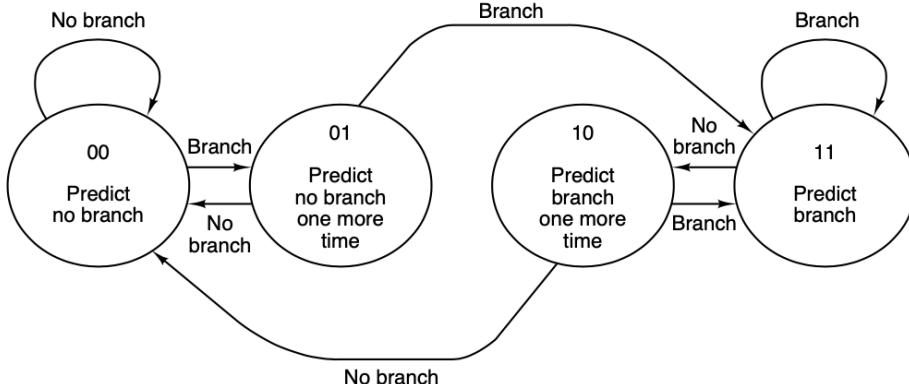


Figure 4-42. A 2-bit finite-state machine for branch prediction.

Cy	#	Decoded			Registers being read							Registers being written									
			Iss	Ret	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
1	1	R3=R0+R1	1		1	1								1							
	2	R4=R0+R2	2		2	1	1							1	1						
2	3	R5=R0+R1	3		3	2	1							1	1	1					
	4	R6=R1+R4	—		3	2	1							1	1	1					
3					3	2	1							1	1	1					
4					1	2	1	1							1	1					
					2	1	1								1	1					
					3										1	1					
5					4										1						
	5	R7=R1+R2	5			1		1							1						
6	6	R1=R0-R2	—			2	1	1							1	1					
	7					2	1	1							1	1					
8					4		1	1								1					
9					5											1					
	7	R3=R3*R1	6			1	1	1	1						1	1	1				
10						1	1	1	1						1	1					
11					6		1	1							1						
12					7																
13	8	R1=R4+R4	8											2		1					
14														2		1					
15														8							

Figure 4-43. Operation of a superscalar CPU with in-order issue and in-order completion.

Cy	#	Decoded			Registers being read							Registers being written									
			Iss	Ret	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
1	1	R3=R0+R1	1		1	1								1							
	2	R4=R0+R2	2		2	1	1							2	1	1					
2	3	R5=R0+R1	3		3	2	1							3	2	1					
	4	R6=R1+R4	—		3	2	1							3	2	1					
3					3	2	1							3	3	2					
	5	R7=R1+R2	4			1		1						4	3	3					
4	6	R1=R0-R2	—			2	1	1						2	3	2	3				
	7	R3=R3+S1	4			3	4	2						3	4	2	3				
5						4	3	2						1	2	3	2	3			
	8	S2=R4+R4	8			3	1	2	2					1	1	1	1	1	1	1	1
6						6	2	1	3					1							
	7					7	2	1	1	3				1	1	1	1	1	1	1	1
8						4	1	1	1	2				1	1	1	1	1	1	1	1
	8					5	1	1	2					1	1	1	1	1	1	1	1
9						8	7														

Figure 4-44. Operation of a superscalar CPU with out-of-order issue and out-of-order completion.

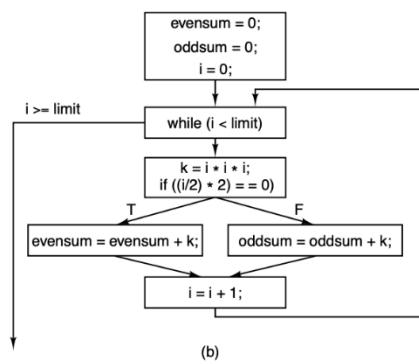


Figure 4-45. (a) A program fragment. (b) The corresponding basic block graph.