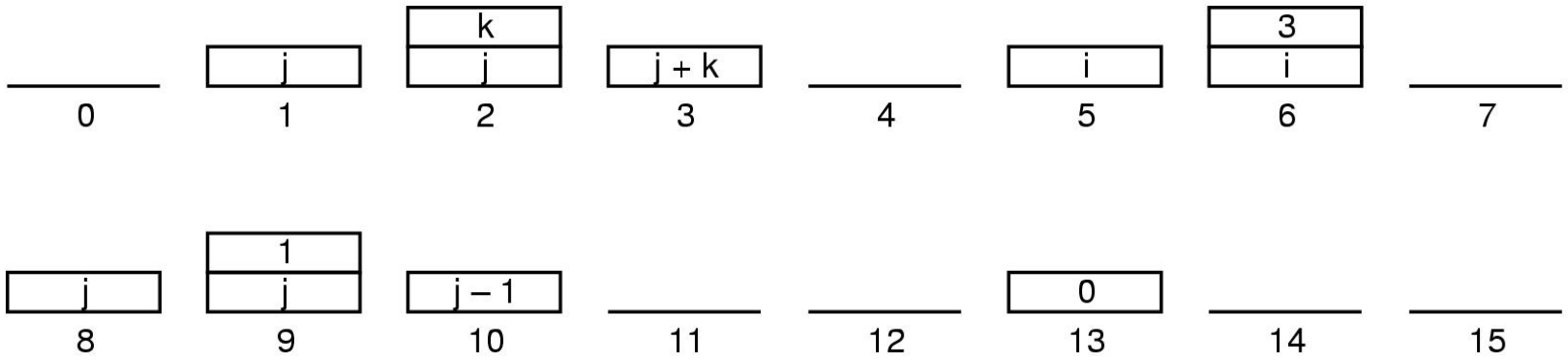
4.2.4 Преведување на JAVA програми за IJVM

- Java програма → IJVM асемблерски код → бинарна програма

i = j + k;	1	ILOAD j	**00010101** 00000010
if (i == 3)	2	ILOAD k	**00010101** 00000011
k=0;	3	IADD	**01100000**
else	4	ISTORE i	**00110110** 00000001
j = j – 1;	5	ILOAD i	**00010101** 00000001
	6	BIPUSH 3	**00010000** 00000011
	7	IF_ICMPEQ L1	**10011111** 00000000 00001101
	8	ILOAD j	**00010101** 00000010
	9	BIPUSH 1	**00010000** 00000001
	10	ISUB	**01100100**
	11	ISTORE j	**00110110** 00000010
	12	GOTO L2	**10100111** 00000000 00000111
	13 L1:	BIPUSH 0	**00010000** 00000000
	14	ISTORE k	**00110110** 00000011
	15 L2:		

4.2.4 Преведување на JAVA програми за IJVM



4.3.1 Комплетна микропрограма на Mic-1 микроархитектурата

Label	Operations	Comments
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch
nop1	goto Main1	Do nothing
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 Main1	Add top two words; write to 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 Main1	Do subtraction; write to top of stack
iand1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iand2	H = TOS	H = top of stack
iand3	MDR = TOS = MDR AND H; wr; goto Main1	Do AND; write to new top of stack
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 Main1	Do OR; write to new top of stack
dup1	MAR = SP = SP + 1	Increment SP and copy to MAR
dup2	MDR = TOS; wr; goto Main1	Write new stack word
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 = 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

4.3.1 Комплетна микропрограма на Mic-1 микроархитектурата

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
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
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_icmpeq1	MAR = SP = SP – 1; rd	Read in next-to-top word 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 (Z) goto T; else goto F	If top 2 words are equal, goto T, else goto
T	OPC = PC – 1; fetch; goto goto2	Same as goto1; needed for target addres
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
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 cour
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

4.3.1 Комплетна микропрограма на Mic-1 микроахитектурата

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

4.4 Дизајн на микроархитектурата

- Едноставните машини не се брзи, а брзите машини не се едноставни
  - Mic-1 микроархитектурата се состои од минимално количество хардвер: 10 регистри, едноставна ALU, поместувач, декодер, контролна меморија, итн. (~5000 транзистори + се` што е потребно за контролната (ROM) и главната (RAM) меморија)

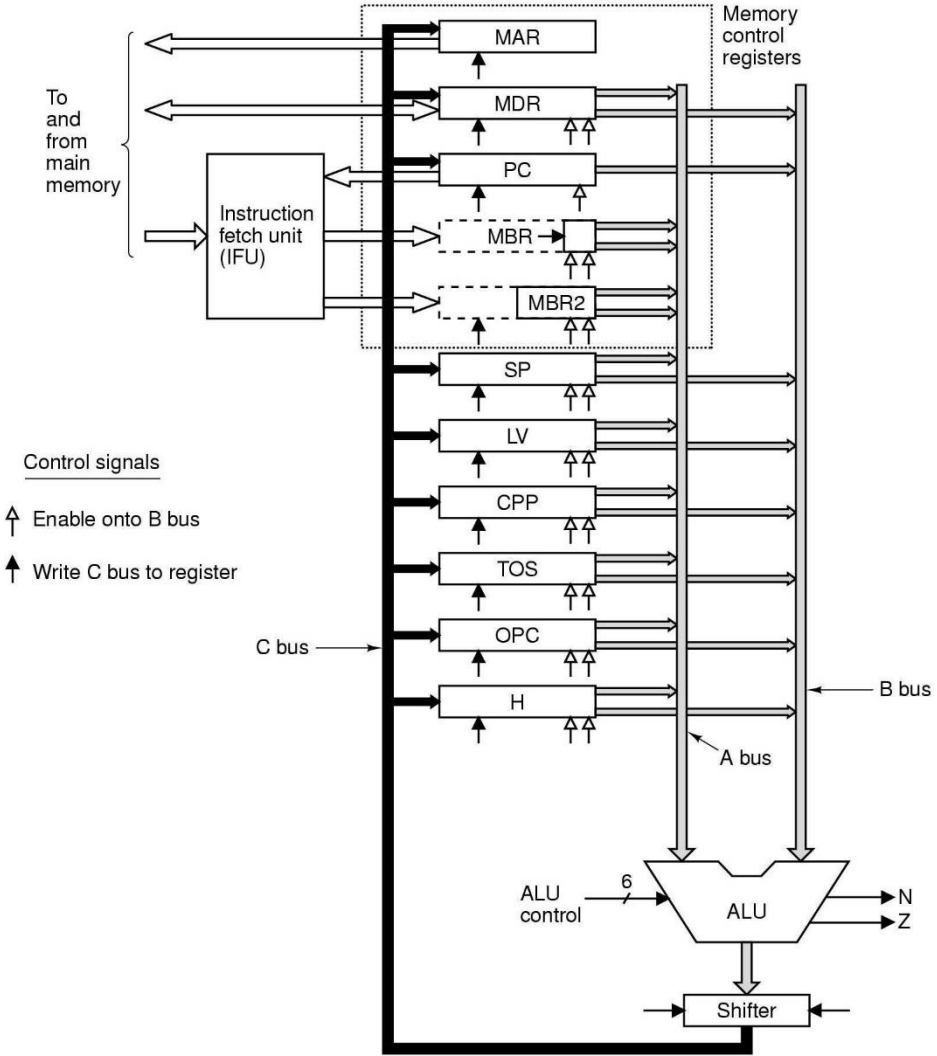
4.4.1 Намалување на должината на патеката на извршување

- Микроархитектура со три магистрали (A, B и C)
  - Две магистрали (A и B) на влезот на ALU, при што сите (или повеќето) регистри ќе имаат пристап до двете магистрали
    - Наместо да се троши еден циклус за копирање на содржината на еден од регистрите во регистерот H, се овозможува собирање на неговата содржина со содржината на друг регистер во еден циклус

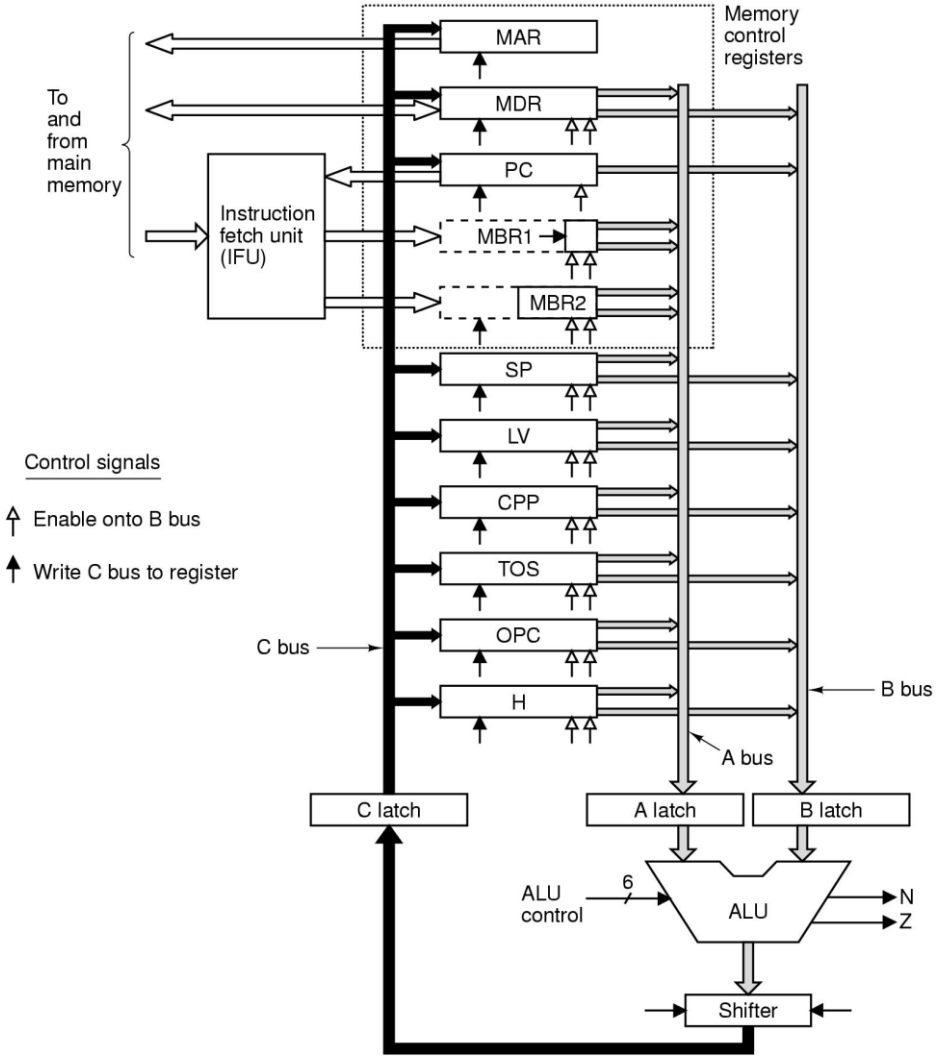
4.4.1 Намалување на должината на патеката на извршување

- Посебна единица за преземање на инструкции (Instruction Fetch Unit – IFU)
  - Независно го инкрементира програмскиот бројач (PC) и презема бајтови од меморијата пред тие навистина да бидат побарани
  - Наместо еден 8-битен MBR, постојат два:
    - 8-битен MBR1 (го содржи следниот бајт, како кај Mic-1)
    - 16-битен MBR2 (ги содржи следните два бајти, бидејќи некои инструкции имаат 16-битни операнди)
  - IFU презема цели зборови од меморијата и ги става во **ред на чекање (shift register)**, од каде излегуваат по еден или по два бајти одеднаш (преку MBR1 или MBR2)

4.4.2 Дизајн со претходно преземање (prefetching): Mic-2



4.4.3 Дизајн со протечна обработка (pipelining): Mic-3



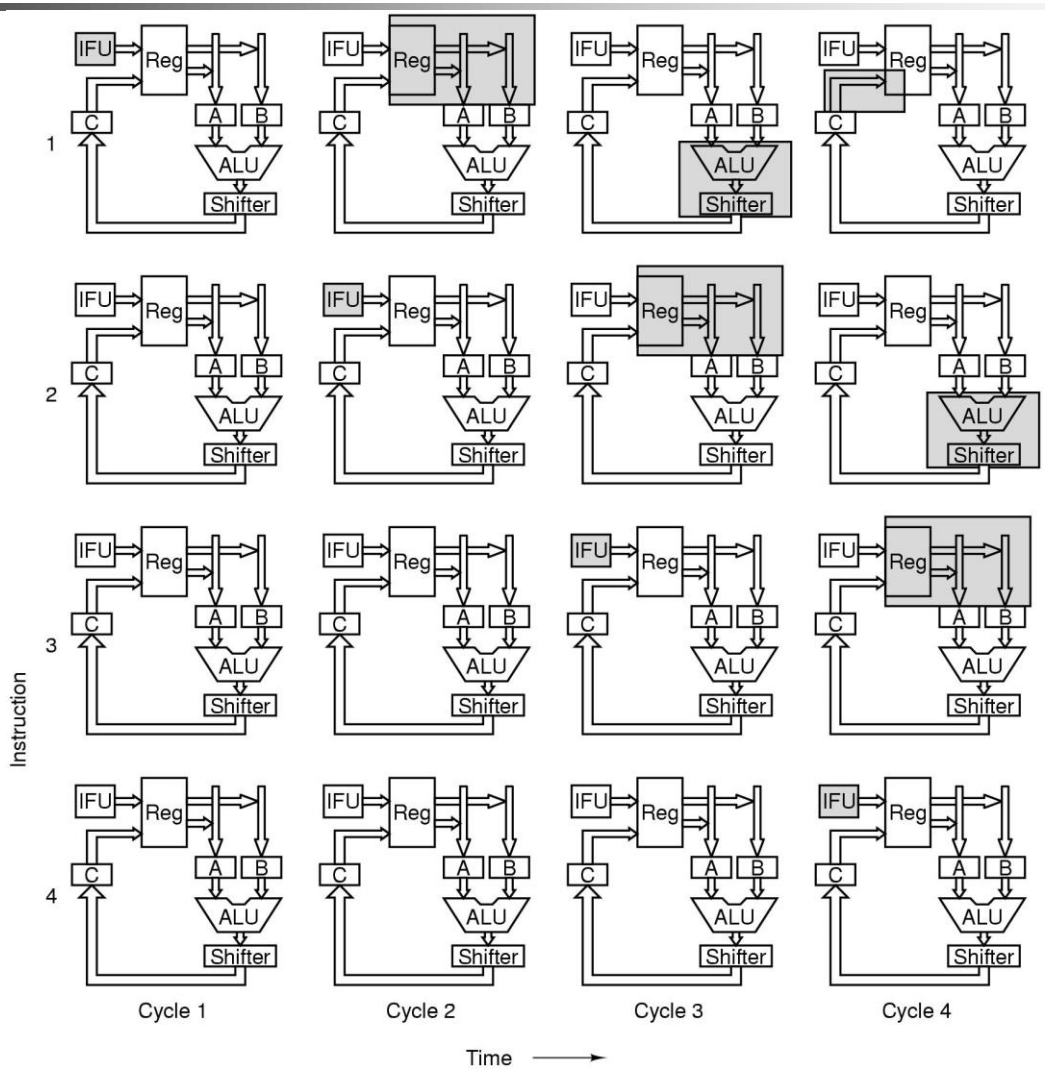
4.4.3 Дизајн со протечна обработка (pipelining): Mic-3

- Во циклусот на податочната патека, во основа, доминираат:
  - Времето потребно за читање на содржината на соодветните регистри преку магистралите A и B
  - Времето потребно за ALU и поместувачот да ја завршат својата работа
  - Времето потребно за запишување на резултатите во соодветните регистри преку магистралата C
- Со внесување на три дополнителни регистри (latches), по еден вдолж секоја од магистралите, податочната патека се **партиционира** на три одделни целини кои можат да функционираат независно една од друга
- Протечната обработка е клучна техника која се применува кај сите современи процесори
- Придобивки:
  - Намалување на времетраењето на еден циклус (забрзување на системскиот часовник; ~3 пати)
    - Забелешка: потребни се 3 циклуси (микро-чекори) за изминување на податочната патека
  - Можност за истовремена работа на сите сегменти од податочната патека

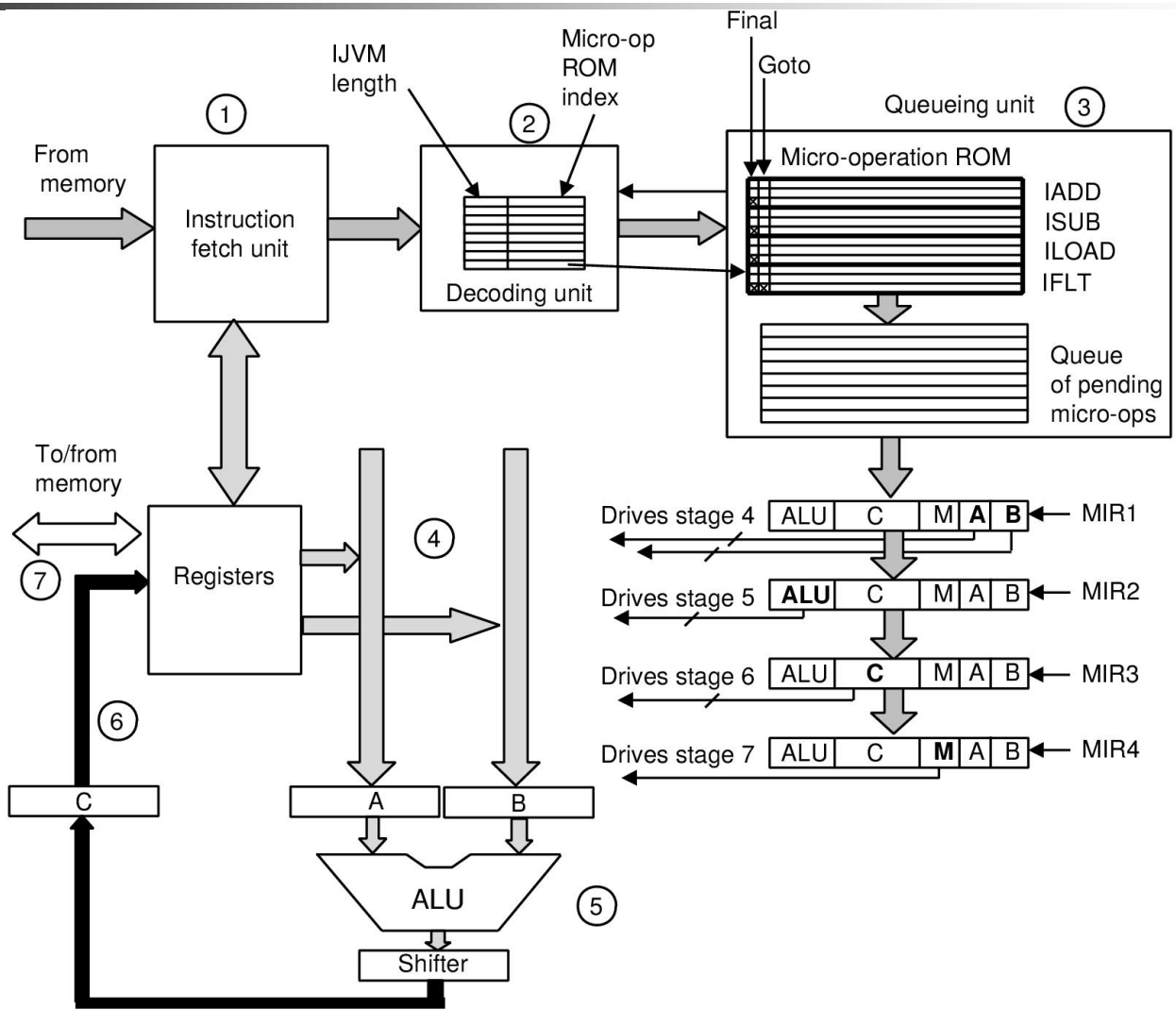
4.4.3 Дизајн со протечна обработка (pipelining): Mic-3

- **Циклус 1**
  - Единицата за преземање на инструкции ја презема Инструкција_1
- **Циклус 2**
  - Се читаат содржините на регистрите потребни на Инструкција_1 и се запишуваат во Latch A и Latch B
  - Единицата за преземање на инструкции ја презема Инструкција_2
- **Циклус 3**
  - ALU и поместувачот ја извршуваат соодветната операција опишана со Инструкција_1 и го запишуваат резултатот во Latch C
  - Се читаат содржините на регистрите потребни на Инструкција_2 и се запишуваат во Latch A и Latch B
  - Единицата за преземање на инструкции ја презема Инструкција_3
- **Циклус 4**
  - Резултатот од Инструкција_1 се запишува во еден или повеќе регистри
  - ALU и поместувачот ја извршуваат соодветната операција опишана со Инструкција_2 и го запишуваат резултатот во Latch C
  - Се читаат содржините на регистрите потребни на Инструкција_3 и се запишуваат во Latch A и Latch B
  - Единицата за преземање на инструкции ја презема Инструкција_4
- Следните циклуси наликуваат на Циклус 4

4.4.3 Дизајн со протечна обработка (pipelining): Mic-3



4.4.4 Дизајн со протечна обработка во седум фази: Mic-4



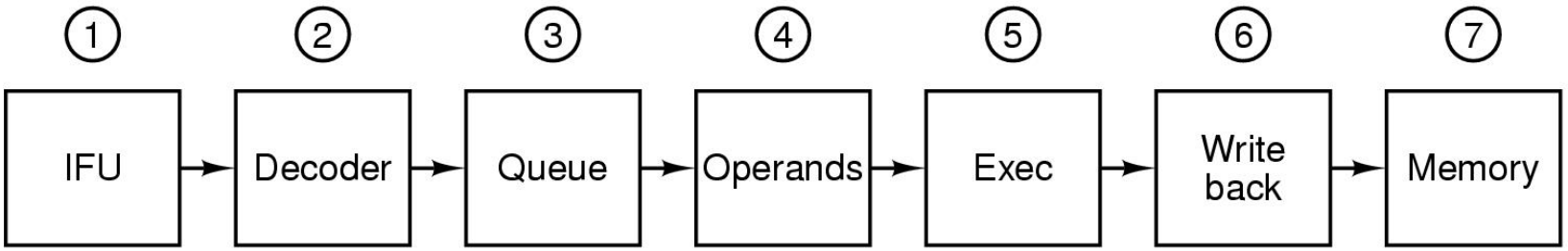
4.4.4 Дизајн со протечна обработка во седум фази: Mic-4

- Единицата за преземање на инструкции (IFU) ја проследува низата од бајтови кон **единица за декодирање (decoding unit)**
- Единицата за декодирање има внатрешна меморија – секоја редица се состои од два дела:
  - должина на тековната IJVM инструкција (во бајти) и
  - покажувач кон друга меморија содржана во соседната единица – **единица со ред на чекање (queueing unit)**
- Единицата со ред на чекање содржи:
  - Меморија на микро-операции (микропрограма – извршувањето на секоја IJVM инструкција е опишано во неколку последователни редици)
    - Микро-операциите имаат сличен формат како и микроинструкциите од 4.1.2:
      - Полињата NEXT_ADDRESS и JAM се изоставени
      - Додадено е ново кодирано поле за избор на влезен регистер за магистралата A
      - Додаден е бит **Final**, кој има вредност 1 само кај последната микро-операција за тековната IJVM инструкција
      - Додаден е бит **Goto**, кој има вредност 1 кај микро-операциите кои претставуваат условни разгранувања

4.4.4 Дизајн со протечна обработка во седум фази: Mic-4

- Ред на чекање
  - Се полни со микро-операции кои треба непосредно да се извршат, се додека не биде забележана микро-операција чиј Final бит има вредност 1
  - Под претпоставка дека не станува збор за инструкција чиј Goto бит има вредност 1, на единицата за декодирање и се испраќа **потврда (acknowledgement)** за продолжување со следната IJVM инструкција
  - Микро-операциите се проследуваат кон првиот од **четирите микро-инструкциски регистри (MIR1-MIR4)**
    - во првиот се активни полињата A и B (избор на влезни регистри)
    - во вториот е активно полето ALU (аритметичко-логичка операција и поместување)
    - во третиот е активно полето C (избор на излезни регистри)
    - во четвртиот е активно полето Mem (мемориска операција)
- Забелешка: штом се појави микро-операција која претставува условно разгранување (Goto=1), на единицата за декодирање не и се испраќа потврда, што доведува до застој на машината се додека не се разреши разгранувањето

4.4.4 Дизајн со протечна обработка во седум фази: Mic-4



- Резултат:
  - Процесор со висок степен на протечност (deeply pipelined CPU)
  - Индивидуалните чекори се кратки
  - Фреквенцијата на системскиот часовник е висока
- Забелешка: Pentium II имплементацијата е концептуално слична на Mic-4