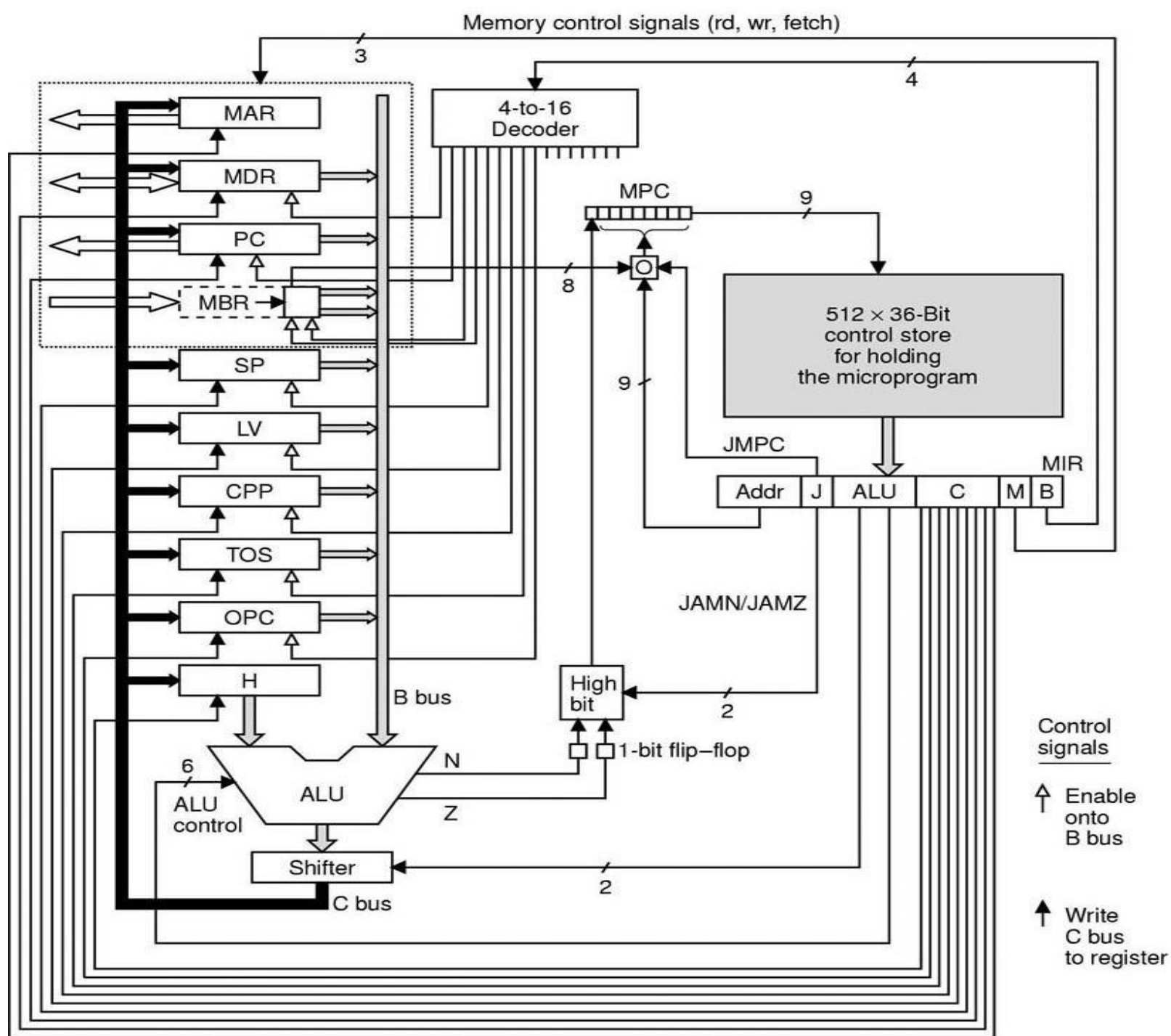


4.1.3 Микроинструкциска контрола: Mic-1

- Како се определува кои од контролните сигнали треба да бидат овозможени на почетокот на секој циклус?
- **Секвенцер (sequencer)** – одговорен за изминување низ секвенцата од операции кои се неопходни за извршување на една единствена инструкција
- Во секој циклус, секвенцерот треба да даде два вида информации:
 - Состојба на секој контролен сигнал во системот
 - Адреса на следната микроинструкција што треба да биде извршена



4.1.3 Микроинструкциска контрола: Mic-1

- Блок-дијаграмот го сочинуваат
 - Податочна патека (лево)
 - Контролна секција (десно)
- **Контролна меморија (control store)** – меморија во која се чува комплетната микропрограма (може да биде имплементирана и како множество логички порти)
 - Капацитет: 512 (2^9) 36-битни микроинструкции
- **MPC (MicroProgram Counter)** – микропрограамски бројач (адресен регистер за контролната меморија)
- **MIR (MicroInstruction Register)** – микроинструкциски регистер (за сместување на тековната микроинструкција)

4.1.3 Микроинструкциска контрола: Mic-1

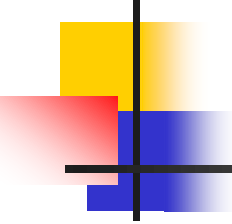
- Што се случува за време од еден циклус?
 - На почетокот на циклусот, во MIR се вчитува оној збор од контролната меморија кон кој покажува MPC (Δw)
 - Содржината на еден од регистрите се проследува на магистралата B, а ALU веќе знае која операција треба да ја изврши (Δx)
 - По време Δu , излезот од поместувачот, ALU, како и сигналите N и Z, се стабилни
 - Вредностите на N и Z се запишуваат во 1-битни флип-флопови (1-битни регистри) на крајот од циклусот (на растечкиот раб од импулсот), како и кај сите други регистри
 - По истекот на уште еден интервал, Δz , излезот од поместувачот доаѓа до регистрите преку магистралата C – тогаш се запишува содржината во регистрите и во N и Z флип-флоповите

4.1.3 Микроинструкциска контрола: Mic-1

- Покрај контролата на податочната патека, микропрограмата мора да определи **која** микроинструкција ќе се извршува во следниот циклус – тоа се прави некаде на средината на импулсот
- Како се пресметува адресата на следната микроинструкција?

4.1.3 Микроинструкциска контрола: Mic-1

- 9-битното поле NEXT_ADDRESS се копира во MPC
- Истовремено, се проверува полето JAM
 - Ако вредноста е 000, не е потребно да се прави ништо повеќе – MPC веќе покажува кон следната микроинструкција
 - Ако барем еден од битовите JAMZ и JAMN е 1, се пресметува т.н. "High bit" (најлевиот бит) според следната Булова (логичка) функција:
$$F = (JAMZ \text{ AND } Z) \text{ OR } (JAMN \text{ AND } N) \text{ OR } NEXT_ADDRESS[8]$$
 - Ако битот JMPC е 1, се пресметува бит-по-бит логичко ИЛИ на 8-те битови од регистерот MBR и 8-те најмалку значајни (најдесни) битови од полето NEXT_ADDRESS
 - Забелешка: ако JMPC е 1, 8-те најдесни битови од полето NEXT_ADDRESS се (обично) нули, така што адресата на следната микроинструкција зависи **само** од содржината на MBR



4.2 Пример за архитектура на инструкциското множество: JVM

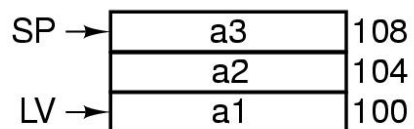
- Некои карактеристики на архитектурата на инструкциското множество (**макроархитектура**)
 - Stack (магацин)
 - JVM мемориски модел
 - JVM инструкциско множество
 - Преведување на Java програми за JVM



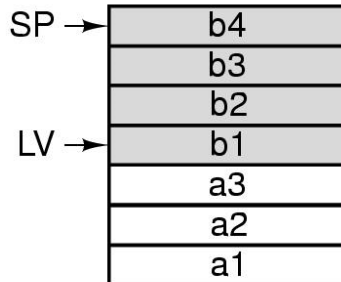
4.2.1 Stack (магацин)

- Дел од меморијата наменет за чување на вредности на **локални променливи**, како и за привремено чување на **операнди** при пресметување на аритметички изрази (operand stack, кај некои машини...)
- Посебен регистер (во случајов наречен **LV – Local Variable Pointer**) покажува на дното од stack-от за тековната процедура (функција, метод)
- Друг регистер (во случајов наречен **SP – Stack Pointer**) покажува на врвот од stack-от за тековната процедура
- **Рамка на локални променливи (local variable frame)** – податочна структура ограничена со LV и SP
- Одделните променливи немаат апсолутни адреси – до нив може да се пристапи ако се знае релативното поместување (offset) во однос на покажувачот LV

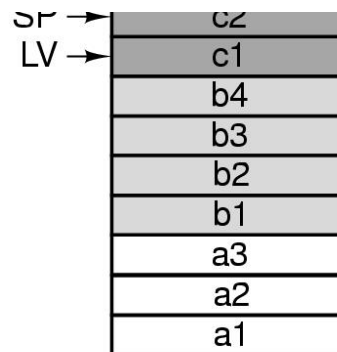
4.2.1 Stack (магацин)



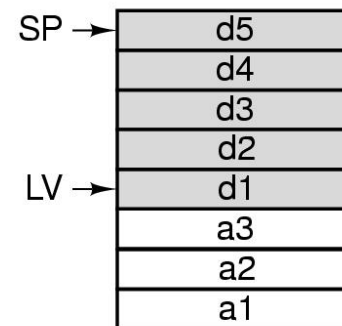
а) Активна е процедурата А



б) А ја повикала В



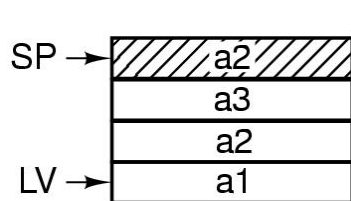
в) В ја повикала С



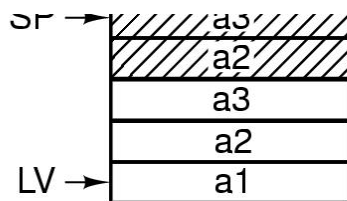
г) А ја повикала D,
откако С и В
завршиле со извршувањето

4.2.1 Stack (магацин)

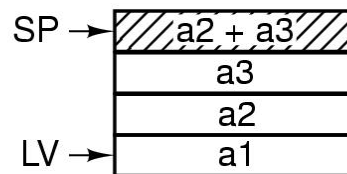
- Пресметување на аритметички израз
- Пример: $a1 = a2 + a3$
 - а) Стави ја (**push**) вредноста на $a2$ на врвот од stack-от и инкрементирај го SP за онолку колку што изнесува бројот на бајти содржани во еден збор (во случајов, за 4) – **ILOAD**
 - б) Стави ја вредноста на $a3$ на врвот од stack-от – **ILOAD**
 - в) Извади (**pop**) две вредности од врвот на stack-от, изврши го собирањето и стави ја вредноста на резултатот на врвот од stack-от – **IADD**
 - г) Извади вредност од врвот на stack-от и запиши ја во локалната променлива $a1$ – **ISTORE**



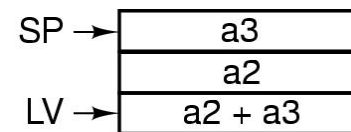
а)



б)



в)



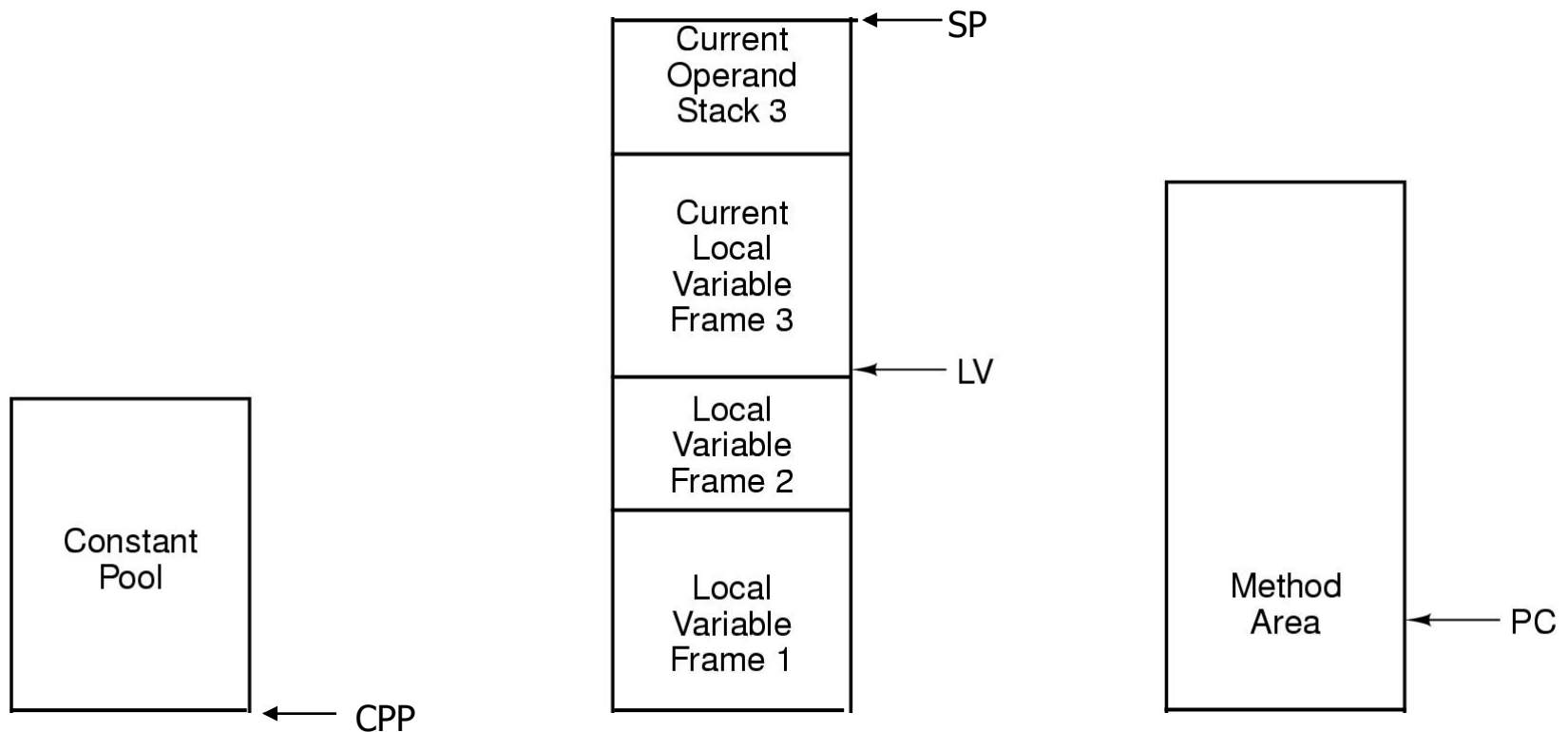
г)



4.2.2 JVM мемориски модел

- Четири мемориски подрачја
 - **Подрачје на константи (Constant Pool)** – подрачје кое го сочинуваат константи, стрингови и покажувачи кон други мемориски подрачја; се полни кога програмата се вчитува во меморија и потоа не се менува
 - Посебен регистер, во случајов наречен **CPP – Constant Pool Pointer**, ја содржи адресата на првиот збор од подрачјето на константи
 - **Рамка на локални променливи (Local Variable Frame)** – в. точка 4.2.1
 - Регистер LV – покажува на дното од stack-от
 - **Stack за операнди (Operand Stack)** – в. точка 4.2.1
 - Регистер SP – покажува на врвот од stack-от
 - **Подрачје на методи (Method Area)** – подрачје кое ја содржи програмата и, за разлика од другите три подрачја, има третман на поле од бајтови
 - Регистер PC – ја содржи адресата на следната инструкцијата што треба да биде преземена

4.2.2 JVM мемориски модел



4.2.3 JVM ИНСТРУКЦИСКО МНОЖЕСТВО

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

4.2.3 JVM инструкциско МНОЖЕСТВО

- Секоја инструкција содржи **код на операцијата (operation code – opcode)**, а понекогаш и **операнд** – мемориски offset, или константа
- Инструкции кои ставаат вредност (од различни извори) на врвот од stack-от
 - LCD_W – вредност од подрачјето на константи
 - ILOAD – вредност од рамката на локални променливи
 - BIPUSH – вредност содржана во самата инструкција
- Инструкција која ја зема вредноста од врвот на stack-от и ја запишува во локална променлива
 - ISTORE
- Аритметички операции – како операнди се јавуваат двата збора од врвот на stack-от, а резултатот се става повторно на врвот од stack-от
 - IADD
 - ISUB

4.2.3 JVM инструкциско МНОЖЕСТВО

- Логички операции – како операнди се јавуваат двата збора од врвот на stack-от, а резултатот се става повторно на врвот од stack-от
 - IAND
 - IOR
- Инструкции за разгранување (branch) – ја менуваат вредноста на програмскиот бројач (PC), додавајќи му 16-битен offset кој следува веднаш по кодот на операцијата
 - GOTO – безусловен скок
 - IFEQ – ако вредноста на зборот од врвот на stack-от е =0
 - IFLT – ако вредноста на зборот од врвот на stack-от е <0
 - IF_ICMPEQ – ако двете вредности од врвот на stack-от се еднакви

4.2.3 JVM инструкциско МНОЖЕСТВО

- Други инструкции за манипулирање со stack-от
 - **SWAP** – меѓусебна замена на местата на двата збора од врвот на stack-от
 - **DUP** – правење копија на зборот од врвот од stack-от
 - **POP** – земање на збор од врвот на stack-от
- Инструкција за модифицирање на следната инструкција во инструкција со долг формат
 - **WIDE**
 - Забелешка: некои инструкции имаат неколку различни формати – **WIDE ILOAD, WIDE ISTORE, ...**
- Инструкција за повикување на друга метода (процедура) и инструкција за враќање на контролата на методата која ја повикала
 - **INVOKEVIRTUAL**
 - **IRETURN**
- Инструкција која, покрај ISTORE, единствена може да менува вредност на локална (мемориска) променлива
 - **IINC** – инкрементирање на вредноста на локална променлива за одредена константна вредност