Міністерство освіти і науки, молоді та спорту України
Прикарпатський національний університет
імені Василя Стефаника

Кафедра радіофізики і електроніки

Терлецький А.І., Фрик О.Б.

БУДОВА ТА ПРОГРАМУВАННЯ 8-РОЗРЯДНОГО МІКРОПРОЦЕСОРА

методичні рекомендації до виконання лабораторних робіт
з дисципліни "Архітектура комп'ютерів" (2-й семестр)
для студентів напряму "Комп'ютерна інженерія"

Івано-Франківськ – 2012

УДК [004.431.2+004.23]:004.318
ББК 32.973-018:32.973.26-04я73
Т35

Терлецький А.І., Фрик О.Б. Будова та програмування 8-розрядного мікропроцесора. Методичні рекомендації до виконання лабораторних робіт з дисципліни "Архітектура комп'ютерів" (2-й семестр) для студентів напряму "Комп'ютерна інженерія". - Івано-Франківськ, 2012. - 88 с.

В запропонованому посібнику описано архітектуру та систему команд типового 8-розрядного мікропроцесора КР580ВМ80А (аналог Intel 8080), який вивчається в другому семестрі дисципліни "Архітектура комп'ютерів" студентами напряму підготовки "Комп'ютерна інженерія". Розглянуто основні типи команд, принципи програмування в машинних кодах, способи адресації даних, організацію та використання стеку тощо. Приклади продемонстровано за допомогою програми - емулятора, яка дозволяє покрокове виконання та відлаштування програм, а також наочне відображення роботи мікропроцесора на моніторі комп'ютера.

Рецензенти:
професор кафедри радіофізики і електроніки
Прикарпатського національного університету імені Василя Стефаника,
доктор технічних наук **Когут І. Т.**

доцент кафедри теоретичної та експериментальної фізики
Прикарпатського національного університету імені Василя Стефаника
кандидат фізико-математичних наук **Ліщинський І. М.**

Рекомендовано до друку Вченою радою фізико-технічного факультету Прикарпатського національного університету імені Василя Стефаника (протокол № 6 від 03.05.2012 р.)

ЗМІСТ

ПЕРЕДМОВА

Так склалося, що мікропроцесор Intel 8080, який появився в квітні 1974 р, зіграв надзвичайно велику роль на певному етапі розвитку обчислювальної техніки, зокрема в колишньому СССР. Основною причиною було те, що на відміну від досить жорсткої архітектури більш новітніх 16-розрядних мікропроцесорів типу LSI 11, 8-розрядний 8080 був дивовижно простий. Простота його архітектури, системи команд, мінімальна кількість периферійних мікросхем, необхідних для його функціонування, зробили цей процесор не тільки основою дешевих ігрових комп'ютерів, але і сприяли розвитку аматорського конструювання. Навіть після появи більш сучасних та потужніших мікропроцесорів та мікроконтролерів фірм Intel, Motorola чи Sun, 8080 тривалий час залишався основним мікропроцесором для створення простих ігрових та побутових пристроїв. Обчислювальні можливості цього процесора не поступаються можливостям великих ЕОМ кінця 60-х - початку 70-х років минулого сторіччя.

8080 та його вітчизняний аналог КР580ВМ80А (К580ИК80А), який було створено в 1978 р., на сьогоднішній час є прикладом класичного мікропроцесора, який вивчається студентами практично всіх комп'ютерних спеціальностей, а отримані знання та навички програмування можуть бути легко перенесені також і на більш сучасні обчислювальні пристрої.

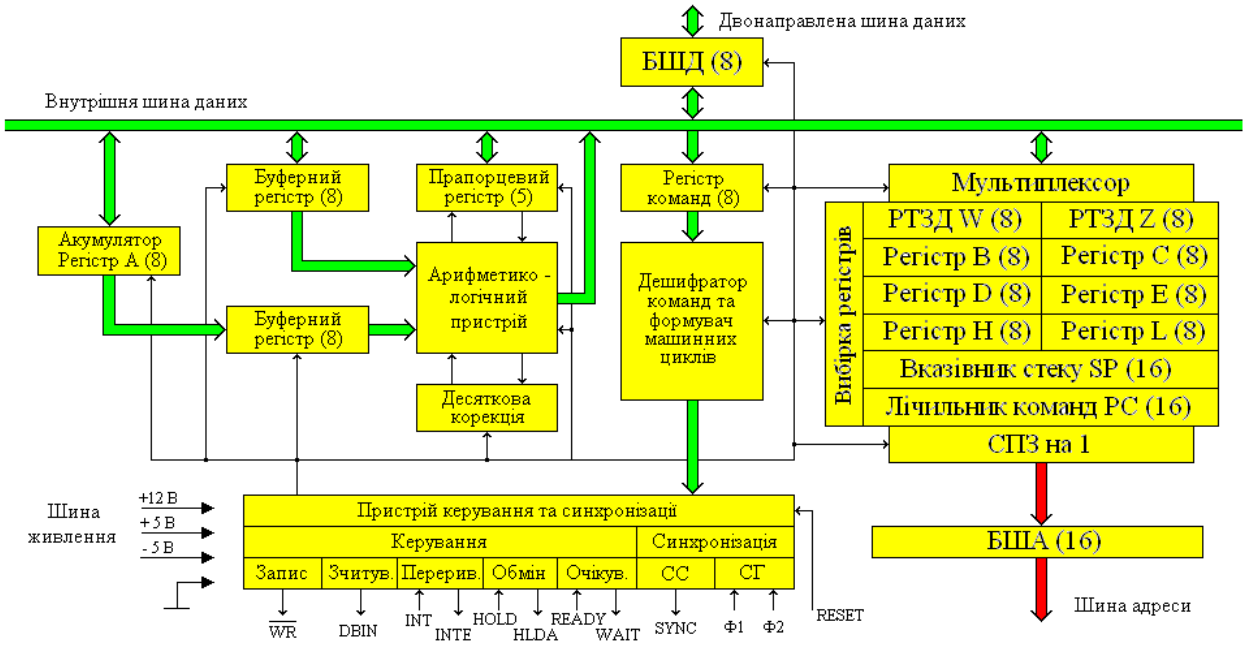
Мікропроцесор КР580ВМ80А випускався в промислових масштабах в багатьох містах колишнього СССР, в тому числі в м. Івано-Франківськ, на підприємстві "Позитрон" виробничого об'єднання "Родон", головним технологом якого та відповідальним за впровадження мікропроцесора в виробництво був д-р. техн. наук, проф. **С. П. Новосядлий.**

На першій сторінці обкладинки зображено мікропроцесор КР580ВМ80А з логотипом ВО "Родон".

Розділ 1. АРХІТЕКТУРА МІКРОПРОЦЕСОРА КР580ВМ80

1.1. Внутрішня архітектура мікропроцесора.

Мікропроцесор складається з таких основних частин (Мал.1.1):



Малюнок 1.1. Структурна схема мікропроцесора КР580ВМ80.

- арифметико-логічний пристрій (АЛП) з регістром А (акумулятор);
- блок регістрів загального призначення (РЗП) зі схемою вибірки;
- регістр команд з дешифратором команд і формувачем машинних циклів;
- лічильник команд (PC – program counter)
- регістр стекової області (SP – stack pointer - вказівник стеку);
- регістр стану (прапорцевий регістр);
- регістри тимчасового зберігання даних (РТЗД) W та Z;
- схема приросту та зменшення (СПЗ);
- пристрій керування та синхронізації.

Буфери шини даних (БШД) та адреси (БША) є додатковими блоками, які обслуговують мікропроцесор, хоча і не входять до його складу.

Число, призначення регістрів, прапорців та команд користувач змінити не може. Він може міняти тільки значення регістрів та використовувати команди в будь-якій, необхідній йому, комбінації.

Регістр – це спеціальний запам’ятовуючий пристрій, який складається з лінійки тригерів і може запам’ятовувати декілька біт інформації одночасно. Більшість регістрів мікропроцесора 8-розрядні, деякі (PC, SP) - 16-розрядні. Всі регістри розбиті на групи і відрізняються різним функціональним призначенням.

Доступними програмісту є такі регістри:

- шість 8-розрядних регістрів, до яких можна звертатися поодинці або парами (регістри B і C, D і E, H і L);
- 8-розрядний акумулятор A;
- 16-розрядні регістри PC (Program Counter - лічильник команд) та SP (Stack Pointer - вказівник стеку).

В деяких спеціальних випадках можуть бути доступними дані регістра команд (8 розрядів) та регістр стану (5 розрядів). Програмно недоступними є регістри тимчасового зберігання даних W та Z.

Регістри загального призначення. Ці регістри розміром в 1 байт позначаються B, C, D, E, H, L. Вони використовуються для зберігання даних та проміжних результатів обчислень, які виконуються за допомогою арифметико-логічного пристрою. Під час роботи з 16-розрядними числами можна звертатися до пар регістрів (B,C), (D,E), (H,L).

Акумулятор – спеціальний однобайтовий регістр, який позначається A. Під час виконання арифметичних чи логічних операцій він є джерелом одного з операндів та місцем запам'ятовування результату виконання операції. Акумулятор є основною операційною ланкою арифметико-логічного пристрою (АЛП). Він використовується також в якості місця зберігання даних та результатів операцій, які виконуються в АЛП. Акумулятор з АЛП та буферними регістрами становлять **тракт даних**.

Регістр команд – однобайтовий регістр, в якому знаходиться код команди, яка виконується. Цей регістр безпосередньо користувачу недоступний. Це не означає, однак, що не існує команди, яка б могла явним чином змінити його значення. Після виконання чергової команди в регістр команд автоматично записується код наступної команди із комірки оперативної пам'яті, адреса якої міститься в лічильнику команд (PC).

Лічильник команд (PC) – двобайтовий (16-розрядний) регістр, який часом іще називають програмним лічильником. Цей регістр містить адресу команди, яка повинна виконуватися наступною. Лічильник команд автоматично отримує приріст адреси, яку містить в залежності від того, яку по тривалості команду (одно-, дво-, чи трибайтову) мікропроцесор зчитує з пам'яті, вказуючи завжди на 1-й байт наступної команди. На зміст цього регістру користувач може вплинути тільки за допомогою команд, які змінюють послідовне виконання програми (наприклад, команд безумовного переходу), а також за допомогою деяких спеціальних команд.

Вказівник стеку (SP) – двобайтовий (16-розрядний) регістр, який містить адресу наступної комірки стеку. Стеком називається особливим чином організована область оперативної пам'яті, яка виділяється програмістом для тимчасового зберігання внутрішніх регістрів мікропроцесора зі спеціальним режимом доступу. Ця область оперативної пам'яті необхідна в тому випадку, коли необхідно припинити виконання поточної послідовності команд та вернутись до неї пізніше, наприклад для негайного виконання підпрограми чи в результаті переривання програми. Дані від мікропроцесора поступають в верхню частину стекової пам'яті, в такому випадку значення вказівника стеку зменшується на одиницю для того, щоби завжди вказувати на адресу останньої

заповненої комірки стеку (дно стеку). Коли дані вибираються зі стеку, значення вказівника стеку збільшується на одиницю з кожним вибраним байтом. Такі операції зі стеком називаються стековими. З їхньою допомогою легко організовуються багаторівневі (вкладені) переривання та звернення до підпрограм із підпрограм (вкладені підпрограми).

Регістр стану (прапорцевий регістр) – регістр, який містить 5 двійкових розрядів, що називаються прапорцями, і які містять спеціальні ознаки результатів деяких операцій. Часом його називають регістром ознак, або регістром бітів умов. Регістр містить такі прапорці: прапорець нуля (Z - zero), прапорець перенесення (C - carry), прапорець знаку (S - sign), прапорець парності (P - parity) та прапорець додаткового перенесення (AC – auxiliary carry). Прапорці завжди встановлюються чи скидаються автоматично після виконання наступної команди, яка впливає на прапорці, в залежності від результату операції. Прапорець вважається встановленим, якщо відповідний розряд регістра набуває значення 1, і скидається, якщо значення розряду 0. Стани прапорців використовують в командах умовного переходу. Результати виконання арифметичних і логічних операцій над вмістом акумулятора, регістрів загального призначення та комірок пам'яті впливають на прапорці наступним чином:

Прапорець нуля встановлюється в 1, якщо в результаті виконання якої-небудь команди отримано нульовий результат (всі біти задіяного регістру чи комірки пам'яті встановлено в 0) і скидається в 0 в випадку ненульового результату.

Прапорець перенесення встановлюється в 1, якщо в результаті операцій додавання та зсуву появляється одиниця перенесення зі старшого розряду байта даних, а також, якщо виконується запозичення зі старшого розряду після виконання операцій віднімання чи порівняння. В іншому випадку прапорець скидається в 0.

Прапорець знаку встановлюється в 1, якщо в результаті виконання операцій появляється одиниця в старшому розряді байту даних (вказує на від'ємний результат) і скидається в 0 в випадку нульового значення старшого розряду (вказує на додатній результат).

Прапорець парності встановлюється в 1, якщо після виконання операцій сума одиниць в байті даних парна (значення суми по модулю 2 рівне 0) і скидається в 0, якщо кількість одиниць непарна.

Прапорець додаткового перенесення встановлюється в 1, якщо в результаті виконання команди появляється одиниця перенесення з третього розряду байта даних в четвертий і скидається в 0, якщо такого перенесення нема. Прапорець додаткового перенесення використовується в багатьох схемах обчислень, однак він особливо необхідний для додавання чисел в двійково-десятковій формі.

1.2. Виконання команди мікропроцесором.

Мікропроцесор містить 16-розрядну однонапрямлену шину адреси та 8-розрядну двонапрямлену шину даних. Інформація, яка може одночасно

передаватись, відповідає одному байту (8 розрядів). Можливі такі передавання даних:

- пересилання байта даних від пристрою введення;
- пересилання байта даних до пристрою виведення;
- зчитування байта даних з пам’яті або запис в пам'ять;
- генерування в шину спеціального байта, який називається керуючим словом і призначений для встановлення правильного схемного з’єднання.

Робота мікропроцесора базується на принципі мікропрограмного керування. Це означає, що кожна команда реалізується як деяка послідовність мікрокоманд чи мікрооперацій, які приводять до необхідного результату. Команда, а фактично її 8-розрядний код зчитується з пам’яті і поступає в регістр команд, де і зберігається до кінця її виконання. Відповідно до результату дешифрування коду команди відбувається формування послідовності мікрокоманд (мікропрограма), процес виконання якої визначає всі наступні операції, необхідні для виконання зчитаної команди. Виконання окремих мікрооперацій синхронізується відповідно до сигналів Ф1 та Ф2 тактового генератора. Найважливішим поняттям всього процесу виконання команд є поняття **машинного циклу**.

Процес виконання кожної команди можна розбити на ряд основних операцій. Час, необхідний на виконання звернень до пам’яті чи пристроїв введення-виведення, складає машинний цикл. Виконання команди займає стільки машинних циклів, скільки необхідно звернень до пам’яті чи пристроїв введення-виведення для її виконання. Машинний цикл в свою чергу складається з машинних тактів (один машинний такт = один період тактових імпульсів синхрогенератора).

Кожна команда в залежності від її виду може займати від одного до п’яти машинних циклів. Мікропроцесор КР580ВМ80 має 10 типів машинних циклів, а кожний машинний цикл може містити від 3 до 5 машинних тактів.

1.3. Система команд мікропроцесора КР580ВМ80.

Система команд мікропроцесора КР580ВМ80 представлена 244 кодами операцій (Табл.1.1), які можна розкласифікувати згідно кількох ознак. Найбільш суттєвими ознаками є такі:

- довжина команди або число байтів, які вона займає в пам’яті;
- функціональна ознака або операції, які вона виконує;
- спосіб адресації.

Із 256 можливих кодів команд не використовуються 12 кодових комбінацій, тому число всіх команд: 256-12=244. Всі команди поділяються на три групи відповідно до кількості байтів, які вони займають в пам’яті: однобайтові, двобайтові та трибайтові. В будь-якій команді перший байт завжди містить код команди, другий і третій байт містять або дані, або адресу комірки пам’яті, яка містить дані.

Мікропроцесор КР580ВМ80 може виконувати 78 базових команд відповідно до функціональної ознаки, а саме 200 однобайтових, 18 двобайтових

та 26 трибайтових команд відповідно до довжини, яку дана команда займає в пам'яті ОЗП.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LXI B, d16	STAX B	INX B	INR B	DCR B	MVI B, d8	RLC	-	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C, d8	RRC
1	-	LXI D, d16	STAX D	INX D	INR D	DCR D	MVI D, d8	RAL	-	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E, d8	RAR
2	-	LXI H, d16	SHLD adr	INX H	INR H	DCR H	MVI H, d8	DAA	-	DAD H	LHLD adr	DCX H	INR L	DCR L	MVI L, d8	CMA
3	-	LXI SP, d16	STA adr	INX SP	INR M	DCR M	MVI M, d8	STC	-	DAD SP	LDA adr	DCX SP	INR A	DCR A	MVI A, d8	CMC
4	MOV B, B	MOV B, C	MOV B, D	MOV B, E	MOV B, H	MOV B, L	MOV B, M	MOV B, A	MOV C, B	MOV C, C	MOV C, D	MOV C, E	MOV C, H	MOV C, L	MOV C, M	MOV C, A
5	MOV D, B	MOV D, C	MOV D, D	MOV D, E	MOV D, H	MOV D, L	MOV D, M	MOV D, A	MOV E, B	MOV E, C	MOV E, D	MOV E, E	MOV E, H	MOV E, L	MOV E, M	MOV E, A
6	MOV H, B	MOV H, C	MOV H, D	MOV H, E	MOV H, H	MOV H, L	MOV H, M	MOV H, A	MOV L, B	MOV L, C	MOV L, D	MOV L, E	MOV L, H	MOV L, L	MOV L, M	MOV L, A
7	MOV M, B	MOV M, C	MOV M, D	MOV M, E	MOV M, H	MOV M, L	HLT	MOV M, A	MOV A, B	MOV A, C	MOV A, D	MOV A, E	MOV A, H	MOV A, L	MOV A, M	MOV A, A
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A
C	RNZ	POP B	JNZ adr	JMP adr	CNZ adr	PUSH B	ADI d8	RST 0	RZ	RET	JZ adr	-	CZ adr	CALL adr	ACI d8	RST 1
D	RNC	POP C	JNC adr	OUT N	CNC adr	PUSH C	SUI d8	RST 2	RC	-	JC adr	IN N	CC adr	-	SBI d8	RST 3
E	RPO	POP D	JPO adr	XTHL	CPO adr	PUSH D	ANI d8	RST 4	RPE	PCHL	JPE adr	XCHG	CPE adr	-	XRI d8	RST 5
F	RP	POP PSW	JP adr	DL	CP adr	PUSH PSW	ORI d8	RST 6	RM	SPHL	JM adr	EI	CM adr	-	CPI d8	RST 7

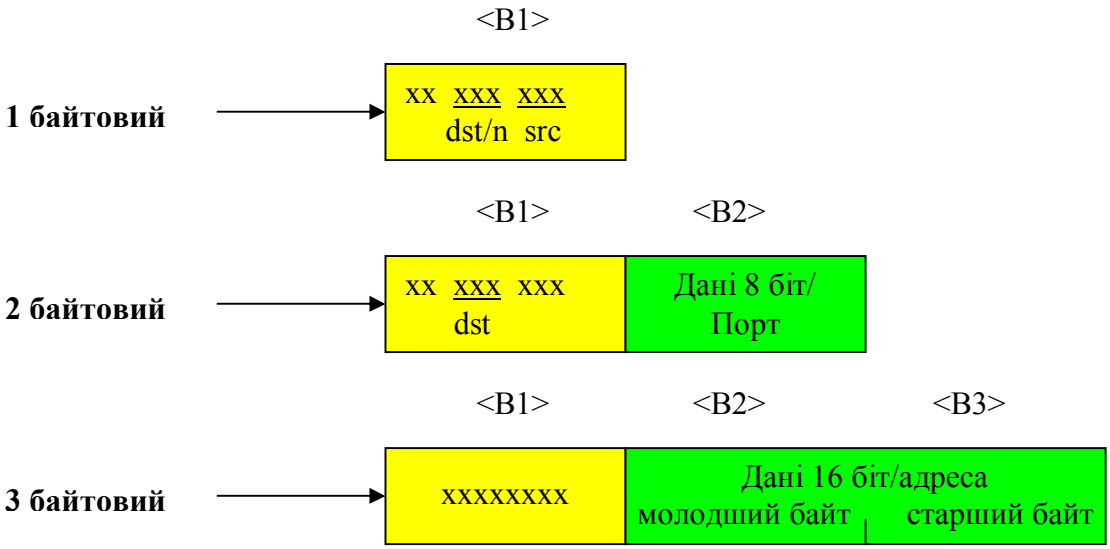
Таблиця 1.1. Команди мікропроцесора КР580ВМ80.

Для всіх можливих пересилань даних з регістрів в регістри, чи з пам'яті в регістри та навпаки, розрізняють регістри – джерела даних, які позначаються символом S (source – джерело), та регістри – приймачі даних, які позначаються символом D (destination – місце призначення). В регістрових парах (B, C), (D, E) та (H, L) старшими є перші регістри пар. Коди регістрів загального призначення, пар регістрів та прапорців строго фіксовані (Табл.1.2).

Регістр	Код	Пара регістрів	Код	Мнемонічне позначення	Код
A	111	B (B, C)	00	NZ (Z=0)	000
B	000	D (D, E)	01	Z (Z=1)	001
C	001	H (H, L)	10	NC (CY=0)	010
D	010	SP	11	C (CY=1)	011
E	011			PO (P=0)	100
H	100			PE (P=1)	101
L	101			P (S=0)	110
M (пам'ять)	110			M (S=1)	111

Таблиця 1.2. Коди регістрів та прапорців мікропроцесора КР580ВМ80.

Мікропроцесор КР580ВМ80 використовує доволі простий формат команд:



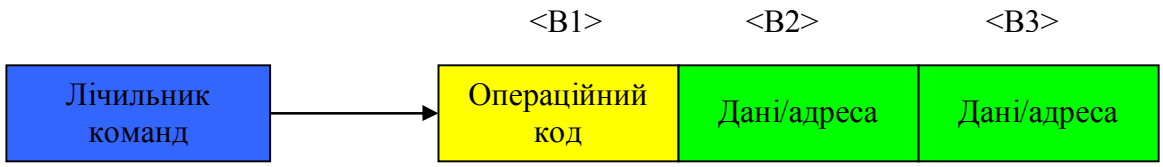
dst (Destination) адресат - приймач;
src (Source) адресат - джерело;
n = 0..7;

Код операції завжди розміщений в першому байті програми. кожен біт відмічено "x". В першому біті може знаходитися інформація про місцезнаходження операндів dst, src або ціле число 0..7. Другий, а якщо необхідно, і третій байти відводяться під безпосередні дані, адресу порту чи комірки пам'яті.

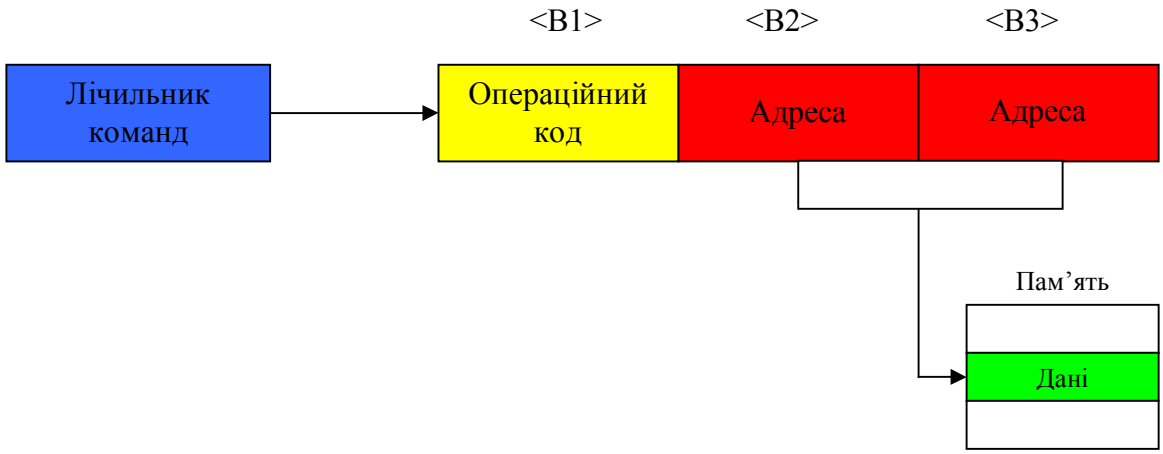
Для мікропроцесора КР580ВМ80 існують чотири можливих способи адресації: безпосередня, пряма, регістрова, та непряма.

Безпосередня адресація є найбільш економічним способом зберігання та пошуку інформації, оскільки необхідні дані містить сама команда. Ці дані містяться в другому та третьому байтах трибайтової команди та в другому байті

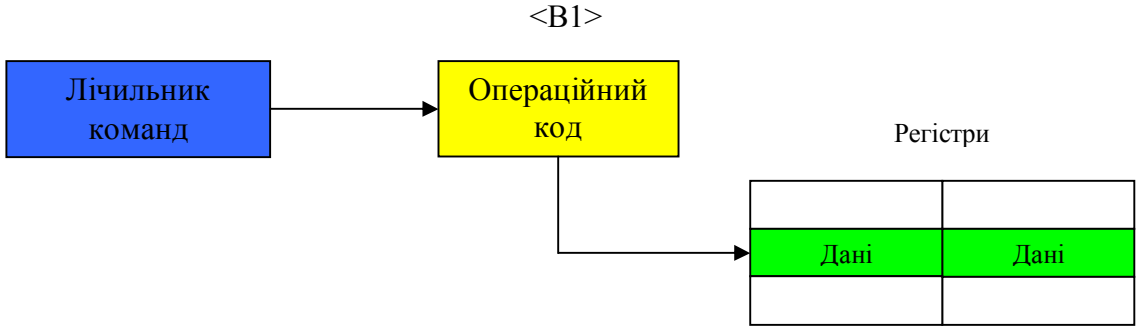
двобайтової команди. В випадку трибайтової команди молодші розряди 16-бітового числа містяться в другому байті, а старші – в третьому байті команди.



Менш економічною, але також доволі простою є **пряма адресація**. В такому випадку в другому і третьому байтах міститься повна 16-бітна адреса комірки пам’яті, в якій знаходяться дані. Молодшим байтом адреси є другий байт команди, а старшим – третій байт команди.

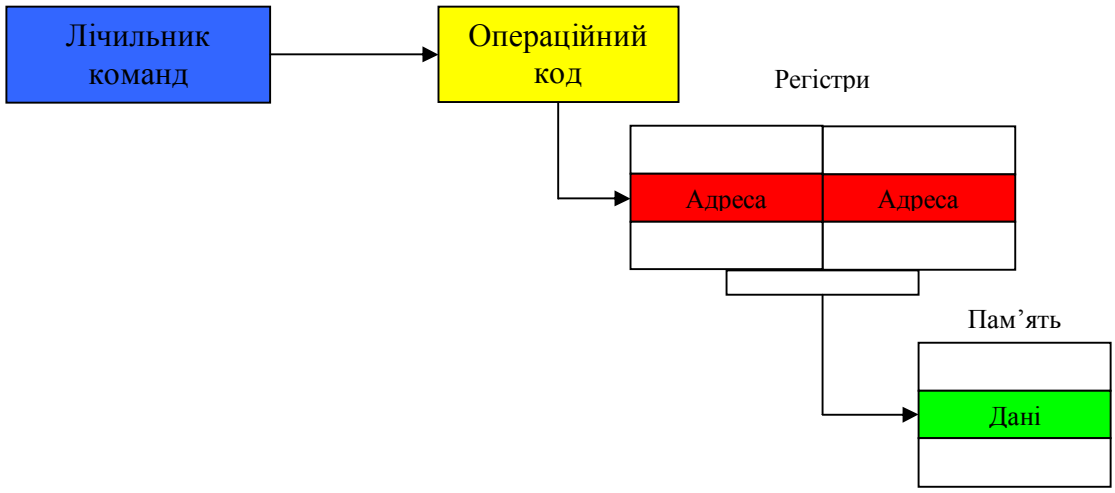


В випадку **регістрової адресації** код команди вказує на регістр або пару регістрів в яких містяться дані. Команди, які використовують регістрову адресацію є однобайтними. В такому випадку адреси регістрів задаються за допомогою трьох або 6-ти молодших бітів команди.



Непряма адресація відрізняється від регістрової тільки тим, що в регістровій парі містяться не дані, а повна 16-розрядна адреса комірки пам’яті, яка містить дані. Старший байт адреси записується в першому регістрі пари, а молодший в другому. Зазвичай вказівником адреси в випадку непрямої адресації є пара регістрів H, тобто регістри (H, L), але інколи використовуються пари (B, C) і (D, E).

<B1>



Для всіх можливих пересилань даних з регістрів в регістри, чи з пам'яті в регістри та навпаки, розрізняють регістри – джерела даних, які позначаються символом S (source – джерело), та регістри – приймачі даних, які позначаються символом D (destination – місце призначення). В регістрових парах (B, C), (D, E) та (H, L) старшими є перші регістри пар. Коди регістрів загального призначення, пар регістрів та прапорців строго фіксовані.

Всі команди відповідно до функціональної ознаки можуть бути розбиті на п'ять груп:

- група команд пересилання даних;
- арифметичні команди;
- логічні команди;
- команди переходів;
- команди управління і роботи зі стеком.

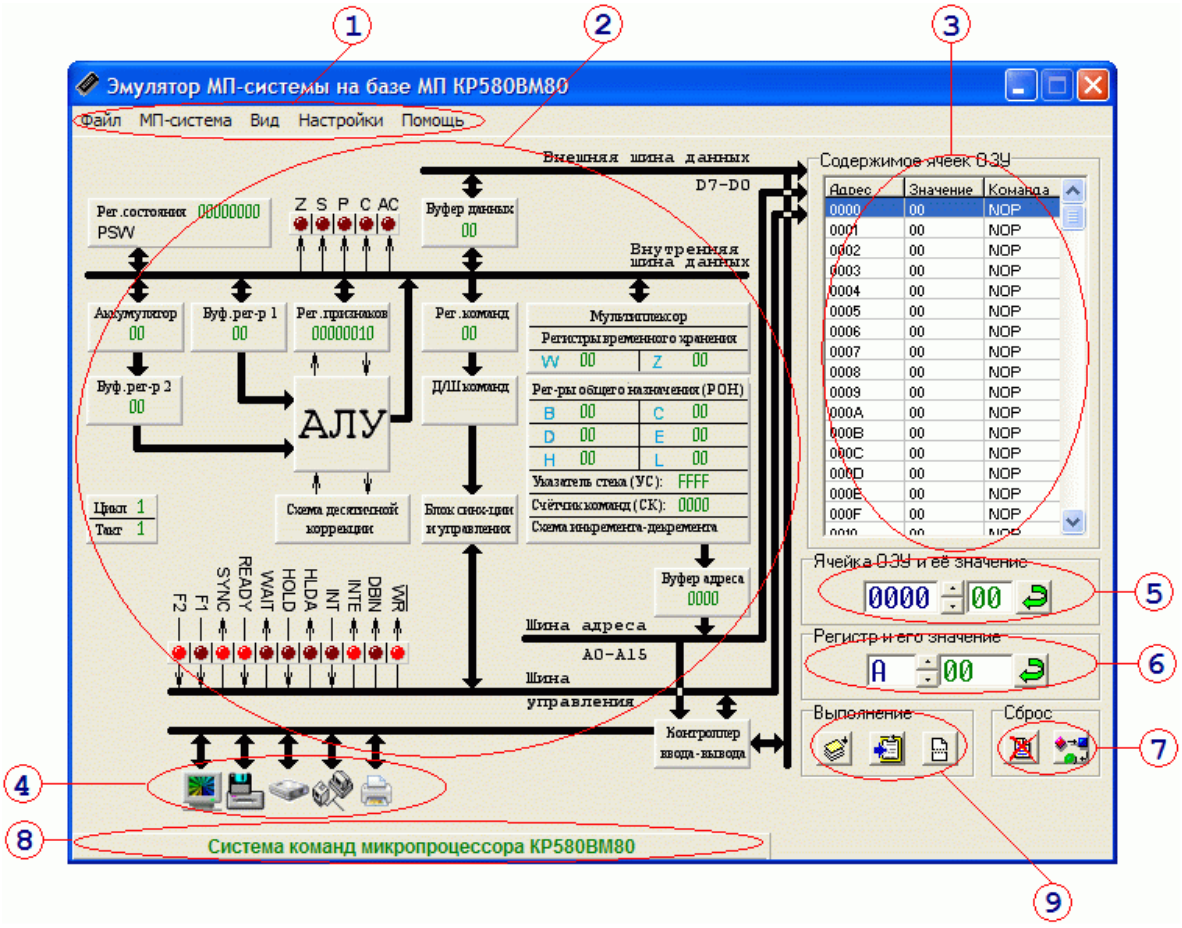
1.4. Емулятор мікропроцесорної системи на базі КР580ВМ80.

Емулятор – це програма, яка відтворює всі процеси, що відбуваються в реальних системах на екрані монітору. Даний емулятор позволяє створювати програми на мові асемблера, використовуючи систему команд мікропроцесора КР580ВМ80, налагоджувати їх виконання в тактовому, командному та наскрізному режимах, вивчати принципи і порядок виконання команд, отримувати уяву про організацію зовнішньої та внутрішньої (регістрової) пам'яті та стекової області.

Головне вікно програми має вигляд, представлений на малюнку 2:

Зміст головного вікна програми:

1. Головне меню програми;
2. Структурна схема мікропроцесорної системи;
3. Таблиця змісту ОЗП мікропроцесорної системи
4. Зовнішні периферійні пристрої, які під'єднані до портів МП системи;
5. Панель редагування значення вибраної (поточної) комірки ОЗП;



Малюнок 1.2. Вигляд головного вікна програми емулятора.

6. Панель редагування значення вмісту вибраного регістра загального призначення МП системи;
7. Група кнопок «Сброс» для обнулення всіх комірок ОЗП та регістрів загального призначення;
8. Панель системи команд мікропроцесора КР580ВМ80;
9. Група кнопок «Выполнение» для виконання програми в наскрізному, командному та тактовому режимах.

Структурна схема містить такі елементи:

- регістр слова стану мікропроцесора (PSW) і регістр ознак (прапорців), а також їх в двійковій системі числення та словесній формі;
- буфер даних МП системи, регістр А (акумулятор), буферні регістри, регістр команд, блок регістрів загального призначення, регістри тимчасового зберігання даних, регістри – вказівник стеку та лічильник команд, а також їхні значення в шістнадцятірковій системі числення;
- блок АЛП та десяткової корекції;
- блок синхронізації та керування;
- дешифратори команд та лічильники машинних мікроциклів;
- Індикатори стану і тактування мікропроцесора: F1, F2, SYNC, READY, WAIT, HOLD, HLDA, INT, INTE, DBIN, WR;

- порти системи від 00h до 04h для монітору, дисководу, мережевого адаптера та принтера відповідно;
- шини даних, адреси, керування, внутрішня шина та шина зовнішніх пристроїв.

ОЗП МП системи представлено в вигляді блоку-таблиці:

- адреса ОЗП – кожна комірка має адресу від 0000h до FFFFh (від 0d до 65535d) всього 64К комірок;
- значення комірки ОЗП – поточне значення вибраної комірки ОЗП (8 біт). Представлено в шістнадцятірковій системі числення від 00h до FFh (від 0d до 255d) всього 256 значень;
- команда мікропроцесора – розшифроване значення комірки ОЗП в виді мнемо коду на мові асемблера. Деякі комірки можуть містити не команди, а дані, однак їхні значення будуть перекодовані в мнемокод автоматично.

Адреса вибраної комірки автоматично відображається в регістрі – лічильнику команд.

В нижній області ОЗП встановлено виділення коричневим кольором тієї комірки, на яку вказує вказівник стеку. Стекова область виділена жовтим кольором.

Зовнішні периферійні пристрої під'єднані до загальної шини контролера введення-виведення. Всього до МП системи під'єднано п'ять віртуальних пристроїв: монітор, який може працювати як в графічному, так і текстовому режимах; накопичувачі на гнучких та твердих магнітних дисках; мережевий адаптер та принтер. Всі пристрої, крім монітору, можуть також працювати в режимі реального часу з реальними фізичними периферійними пристроями.

Панелі редагування значення комірок ОЗП чи регістрів дозволяють вводити програми та задавати початкові умови, а також редагувати введені чи зчитані з файлу програми.

Комірки «Сброс» дозволяють обнуляти всі комірки ОЗП, або регістри МП системи.

Панель системи команд мікропроцесора КР580ВМ80 представлена в вигляді прихованої таблиці 16х16 рядки і стовпчики якої пронумеровані в шістнадцятірковій системі числення. Їхня послідовна комбінація (рядок-стовпчик) є кодом вибраної команди. Всі команди умовно розбиті на дванадцять груп, об'єднаних за функціональною ознакою, кожна з яких позначена своїм кольором комірки. Панель системи можна активізувати за допомогою клавіші «Space» або наводячи на неї курсор мишки. Панель полегшує програмування емулятора, оскільки вибрані команди можна «перетягувати» в комірки ОЗП за допомогою лівої клавіші мишки, а права клавіша позволяє отримати повну інформацію про команду.

Група кнопок «Выполнение».

- «Виконати такт» - це виконати один такт команди ОЗП на яку вказує лічильник команд. Якщо команда виконана на повністю, деякі елементи керування стають недоступними для редагування., а ті елементи, які є активними в даному такті відмічаються червоним кольором;
- «Виконати команду» позволяє виконати програму покомандно і контролювати її виконання та стан регістрів і пам'яті;
- «Виконати програму» запускає програму на виконання до тих пір, поки дана кнопка не буде натиснута повторний раз, або в випадку команди HLT(76h).

Основні принципи роботи з програмою.

Початок роботи з програмою полягає в написанні або завантаженні програми на асемблері в емулятор. Для цього можна скористатися або панеллю системи команд програми, або панеллю редагування значень комірок ОЗП емулятора, або завантажити образ ОЗП з носія.

В випадку необхідності можна заповнити відповідними значеннями регістри загального призначення емулятора.

Для детального вивчення кожного такту конкретної команди можна скористатися кнопкою по тактового виконання команди. Для налаштовування програми використовується кнопка по командного виконання.

Написану програму на мові асемблера можна зберегти в вигляді образа ОЗП та РЗП на будь-який носій, а також в випадку необхідності завантажити з носія в емулятор.

Програма також позволяє можливість експорту частини ОЗП та/чи РЗП емулятора в MS Excel, MS Word та текстовий файл.

Розділ 2. КОМАНДИ ПЕРЕСИЛАННЯ ДАНИХ

2.1. Способи адресації в командах пересилання даних.

Команди пересилання даних використовуються мікропроцесором для передавання даних в різні пристрої зберігання інформації, які є в його розпорядженні, як в регістри, так і в комірки оперативної пам'яті. Розрізняють такі типи команд: **завантаження, пересилання регістр - регістр, запис в пам'ять та зчитування з пам'яті (пересилання пам'ять - регістр)**.

Команди пересилання даних фактично є командами копіювання даних, оскільки під час виконання команди дані, які містить джерело, не руйнуються. Код операції в команді пересилання даних задає **джерело даних S** (Source), **приймач даних D** (Destination) і спосіб їхньої адресації (Коди джерел та приймачів даних наведено в Табл.2.1). Мікропроцесор КР580ВМ80, як відомо, використовує чотири види адресації: безпосередню, пряму, регістрову та непряму. Кожен із видів адресації може бути використаний як для джерела так і для приймача даних. З шістнадцяти можливих комбінацій в мікропроцесорі КР580ВМ80 для команд завантаження даних використовується сім:

безпосередня / регістрова
безпосередня / непряма
регістрова / регістрова
регістрова / пряма
регістрова / непряма
пряма / регістрова
непряма / регістрова

S/D	код	S/D	код
Регістр B	000b / 0q	Регістр H	100b / 4q
Регістр C	001b / 1q	Регістр L	101b / 5q
Регістр D	010b / 2q	Память M	110b / 6q
Регістр E	011b / 3q	Регістр A	111b / 7q

Таблиця 2.1. Коди джерел та приймачів даних.

Безпосередня адресація використовується в тому випадку, коли необхідно завантажити початкові дані в регістри або пам'ять, чи константи, які будуть використовуватися на протязі роботи програми. Безпосередня адресація є зручною, наочною і може використовуватись для початкової розробки програми. Недоліком є те, що програми, написані з її використанням не мають достатньо гнучкості, а саме: для того, щоби змінити значення констант або початкових даних, необхідно переписувати код програми. Також недоліком є те, що програмний код займає більше комірок пам'яті (2, або 3) та виконується за більше циклів (2 або 3), ніж в випадку регістрової адресації. Безпосередня адресація часто використовується для завантаження одного з регістрів початковим значенням кількості циклів, які необхідно виконати, та наступним зменшенням значення регістра на 1 після виконання тіла циклу.

Регістрова адресація використовується головним чином для пересилання проміжних даних які виникають в процесі роботи програми. Пересилання між регістрами займають найменше часу (1 цикл, 4 або 5 тактів), тому їх використовують для скорочення часу виконання програм. Програмний код займає найменше пам'яті, що теж є перевагою даного виду адресації. Недоліком регістрової адресації є те, що кількість регістрів для зберігання проміжних даних є обмежена (6 однобайтних **B, C, D, E, H, L**), що недостатньо для роботи навіть не дуже складних програм. Тому одним із принципів розробки перспективних процесорів RISC є вимога наявності великої кількості однотипних регістрів.

Пряма адресація використовується тоді, коли кількість проміжних даних перевищує кількість наявних внутрішніх регістрів. Як правило, програміст виділяє під зберігання вхідних, проміжних та вихідних даних програми деяку область оперативної пам'яті, яка не повинна перекриватись з областю програмного коду та областю стека. Якщо таке перекриття відбувається, то результат дії програми неможливо передбачити. Область даних оперативної пам'яті в такому випадку чітко визначається кількістю змінних програми, тому мови високого рівня, які написані, в основному, на мові асемблера чи машинному коді, вимагають попереднього оголошення змінних на початку роботи програми. Область даних програми зручно розміщувати в тілі програми, для того щоби завантаження підпрограм зі зміщеним кодом не перекривалось з цією областю. Незважаючи на те, що пряма адресація вимагає трибайтних команд, та додаткових двох циклів для завантаження адреси комірки пам'яті + 1 цикл завантаження даних з комірки пам'яті, використання її є більш ефективним, ніж використання непрямої адресації, для якої спочатку слід забезпечити занесення адреси комірки пам'яті в відповідну пару регістрів.

Непряма адресація використовується в основному для роботи з областю стеку, а також для зберігання та завантаження даних в/з суміжних комірок пам'яті, які використовуються для організації масивів. За допомогою непрямої адресації також можна організовувати масиви змінної довжини типу вказівник або список. Програмний код з командами, які використовують непряму адресацію займає менше місця, є достатньо гнучким та зручним для перенесення на інші процесори. В випадку організації списків чи вказівників, необхідно чітко слідкувати за тим, щоби кількість байтів, занесених в пам'ять, завжди рівнялась кількості вибраних, в іншому випадку результат дії програми буде непередбачуваним.

В таблиці (1.1. розділ 1) всі команди пересилання даних відмічені оранжевим (однобайтові пересилання) та жовтим (двобайтові пересилання) кольорами.

2.2. Машинні цикли мікропроцесора КР580ВМ80.

Команди в КР580ВМ80 виконуються як послідовність внутрішніх мікрокоманд. Кожна з мікрокоманд виконується на протязі одного циклу, тривалість якого складає 3-5 тактів. Кількість машинних циклів та загальна

кількість тактів визначається типом команди. Всього є 10 типів машинних циклів:

вибірка команди;
зчитування слова з запам'ятовуючого пристрою
запис слова в запам'ятовуючий пристрій
зчитування слова зі стеку
запис слова в стек
зчитування слова з пристрою введення/виведення
запис слова в пристрій введення/виведення
підтвердження переривання
підтвердження зупинки
підтвердження переривання під час зупинки.

Перший цикл завжди є цикл вибірки команди і займає 4 або 5 тактів. Три наступних цикли завжди виконуються за три такти, а п'ятий за три або п'ять тактів. З огляду на це завжди можна розрахувати кількість циклів і тактів команди, якщо відомо способи адресації, які вона використовує.

Наприклад: команда **MVI M, B_2** двобайтова, використовує безпосередню/непряму адресацію. Під час виконання команди дані, які містяться в другому байті програми, завантажуються в комірку пам'яті, адреса якої знаходиться в регістровій парі **HL.** Перший цикл - вибірка першого байта з пам'яті (4 або 5 тактів), наступний - вибірка другого байта (3 такти). Ще один цикл - запис даних в комірку пам'яті (3 такти). Отже команда виконується за 3 цикли 10 або 11 тактів. (насправді 10 тактів).

2.3.Команди пересилання мікропроцесора КР580ВМ80.

Команди мікропроцесора КР580ВМ80 зручно вивчати згідно такої схеми:

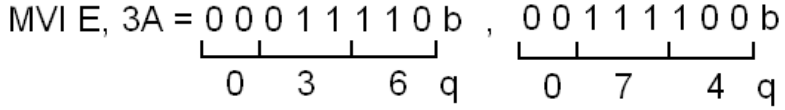
<table>
<tr><td>1</td><td colspan="2">Назва</td></tr>
<tr><td>2</td><td colspan="2">Способи адресації, які використовує команда</td></tr>
<tr><td>3</td><td colspan="2">Мнемонічна форма запису</td></tr>
<tr><td>4</td><td colspan="2">Дії, які виконує команда в формі логічних символів</td></tr>
<tr><td rowspan="2">5</td><td colspan="2">Машинний код команди в</td></tr>
<tr><td>шістнадцятковій системі</td><td>вісімковій системі</td></tr>
<tr><td>6</td><td colspan="2">Довжина команди, кількість циклів, кількість тактів</td></tr>
<tr><td>7</td><td colspan="2">Опис дій команди</td></tr>
<tr><td>8</td><td colspan="2">Дія результату виконання команди на регістр станів мікропроцесора</td></tr>
</table>

<u>Увага!</u> Наведена схема відноситься не тільки до даної лабораторної роботи, але і до наступних робіт з вивчення системи команд мікропроцесора КР580ВМ80.

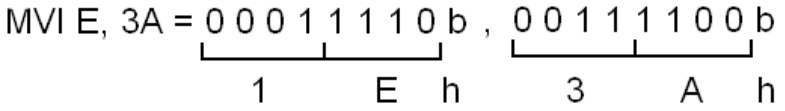
2.3.1. Команди завантаження.

1	Завантаження регістра	
2	безпосередня / регістрова	
3	**MVI r, B_2**	
4	**B_2 → r**	
5	06h, 16h, 26h, 0Eh, 1Eh, 2Eh, 3Eh	0D6q (D ≠ 6)
6	Команда займає 2 байта пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, які знаходяться в другому байті програми, завантажуються в регістр за номером D.	
8	Виконання команди не впливає на регістр стану.	

Номер D береться з Табл.1. Наприклад для регістра **E** код 011b, або 3q. Тому команда завантаження регістра **E** числом, наприклад 3Ah, виглядає таким чином: **MVI E,** 3A = 1E, 3A, або в вісімковій системі числення



чи в шістнадцятковій системі числення



1	Завантаження регістрової пари **BC** (**DE, HL**)	
2	безпосередня / регістрова	
3	**LXI B, B_2, B_3 (LXI D, B_2, B_3; LXI H, B_2, B_3)**	
4	**B_3 → B, B_2 → C (B_3 → D, B_2 → E; B_3 → H, B_2 → L)**	
5	01h (11h; 21h)	001q (021q; 041q)
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Під час виконання команди третій байт команди записується в регістр **B** (**D**, **H**), другий байт команди записується в регістр **C** (**E**, **L**).	
8	Виконання команди не впливає на регістр стану.	

1	Завантаження регістра **SP** - вказівника стеку	
2	безпосередня / регістрова	
3	**LXI SP, B_2, B_3**	
4	**B_3, B_2 → SP**	
5	31h	061q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Під час виконання команди другий (молодший) та третій (старший) байти команди записується в регістр **SP**.	
8	Виконання команди не впливає на регістр стану.	

Команда **LXI SP** буде детально розглядатись в лабораторній роботі по вивченню стеку і команд роботи з ним.

1	Завантаження комірки пам'яті	
2	безпосередня / непряма	
3	**MVI M, B_2**	
4	**B_2 → M(H, L)**	
5	36h	066q
6	Команда займає 2 байта пам'яті, виконується за 3 цикли, 10 тактів.	
7	Під час виконання команди дані, які містяться в другому байті програми, завантажуються в комірку пам'яті, адреса якої знаходиться в регістровій парі **HL**.	
8	Виконання команди не впливає на регістр стану.	

Перед завантаженням комірки пам'яті командою **MVI M, B_2** необхідно завантажити пару **HL** адресою комірки.

2.3.2. Команди пересилання регістр-регістр

1	Пересилання з регістра в регістр	
2	регістрова / регістрова	
3	**MOV r1, r2**	
4	**r2 → r1**	
5	40h-45h, 47h-4Dh, 4Fh 50h-55h, 57h-5Dh, 5Fh 60h-65h, 67h-6Dh, 6Fh 78h-7Dh, 7Fh	1DSq (D ≠ 6, S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 5 тактів.	
7	Під час виконання команди дані, які містяться в регістрі за номером S пересилаються в регістр за номером D.	
8	Виконання команди не впливає на регістр стану.	

Номери D та S беруться з Табл.1. аналогічно, як і для команди **MVI r, B_2.**

1	Обмін даними між регістрами **HL** та **DE**	
2	регістрова / регістрова	
3	**XCHG**	
4	**H ↔ D, L ↔ E**	
5	EBh	353q
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди регістри **H, L** обмінюються даними з регістрами **D, E**.	
8	Виконання команди не впливає на регістр стану.	

1	Завантаження регістра **SP** - вказівника стеку	
2	регістрова / регістрова	
3	**SPHL**	
4	**HL → SP**	
5	F9h	371q
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 5 тактів.	
7	Під час виконання команди дані, які містяться регістрі **HL** завантажуються у вказівник стеку **SP**.	
8	Виконання команди не впливає на регістр стану.	

Команда **SPHL** буде детально розглядатись в лабораторній роботі по вивченню стеку і команд роботи з ним.

1	Завантаження лічильника команд **PC** (непрямий перехід)	
2	регістрова / регістрова	
3	**PCHL**	
4	**HL → PC**	
5	E9h	351q
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 5 тактів.	
7	Під час виконання команди дані, які містяться в регістрі **L** пересилаються в молодший байт лічильника команд **PC**, а дані, які містяться в регістрі **H** - в старший байт. Керування передається команді, яка буде занесена в **PC**.	
8	Виконання команди не впливає на регістр стану.	

Команда **PCHL** фактично є командою безумовного переходу і до команд пересилання даних віднесена завдяки подібності порядку виконання.

2.3.3. Команди запису в пам'ять

1	Пересилання з регістра в пам'ять	
2	регістрова / непряма	
3	**MOV M, r**	
4	**r → M(H, L)**	
5	70h - 75h, 77h	16Sq (S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, які містяться в регістрі за номером S пересилаються в комірку пам'яті, адреса якої знаходиться в регістровій парі **HL**.	
8	Виконання команди не впливає на регістр стану.	

Перед пересиланням в комірку пам'яті командою **MOV M, r** необхідно завантажити пару **HL** адресою комірки.

1	Збереження акумулятора	
2	регістрова / пряма	
3	**STA, B_2, B_3**	
4	**A → M(B_3, B_2)**	
5	32h	062q
6	Команда займає 3 байта пам'яті, виконується за 4 цикли, 13 тактів.	
7	Під час виконання команди дані, які містяться в акумуляторі **A**, пересилаються в комірку пам'яті, адреса якої знаходиться в другому та третьому байтах команди. (**B_2** - молодший байт адреси, **B_3** - старший байт адреси)	
8	Виконання команди не впливає на регістр стану.	

1	Збереження акумулятора	
2	регістрова / непряма	
3	**STAX B (STAX D)**	
4	**A → M(B, C) (A → M(D, E))**	
5	02h (12h)	002q (022q)
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, які містяться в акумуляторі **A**, пересилаються в комірку пам'яті, адреса якої знаходиться в регістровій парі **BC** (**DE**). (**B** (**D**) - старший байт адреси, **C** (**E**) - молодший байт адреси)	
8	Виконання команди не впливає на регістр стану.	

Перед збереженням в комірку пам'яті командою **STAX B (STAX D)** необхідно завантажити відповідну регістрову пару адресою комірки.

1	Збереження регістрової пари **HL**	
2	регістрова / пряма	
3	**SHLD, B_2, B_3**	
4	**L → M(B_3, B_2), H → M(B_3, B_2+1)**	
5	22h	042q
6	Команда займає 3 байти пам'яті, виконується за 5 цикли, 16 тактів.	
7	Під час виконання команди дані, які містяться в регістрі **L** пересилаються в комірку пам'яті, адреса якої знаходиться в другому та третьому байтах команди. Дані, які містяться в регістрі **H** пересилаються в наступну комірку пам'яті. (**B_2** - молодший байт адреси, **B_3** - старший байт адреси)	
8	Виконання команди не впливає на регістр стану.	

2.3.4. Команди зчитування з пам'яті

1	Пересилання з пам'яті в регістр	
2	непряма / регістрова	
3	**MOV r, M**	
4	**M (H, L)→ r**	
5	46h, 56h, 66h, 4Eh, 5Eh, 6Eh, 7Eh	1D6q (D ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, які містяться в комірці пам'яті, адреса якої знаходиться в регістровій парі **HL**, пересилаються в регістр за номером D.	
8	Виконання команди не впливає на регістр стану.	

1	Завантаження акумулятора	
2	пряма / регістрова	
3	**LDA, B_2,B_3**	
4	**M(B_3, B_2) → A**	
5	3Ah	072q
6	Команда займає 3 байта пам'яті, виконується за 4 цикли, 13 тактів.	
7	Під час виконання команди дані, які містяться в комірці пам'яті, адреса якої знаходиться в другому та третьому байтах команди, пересилаються в акумулятор **A**.. (**B_2** - молодший байт адреси, **B_3** - старший байт адреси)	
8	Виконання команди не впливає на регістр стану.	

1	Завантаження акумулятора	
2	непряма / регістрова	
3	**LDAX B (LDAX D)**	
4	**M(B, C)→ A (M(C, E)→ A)**	
5	0Ah (1Ah)	012q (032q)
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, які містяться в комірці пам'яті, адреса якої знаходиться в регістровій парі **BC** (**DE**), пересилаються в акумулятор **A.**. (**B** (**D**) - старший байт адреси, **C** (**E**) - молодший байт адреси)	
8	Виконання команди не впливає на регістр стану.	

1	Завантаження регістрової пари **HL**	
2	пряма / регістрова	
3	**LHLD, B_2, B_3**	
4	**M(B_3, B_2) → L, M(B_3, B_2+1) → H**	
5	2Ah	051q
6	Команда займає 3 байти пам'яті, виконується за 5 циклів, 16 тактів.	
7	Під час виконання команди дані, які містяться в комірці пам'яті,	

	адреса якої знаходиться в другому та третьому байтах команди, пересилаються в регістр **L**. В регістр **H** пересилаються дані наступної комірки пам'яті. (**B_2** - молодший байт адреси, **B_3** - старший байт адреси)
8	Виконання команди не впливає на регістр стану.

До приведеного списку команд не ввійшли команди роботи зі стеком, які будуть вивчатися в одній із наступних лабораторних робіт.

Розділ 3. АРИФМЕТИЧНІ КОМАНДИ

Якими б потужними та складними пристроями не рахували комп'ютери, всі вони побудовані на основі мікропроцесорів, які можуть виконувати тільки найпростіші арифметичні та логічні операції. До них відноситься операції ДОДАВАННЯ та ВІДНІМАННЯ. Такі операції, як МНОЖЕННЯ та ДІЛЕННЯ реалізуються програмним методом за допомогою операцій ДОДАВАННЯ, ВІДНІМАННЯ та деяких інших логічних операцій. В загальному випадку операція ВІДНІМАННЯ зводиться до операції ДОДАВАННЯ числа, записаного в оберненому коді, однак реалізація віднімання таким способом вимагає значних програмних витрат (збільшує довжину коду програми та ускладнює її).

Мікропроцесор КР580ВМ80 може виконувати операції ДОДАВАННЯ над 8-ми та 16-ти розрядними двійковими числами. Операції ВІДНІМАННЯ застосовні тільки до 8-ми розрядних чисел.

Виконання арифметичних команд впливає на значення регістру стану (прапорцевого регістру)

3.1. Регістр стану.

Регістр стану (прапорцевий регістр) – регістр, який містить 5 двійкових розрядів, що називаються прапорцями, і які містять спеціальні ознаки результатів деяких операцій. Часом його називають регістром ознак, або регістром бітів умов. Регістр містить такі прапорці: прапорець нуля (Z - zero), прапорець перенесення (C - carry), прапорець знаку (S - sign), прапорець парності (P - parity) та прапорець додаткового перенесення (AC – auxiliary carry). Прапорці завжди встановлюються чи скидаються автоматично після виконання наступної команди, яка впливає на прапорці, в залежності від результату операції. Прапорець вважається встановленим, якщо відповідний розряд регістра набуває значення 1, і скидається, якщо значення розряду 0. Стани прапорців використовують в командах умовного переходу. Результати виконання арифметичних і логічних операцій над вмістом акумулятора, регістрів загального призначення та комірок пам’яті впливають на прапорці наступним чином:

Прапорець нуля встановлюється в 1, якщо в результаті виконання якої-небудь команди отримано нульовий результат (всі біти задіяного регістру чи комірки пам’яті встановлено в 0) і скидається в 0 в випадку ненульового результату.

Прапорець перенесення встановлюється в 1, якщо в результаті операцій додавання та зсуву появляється одиниця перенесення зі старшого розряду байта даних, а також, якщо виконується запозичення зі старшого розряду після виконання операцій віднімання чи порівняння. В іншому випадку прапорець скидається в 0.

Прапорець знаку встановлюється в 1, якщо в результаті виконання операцій появляється одиниця в старшому розряді байту даних (вказує на від’ємний результат) і скидається в 0 в випадку нульового значення старшого розряду (вказує на додатній результат).

Прапорець парності встановлюється в 1, якщо після виконання операцій сума одиниць в байті даних парна (значення суми по модулю 2 рівне 0) і скидається в 0, якщо кількість одиниць непарна.

Прапорець додаткового перенесення встановлюється в 1, якщо в результаті виконання команди появляється одиниця перенесення з третього розряду байта даних в четвертий і скидається в 0, якщо такого перенесення нема. Прапорець додаткового перенесення використовується в багатьох схемах обчислень, однак він особливо необхідний для додавання чисел в двійково-десятковій формі.

Деякі з арифметичних команд можуть враховувати значення окремих прапорців стану, які були встановлені в результаті виконання попередніх операцій. До таких команд належить додавання з врахуванням попереднього перенесення, та команди, що працюють з двійково-десятковими числами.

3.1.1. 8-ми розрядні додавання та віднімання.

Всі 8-ми розрядні додавання здійснюються за допомогою акумулятора **A**. Це означає, що один із операндів обов'язково повинен міститися в регістрі **A**. Другий із операндів може бути адресований одним із трьох видів адресації: безпосередньою, регістровою та непрямою. Пряма адресація операнда не використовується в даному мікропроцесорі, однак може бути реалізована в інших. Результат додавання завжди записується в акумулятор **A**. Іншими словами процесор виконує додавання шляхом збільшення значення акумулятора **A**, що містить один доданок, на величину другого доданку, який може знаходитися в одному із регістрів мікропроцесора, або комірці зовнішньої (оперативної) пам'яті, або безпосередньо в наступному байті команди. Слід пам'ятати, що попереднє значення акумулятора **A** втрачається. Це необхідно враховувати в випадку розробки програм та забезпечувати зберігання проміжних значень акумулятора в разі потреби.

В випадку 8-ми розрядних віднімань зменшуване завжди повинно міститися в акумуляторі **A**. Від'ємник адресується аналогічно, як це здійснюється у випадку додавання. Різниця записується в акумулятор **A**.

Додавання та віднімання 8-ми розрядних чисел без врахування попереднього перенесення продемонстровано на наступних прикладах.

Додавання: Сума доданків не перевищує 256:

1.

	Десяткова	двійкова	шістнадцяткова
1-й доданок:	12	00001100	0C
2-й доданок:	41	00101001	29
Сума:	53	00110101	35
Результат в регістрі **A**	53	00110101	35
Перенесення (запозичення) C	0	0	0

2.

	Десяткова	двійкова	шістнадцяткова
1-й доданок:	55	00110111	37
2-й доданок:	67	01000011	43
Сума:	122	01111010	7A
Результат в регістрі **A**	122	01111010	7A
Перенесення (запозичення) C	0	0	0

Додавання: Сума доданків перевищує 256:

	Десяткова	двійкова	шістнадцяткова
1-й доданок:	156	10011100	9C
2-й доданок:	189	10111101	BD
Сума:	345	101011001	59
Результат в регістрі **A**	89	01011001	59
Перенесення (запозичення) C	1	1	1

В результаті виконання такого додавання буде встановлено прапорець перенесення C (C = 1). В регістр **A** буде записано число, яке менше суми на 256d.

Віднімання: Зменшуване більше від'ємника.

	Десяткова	двійкова	шістнадцяткова
Зменшуване:	156	10011100	9C
Від'ємник:	41	00101001	29
Різниця:	115	01110011	73
Результат в регістрі **A**	115	01110011	73
Перенесення (запозичення) C	0	0	0

Віднімання: Зменшуване менше від'ємника.

	Десяткова	двійкова	шістнадцяткова
Зменшуване:	12	00001100	0C
Від'ємник:	41	00101001	29
Різниця:	-29	11100011	E3
Результат в регістрі **A**	227	11100011	E3
Перенесення (запозичення) C	1	1	1

В результаті виконання такого віднімання буде встановлено прапорець перенесення C (C = 1). Це еквівалентно запозиченню одиниці молодшого розряду старшого байта (якщо такий є). Слід зауважити, що в регістр **A** буде записане число 11100011b, що дорівнює числу 29d, записаному в доповненому коді.

8-ми розрядні додавання та віднімання без врахування попереднього перенесення реалізуються за допомогою таких команд, як:

1	Додавання до акумулятора	
2	безпосередня	
3	**ADI, B_2**	
4	**A + B_2 → A**	
5	C6h	306q
6	Команда займає 2 байта пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, які містяться в другому байті програми, додаються до значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

1	Віднімання від акумулятора	
2	безпосередня	
3	**SUI, B_2**	
4	**A - B_2 → A**	
5	D6h	326q
6	Команда займає 2 байта пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, які містяться в другому байті програми, віднімаються від значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

1	Додавання до акумулятора	
2	регістрова	
3	**ADD r**	
4	**A + r → A**	
5	80h - 85h, 87h	20Sq (S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди дані, які містяться в регістрі за номером S, додаються до значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

Номер S береться з Табл.1. Лабораторна робота 2.

1	Віднімання від акумулятора	
2	регістрова	
3	**SUB r**	
4	**A - r → A**	
5	90h - 95h, 97h	22Sq (S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди дані, які містяться в регістрі за номером S, віднімаються від значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

Номер S береться з Табл.1. Лабораторна робота 2.

1	Додавання до акумулятора	
2	непряма	
3	**ADD M**	
4	**A + M(HL) → A**	
5	86h	206q
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, що містяться в комірці пам'яті, адреса якої знаходиться в регістровій парі **HL**, додаються до значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

Перед виконанням команди командою **ADD M** необхідно завантажити пару **HL** адресою комірки, де міститься другий доданок.

1	Віднімання від акумулятора	
2	непряма	
3	**SUB M**	
4	**A - M(HL) → A**	
5	86h	206q
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, що містяться в комірці пам'яті, адреса якої знаходиться в регістровій парі **HL**, віднімаються від значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

Перед виконанням команди командою **SUB M** необхідно завантажити пару **HL** адресою комірки, де міститься від'ємник.

Можна замітити, що найшвидше виконуються команди додавання та віднімання з використанням одного із регістрів загального призначення, оскільки безпосередня та непряма адресації вимагають додаткового звернення до пам'яті.

3.1.2. Додавання та віднімання з урахуванням попереднього перенесення.

Команди 8-розрядного додавання та віднімання можна використовувати в випадку чисел, довжина яких складає більш ніж 8 біт. Такі числа використовують для двійкового представлення чисел, значно більших 256d. Наприклад:

Кількість байтів (бітів)	Максимальне десяткове число
1	2^8-1, 255
2	2^{16}-1, 65535
3	2^{24}-1, 16777215
4	2^{32}-1, 4294967295
і т.д.	

Додавати такі числа можна побайтно, однак слід враховувати перенесення з старшого розряду молодшого байту суми в молодший розряд наступного за старшинством байту. В випадку віднімання необхідно враховувати запозичення з молодшого розряду старшого байту. Розглянемо такий приклад:

Додати числа 30315d та 13802d. Записавши числа в двійковій системі числення та виконуючи побітне додавання можна побачити, що під час додавання молодших байтів виникає перенесення в молодший розряд суми старших байтів.

	старший байт								перенесення	молодший байт							
1-й доданок	0	1	1	1	0	1	1	0		0	1	1	0	1	0	1	1
2-й доданок	0	0	1	1	0	1	0	1		1	1	1	0	1	0	1	0
сума без перенесення	1	0	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1
перенесення								1	←								
сума з перенесенням	1	0	1	0	1	1	0	0		0	1	0	1	0	1	0	1

Якщо здійснювати додавання звичайним способом, то необхідно було б спочатку додати два старших байти, а потім до результату додати значення перенесення, оскільки команда додавання оперує тільки двома доданками. Однак для полегшення роботи програміста існує тип команд ДОДАВАННЯ З ПЕРЕНЕСЕННЯМ за допомогою яких можна додавати одразу три числа: два доданки та перенесення, отримане в попередній операції додавання. Важливо зауважити, що додаткове додавання перенесення здійснюється апаратним способом і не впливає на кількість тактів команд.

1	Додавання до акумулятора з урахуванням попереднього перенесення	
2	безпосередня	
3	**ACI, B_2**	
4	**A + B_2 + C→ A**	
5	CEh	316q
6	Команда займає 2 байта пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, які містяться в другому байті програми, та біт перенесення C додаються до значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

1	Додавання до акумулятора з урахуванням попереднього перенесення	
2	регістрова	
3	**ADC r**	
4	**A + r + C→ A**	
5	88h - 8Dh, 8Fh	21Sq (S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди дані, які містяться в регістрі за номером S, та біт перенесення C додаються до значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

Номер S береться з Табл.1. Лабораторна робота 2.

1	Додавання до акумулятора з урахуванням попереднього перенесення	
2	непряма	
3	**ADC M**	
4	**A + M(HL) + C→ A**	
5	8Eh	216q
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, що містяться в комірці пам'яті, адреса якої знаходиться в регістровій парі **HL**, та біт перенесення C додаються до значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

Перед виконанням команди командою **ADD M** необхідно завантажити пару **HL** адресою комірки, де міститься другий доданок.

Аналогічно можна розглянути віднімання двох чисел з запозиченням. Нехай це будуть ті ж числа 30315d та 13802d. Під час віднімання молодших байтів виникає необхідність запозичення одиниці молодшого розряду старшого байту:

	старший байт								запозичення	молодший байт							
зменшуване	0	1	1	1	0	1	1	0		0	1	1	0	1	0	1	1
від'ємник	0	0	1	1	0	1	0	1		1	1	1	0	1	0	1	0
різниця без запозичення	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1
запозичення								1	→								
сума з запозичення	0	1	0	0	0	0	0	0		1	0	0	0	0	0	0	1

Віднімання звичайним способом також вимагає здійснення його в два етапи: спочатку знайти різницю двох байтів, а потім від неї відняти значення запозичення. Принципової різниці між перенесенням та запозиченням немає, оскільки операцію віднімання мікропроцесор реалізує шляхом додавання до зменшуваного числа від'ємника в доповненому коді. Тому для полегшення роботи програміста існують команди ВІДНІМАННЯ З ПЕРЕНЕСЕННЯМ. Як і у випадку ДОДАВАННЯ З ПЕРЕНЕСЕННЯМ, додаткове віднімання перенесення не впливає на час виконання команд (кількість тактів).

1	Віднімання від акумулятора з урахуванням попереднього перенесення
2	безпосередня
3	**SBI, B_2**

4	**A - B_2 - C→ A**	
5	DEh	336q
6	Команда займає 2 байта пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, які містяться в другому байті програми, та біт перенесення С віднімаються від значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

1	Віднімання від акумулятора з урахуванням попереднього перенесення	
2	регістрова	
3	**SBB r**	
4	**A - r - C→ A**	
5	98h - 9Dh, 9Fh	23Sq (S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди дані, які містяться в регістрі за номером S, та біт перенесення С віднімаються від значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

Номер S береться з Табл.1. Лабораторна робота 2.

1	Віднімання від акумулятора з урахуванням попереднього перенесення	
2	непряма	
3	**SBB M**	
4	**A - M(HL) - C→ A**	
5	9Eh	236q
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди дані, що містяться в комірці пам'яті, адреса якої знаходиться в регістровій парі **HL**, та біт перенесення С віднімаються від значення акумулятора **A**. Результат записується в акумулятор **A**.	
8	Всі прапорці стану встановлюються в відповідності з результатом операції.	

Перед виконанням команди командою **SUB M** необхідно завантажити пару **HL** адресою комірки, де міститься від'ємник.

3.1.3. 16-ти розрядні додавання.

Мікропроцесор КР580ВМ80 має спеціальні команди для додавання 16-ти розрядних чисел. Використання цих команд дозволяє спростити написання програм, а також зменшити час їх виконання. В деяких випадках 16-ти розрядні числа можна розглядати як адреси комірок пам'яті, а операції додавання над цими адресами як обчислення зміщених відносно базових адрес комірок

підпрограм та даних. Наприклад, деяка підпрограма займає в пам'яті 50 байтів коду та 20 байт зарезервованого місця для зберігання проміжних результатів обчислень. Якщо ця підпрограма буде використовувати фіксовані адреси комірок пам'яті, то може виникнути ситуація, коли її використання призведе до затирання певних даних конкретної програми. Тому в програмі використовується базова адреса, за якою в пам'яті розміщується перша комірка підпрограми, а адреси інших команд та комірок даних підпрограми обчислюються за зміщенням відносно базової адреси.

Додавання 16-розрядних чисел здійснюється аналогічно, як і 8-ми розрядних, однак прапорець перенесення встановлюється в випадку, якщо в результаті додавання виникає перенесення з старшого розряду старшого байта суми. ДОДАВАННЯ З ПЕРЕНЕСЕННЯМ 16-ти розрядних чисел мікропроцесор КР580ВМ80 не підтримує, тому використовуючи команди 16-ти розрядного додавання до більш, ніж двобайтних чисел, слід враховувати перенесення (запозичення) програмним способом.

16-ти розрядні додавання здійснюються за допомогою регістра **HL**. Це означає, що один із доданків (операндів) повинен міститися в регістрі **HL**, туди ж записується результат операції. Другий з доданків може міститися в одному із 16-ти розрядних регістрів **BC**, **DE**, **HL** або **SP**, тобто адресація другого операнда завжди регістрова.

16-ти розрядних команд додавання всього чотири **DAD B**, **DAD D**, **DAD H** та **DAD SP**.

1	16-ти розрядне додавання	
2	регістрова/регістрова	
3	**DAD B (DAD D; DAD H; DAD SP)**	
4	**HL + BC → HL** **(HL + DE → HL; HL + HL → HL; HL + SP → HL)**	
5	09h (19h, 29h, 39h)	011q (031q, 051q, 071q)
6	Команда займає 1 байт пам'яті, виконується за 3цикли, 10 тактів.	
7	Під час виконання команди дані, які містяться в регістровій парі **BC (DE**, **HL, SP)** додаються до значення регістрової пари **HL**. Результат записується в регістрову пару **HL**.	
8	Результат виконання команди впливає тільки на прапорець перенесення.	

3.2. Додавання в двійково-десятковому коді. Десяткова корекція.

Наявність команди десяткової корекції дозволяє мікропроцесору виконувати додавати числа, записані в двійково-десятковому коді, як звичайні двійкові числа. Чому необхідний двійково-десятковий код? Відповідь на це питання пов'язане з областю застосування мікропроцесорів. Як правило їх використовують в простих системах необчислювального характеру, для яких характерне введення та виведення інформації в двійково-десятковому коді. Це можуть бути різного виду вимірювальні прилади з виведенням результатів на табло, створене на основі семисегментних знакоіндикаторів.

Сутність двійково-десяткового коду полягає в заміні кожної десяткової цифри її чотирибітним двійковим представленням:

0	0000	4	0100	8	1000
1	0001	5	0101	9	1001
2	0010	6	0110		
3	0011	7	0111		

Тоді число 3709d можна записати як 0011011100001001bd

Перетворення двійково-десяткового числа в десятковий здійснюється шляхом розбиття його на четвірки бітів зліва направо і заміною кожної четвірки її десятковим представленням. Таким чином за допомогою 1 байту можна представити всі дворозрядні десяткові числа від 00d (00000000bd) до 99d (10011001bd).

Необхідність корекції результатів додавання в двійково-десятковому коді виникає внаслідок того, що мікропроцесор інтерпретує двійково-десяткові числа як звичайні двійкові числа, а тетрадам 1010, 1011, 1100, 1101, 1110 та 1111 не відповідає жодна десяткова цифра.

Розглянемо можливі варіанти додавання двох однорозрядних десяткових чисел X та Y в двійково-десятковому коді.

$X + Y \leq 9$ - результат правильний, корекція непотрібна, перенесення в наступну тетраду відсутнє. Наприклад 0001 + 0011 = 0100.

$9 < X + Y \leq 15$ - результат невірний оскільки не може бути інтерпретованим, потрібна корекція, перенесення в наступну тетраду відсутнє. Наприклад 0110 + 0111 = 1101.

$15 < X + Y \leq 18$ - результат невірний, число можна інтерпретувати як десяткове, але його значення відрізняється від дійсного, потрібна корекція, є перенесення в наступну тетраду. Наприклад 1000 + 1001 = 1 0001

Корекція здійснюється шляхом додавання числа 6 (0110) у випадку, якщо перенесення відсутнє, а результат додавання перевищує 9, або у випадку, якщо було перенесення в наступну тетраду. Оскільки одним байтом можна кодувати дворозрядне десяткове число, необхідно контролювати перенесення з молодшої тетради в старшу. Для цього існує прапорець додаткового перенесення AC, який встановлюється в 1 в випадку перенесення з третього розряду в четвертий.

Наприклад необхідно додати числа 1889d та 6376d. Кожне з чисел може бути представлене за допомогою 2 байтів: 00011000bd, 10001001bd та 01100011bd, 01110110bd.

Доданок 1. Молодший байт				1 0 0 0		1 0 0 1
Доданок 2. Молодший байт				0 1 1 1		0 1 1 0
				--------		--------
Сума				1 1 1 1		1 1 1 1
Десяткова корекція				0 1 1 0		0 1 1 0
						--------
				1 ←		0 1 0 1
				---------		
			1 ←	0 1 1 0		
Доданок 1. Старший байт	0 0 0 1		1 0 0 0			
Доданок 1. Старший байт	0 1 1 0		0 0 1 1			
	--------		--------			
Сума	0 1 1 1		1 1 0 0			
Десяткова корекція	0 0 0 0		0 1 1 0			
			--------			
	1 ←		0 0 1 0			
	--------					
	1 0 0 0					
Результат додавання:	1 0 0 0		0 0 1 0		0 1 1 0		0 1 0 1

Отриманий в результаті додавання результат переводимо в десяткову систему: 8265d. Перевіримо:

$$\begin{array}{r} 1889 \\ \underline{6376} \\ 8265 \end{array}$$

Десяткова корекція здійснюється за допомогою команди **DAA**. Однак вона застосовна тільки до операції 8-ми розрядного додавання. В випадку 16-ти розрядних додавань, та 8-ми розрядних віднімань її реалізують програмними засобами.

1	Десяткова корекція акумулятора **A**	
2	регістрова	
3	**DAA**	
4	(Десяткова корекція **A) → A**	
5	27h	047q
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди 8-ми розрядне слово, яке містяться в акумуляторі А, перетворюється в два чотирибітних двійково-десяткові числа наступним чином: 1) Якщо величина молодших чотирьох бітів акумулятора більше 1001, або якщо встановлено прапорець AC=1, то до значення цих бітів додається число 0110. 2) Якщо тепер величина старших чотирьох бітів акумулятора більше 1001, або якщо встановлено прапорець C=1, то до значення цих бітів	

	додається число 0110.
8	Всі прапорці стану встановлюються в відповідності з результатом операції.

УВАГА! Команда **DAA** на жаль не реалізована в емуляторі мікропроцесора КР580ВМ80, а спроба її виконання призводить до "зависання" роботи емулятора.

3.3. Команди приросту (інкременту-декременту).

Команди додатного (increment) та від'ємного (decrement) приросту є спеціальними арифметичними командами. Виконання команд додатного приросту приводить до збільшення значення відповідного регістру чи регістрової пари на 1. Аналогічно команди від'ємного приросту зменшують значення регістру чи пари на 1. Дані команди використовуються в ситуаціях, коли необхідно підрахувати кількість деяких подій, які виконуються з повтором.

Команда декременту часто використовується для встановлення того факту, що певна частина програми була виконана задану кількість разів. Можна, звичайно, використати команду інкременту, однак після виконання тіла циклу необхідно перевіряти, чи відповідає число здійснених повторів заданій величині. Така перевірка здійснюється за допомогою команди порівняння, і займає додаткову пам'ять програми. Використовуючи команду декременту можна реалізувати цикл простіше, оскільки кількість повторів задається в деякому регістрі, а після виконання кожного повтору зменшується на одиницю. Далі програма перевіряє цей регістр на нульовий результат відповідно до прапорця нуля. Якщо всі повтори виконано, в регістрі - нульове значення, прапорець нуля встановлено в 1, то програма виходить із циклу та виконує наступну за циклом команду.

Команди інкременту, як правило, використовуються для організації масивів даних. В такому випадку задається початкова адреса першої комірки пам'яті, яка записується в регістрову пару **HL.** Звертання до комірок пам'яті здійснюється, таким чином, за допомогою непрямої адресації. Застосовуючи команду інкременту до регістрової пари **HL,** можна адресуватися до наступного байту масиву.

1	Інкремент (додатній приріст) регістру	
2	регістрова	
3	**INR r**	
4	**r + 1 → r**	
5	04h, 0Ch, 14h, 1Ch, 24h, 2Ch, 3Ch	0D4q (D ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 5 тактів.	
7	Під час виконання команди вміст регістру за номером D збільшується на 1.	
8	Всі прапорці стану за винятком прапорця перенесення встановлюються в відповідності з результатом операції.	

Номер D береться з Табл.1. Лабораторна робота 2.

1	Декремент (від'ємний приріст) регістру	
2	регістрова	
3	**DCR r**	
4	**r - 1 → r**	
5	05h, 0Dh, 15h, 1Dh, 25h, 2Dh, 3Dh	0D5q (D ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 5 тактів.	
7	Під час виконання команди вміст регістру за номером D зменшується на 1.	
8	Всі прапорці стану за винятком прапорця перенесення встановлюються в відповідності з результатом операції.	

Номер D береться з Табл.1. Лабораторна робота 2.

1	Інкремент (додатній приріст) комірки памяті	
2	непряма	
3	**INR M**	
4	**M(H, L) + 1 → M(H, L)**	
5	34h	064q
6	Команда займає 1 байт пам'яті, виконується за 3 цикли, 10 тактів.	
7	Під час виконання команди вміст комірки пам'яті, адреса якої знаходиться в регістровій парі **HL**, збільшується на 1.	
8	Всі прапорці стану за винятком прапорця перенесення встановлюються в відповідності з результатом операції.	

Перед інкрементом комірки пам'яті за допомогою команди **INR M** необхідно завантажити пару **HL** адресою комірки.

1	Декремент (від'ємний приріст) комірки памяті	
2	непряма	
3	**DCR M**	
4	**M(H, L) - 1 → M(H, L)**	
5	35h	065q
6	Команда займає 1 байт пам'яті, виконується за 3 цикли, 10 тактів.	
7	Під час виконання команди вміст комірки пам'яті, адреса якої знаходиться в регістровій парі **HL**, зменшується на 1.	
8	Всі прапорці стану за винятком прапорця перенесення встановлюються в відповідності з результатом операції.	

Перед декрементом комірки пам'яті за допомогою команди **DCR M** необхідно завантажити пару **HL** адресою комірки.

1	Інкремент (додатній приріст) регістрової пари
2	регістрова

3	**INX B (INX D; INX H; INX SP)**	
4	**BC + 1 → BC** **(DE + 1 → DE; HL + 1 → HL; SP + 1 → SP)**	
5	03h (13h, 23h, 33h)	003q (023q, 043q, 063q)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 5 тактів.	
7	Під час виконання команди вміст регістрової пари **BC (DE**, **HL, SP)** збільшується на 1	
8	Виконання команди не впливає на регістр стану.	

1	Декремент (від'ємний приріст) регістрової пари	
2	регістрова	
3	**DCX B (DCX D; DCX H; DCX SP)**	
4	**BC - 1 → BC** **(DE - 1 → DE; HL - 1 → HL; SP - 1 → SP)**	
5	0Bh (1Bh, 2Bh, 3Bh)	013q (033q, 053q, 073q)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 5 тактів.	
7	Під час виконання команди вміст регістрової пари **BC (DE**, **HL, SP)** зменшується на 1	
8	Виконання команди не впливає на регістр стану.	

Розділ 4. ЛОГІЧНІ КОМАНДИ

Слово "Комп'ютер" в перекладі означає "Обчислювач". Однак більшість операцій над числами, які він виконує, відносяться до логічних, а не до математичних. Тому крім широкого набору математичних команд, за допомогою яких здійснюються математичні операції додавання та віднімання, в мікропроцесорі КР580ВМ80 також передбачено і ряд логічних команд. До них відносяться команди "І", "АБО", "ВИКЛЮЧНЕ АБО" та "ЗАПЕРЕЧЕННЯ" ("ІНВЕРСІЯ"), а також команда "ПОРІВНЯННЯ", за допомогою якої здійснюються різноманітні перевірки. Окремо слід виділити команди простого та циклічного зсуву, які використовуються для реалізації операцій "МНОЖЕННЯ" і "ДІЛЕННЯ" та деяких інших цілей.

4.1. Особливості виконання логічних команд.

Оскільки логічні функції завжди виконуються над однобітними числами, які можуть набувати значення 0 та 1 (хиба та істина), тому логічні команди оперують не безпосередньо з числами, записаними в регістри чи пам'ять, а з окремими бітами цих чисел. Всі логічні команди мікропроцесора КР580ВМ80 оперують з 8-ми розрядними числами побітно. Це означає, що кожна з логічних команд насправді реалізує 8 окремих незалежних між собою логічних функцій над однаковими розрядами операндів. До різних розрядів операндів ніякі логічні команди не застосовуються. Один із операндів логічних команд завжди міститься в акумуляторі **А**. Другий із операндів може міститися в другому байті команди (безпосередня адресація) або в одному із регістрів загального призначення (регістрова адресація) або в комірці пам'яті (непряма адресація). Адреса комірки пам'яті повинна бути записана в регістровій парі **HL**. Команда інверсії оперує тільки з вмістом акумулятора. Результат виконання команд завжди записується в акумулятор, тому значення першого операнда буде втрачене, якщо не передбачити попереднього його збереження. Результат виконання логічних команд впливає тільки на прапорці нуля, знаку та парності і не впливають на прапорці простого та додаткового перенесення, оскільки перенесення між розрядами відноситься тільки до арифметичних команд.

4.2. Команди порозрядного "І".

Виконання команди "**І**" продемонстровано на прикладах 1 і 2.

Приклад 1.

	двійкова	шістнадцяткова
1-й операнд X (акумулятор):	10111010	BA
2-й операнд Y (регістр або пам'ять):	11110000	F0
X & Y	10110000	B0
Результат в регістрі **А**	10110000	B0

В даному випадку команда "**I**" реалізує такі функції:

X_7 & Y_7 = 1; X_6 & Y_6 = 0; X_5 & Y_5 = 1; X_4 & Y_4 = 1;
X_3 & Y_3 = 0; X_2 & Y_2 = 0; X_1 & Y_1 = 0; X_0 & Y_0 = 0.

Цей приклад демонструє одне із типових застосувань команди "**I**", яке називається **порозрядним маскуванням**. Сутність його полягає в виділенні окремих бітів багаторозрядного числа для їхнього наступного аналізу. Це позволяє використовувати результат такого виділення для виконання певних арифметичних чи логічних команд, а також для організації умовних переходів. В даному прикладі із числа X відповідно до маски - числа Y виділено значення перших чотирьох бітів.

В якості маски може бути використаний будь-який набір бітів. На прикладі 2 показано виділення наймолодшого нульового та найстаршого сьомого біта числа. Якщо необхідно здійснити перехід на підпрограму додаткової обробки даних в випадку рівності цих бітів числа нулю, то достатньо перевірити результат X & Y на нуль.

Найчастіше маскування здійснюється за допомогою команди **ANI, B_2**, де другий байт команди **B_2** - маска.

Приклад 2.

	двійкова	шістнадцяткова
1-й операнд X (акумулятор):	10010100	94
2-й операнд Y (регістр або пам'ять):	10000001	81
X & Y	10000000	80
Результат в регістрі **A**	10000000	80

Операція порозрядного "I" реалізована за допомогою таких 9-ти команд:

1	Порозрядне "**I**" числа та акумулятора	
2	безпосередня	
3	**ANI, B_2**	
4	**A & B_2 → A**	
5	E6h	346q
6	Команда займає 2 байти пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди над кожним із розрядів двійкових чисел в акумуляторі **A** та другому байті команди здійснюється логічне "**I**". Результат записується у відповідний розряд акумулятора **A**.	
8	Прапорці стану C та AC встановлюються в нуль, решта у відповідності з результатом операції.	

1	Порозрядне "**І**" регістра та акумулятора	
2	регістрова	
3	**ANA r**	
4	**A & r → A**	
5	A0h-A5h, A7h	24Sq (S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди над кожним із розрядів двійкових чисел в акумуляторі **A** та регістрі за номером S здійснюється логічне "**І**". Результат записується у відповідний розряд акумулятора **A**.	
8	Прапорці стану C та AC встановлюються в нуль, решта у відповідності з результатом операції.	

Номер S береться з Табл.1. Лабораторна робота 2.

1	Порозрядне "**І**" комірки пам'яті та акумулятора	
2	непряма	
3	**ANA M**	
4	**A & M(H, L) → A**	
5	A6h	246q
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди над кожним із розрядів двійкових чисел в акумуляторі **A** та комірці пам'яті, адреса якої міститься в регістровій парі **HL**, здійснюється логічне "**І**". Результат записується у відповідний розряд акумулятора **A**.	
8	Прапорці стану C та AC встановлюються в нуль, решта у відповідності з результатом операції.	

Перед виконанням команди **ANA, M** необхідно завантажити пару **HL** адресою комірки.

4.3. Команди порозрядного "АБО".

Виконання команди "**АБО**" продемонстровано на прикладі 3.

Біт певного розряду результату дорівнює 1, якщо таке значення має хоча б один із операндів у відповідному розряді. Я видно з прикладу 3, команда порозрядного "**АБО**" також може використовуватись для порозрядного маскування. Якщо біт маски рівний нулю, то відповідний біт акумулятора залишається без змін. В випадку, якщо біти маски рівні 1, значення відповідних бітів початкового значення акумулятора блокуються, тобто приймають одиничні значення. В залежності від результату, який необхідно досягнути, використовують маскування по "**І**" чи маскування по "**АБО**".

Приклад 3.

	двійкова	шістнадцяткова
1-й операнд X (акумулятор):	10111010	BA
2-й операнд Y (регістр або пам'ять):	00001111	0F
X ∨ Y	10111111	BF
Результат в регістрі **A**	10111111	BF

Операція порозрядного "І" реалізована за допомогою таких 9-ти команд:

1	Порозрядне "**АБО**" числа та акумулятора	
2	безпосередня	
3	**ORI, B_2**	
4	**A ∨ B_2 → A**	
5	F6h	366q
6	Команда займає 2 байти пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди над кожним із розрядів двійкових чисел в акумуляторі **A** та другому байті команди здійснюється логічне "**АБО**". Результат записується у відповідний розряд акумулятора **A**.	
8	Прапорці стану C та AC встановлюються в нуль, решта у відповідності з результатом операції.	

1	Порозрядне "**АБО**" регістра та акумулятора	
2	регістрова	
3	**ORA r**	
4	**A ∨ r → A**	
5	B0h-B5h, B7h	26Sq (S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди над кожним із розрядів двійкових чисел в акумуляторі **A** та регістрі за номером S здійснюється логічне "**АБО**". Результат записується у відповідний розряд акумулятора **A**.	
8	Прапорці стану C та AC встановлюються в нуль, решта у відповідності з результатом операції.	

Номер S береться з Табл.1. Лабораторна робота 2.

1	Порозрядне "**АБО**" комірки пам'яті та акумулятора	
2	непряма	
3	**ORA M**	
4	**A ∨ M(H, L) → A**	
5	B6h	266q
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	

7	Під час виконання команди над кожним із розрядів двійкових чисел в акумуляторі **A** та комірці пам'яті, адреса якої міститься в регістровій парі **HL**, здійснюється логічне "**АБО**". Результат записується у відповідний розряд акумулятора **A**.
8	Прапорці стану C та AC встановлюються в нуль, решта у відповідності з результатом операції.

Перед виконанням команди **ORA, M** необхідно завантажити пару **HL** адресою комірки.

4.4. Команди порозрядного "Виключне АБО".

Функція "**Виключне АБО**" задається у відповідності з таблицею істинності:

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Іншими словами функція "**Виключне АБО**" реалізує операцію додавання по модулю однорозрядних чисел A та B (1+1=2; 2 mod 2 = 0). Команда "**Виключне АБО**" здійснює порозрядне виконання функції "**Виключне АБО**" над відповідними розрядами операндів. Виконання команди "**Виключне АБО**" продемонстровано на прикладах 4 та 5.

Приклад 4.

	двійкова	шістнадцяткова
1-й операнд X (акумулятор):	10010110	96
2-й операнд Y (регістр або пам'ять):	10101111	AF
X ⊕ Y	00111001	39
Результат в регістрі **A**	00111001	39

Приклад 5.

	двійкова	шістнадцяткова
1-й операнд X (акумулятор):	10010110	96
2-й операнд Y (регістр або пам'ять):	10010110	96
X ⊕ Y	00000000	00
Результат в регістрі **A**	00000000	00

Наведені приклади ілюструють один із широко використовуваних способів застосування команди "**Виключне АБО**" - використання її для здійснення перевірок на співпадіння окремих бітів операндів. Якщо відповідні розряди акумулятора та маски співпадають, то в такому ж розряді результату встановлюється значення 0, в випадку неспівпадіння встановлюється 1. Як частковий випадок можна використовувати команду "**Виключне АБО**" для перевірки співпадіння двох чисел (приклад 5). Нульове значення результату буде свідчити про таке співпадіння.

Ще один спосіб використання команди "**Виключне АБО**" - для обнулення акумулятора **A** (приклад 5). Мікропроцесор КР580ВМ86 не має команд очищення чи скидання акумулятора. Для цього достатньо виконати команду **XRA A,** тобто здійснити порозрядне "**Виключне АБО**" над акумулятором, та другим операндом, яким також є акумулятор. Оскільки ці числа однакові, в акумулятор буде записано число 0 в усі розряди. Звісно, можна завантажити акумулятор числом 00h, за допомогою команди **MVI A,** 00h, однак ця команда займає в пам'яті 2 байти, виконується за 2 цикли та 7 тактів в той час як команда **XRA A** займає 1 байт та виконується за 1 цикл та 4 такти.

Операція порозрядного "**Виключне АБО**" реалізована за допомогою таких 9-ти команд:

1	Порозрядне "**Виключне АБО**" числа та акумулятора	
2	безпосередня	
3	**XRI, B_2**	
4	**A $\oplus$ B_2 → A**	
5	EEh	356q
6	Команда займає 2 байти пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди над кожним із розрядів двійкових чисел в акумуляторі **A** та другому байті команди здійснюється логічне "**Виключне АБО**". Результат записується у відповідний розряд акумулятора **A**.	
8	Прапорці стану C та AC встановлюються в нуль, решта у відповідності з результатом операції.	

1	Порозрядне "**Виключне АБО**" регістра та акумулятора	
2	регістрова	
3	**XRA r**	
4	**A $\oplus$ r → A**	
5	A8h-ADh, AFh	25Sq (S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди над кожним із розрядів двійкових чисел в акумуляторі **A** та регістрі за номером S здійснюється логічне "**Виключне АБО**". Результат записується у відповідний розряд акумулятора **A**.	

8	Прапорці стану C та AC встановлюються в нуль, решта у відповідності з результатом операції.

Номер S береться з Табл.1. Лабораторна робота 2.

1	Порозрядне "**Виключне АБО**" комірки пам'яті та акумулятора	
2	непряма	
3	**XRA M**	
4	**A ⊕ M(H, L) → A**	
5	AEh	256q
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди над кожним із розрядів двійкових чисел в акумуляторі **A** та комірці пам'яті, адреса якої міститься в регістровій парі **HL**, здійснюється логічне "**Виключне АБО**". Результат записується у відповідний розряд акумулятора **A**.	
8	Прапорці стану C та AC встановлюються в нуль, решта у відповідності з результатом операції.	

Перед виконанням команди **XRA, M** необхідно завантажити пару **HL** адресою комірки.

4.5. Команда порозрядного "ЗАПЕРЕЧЕННЯ".

Приклад 6. демонструє виконання команди порозрядного "**ЗАПЕРЕЧЕННЯ**":

Приклад 6.

	двійкова	шістнадцяткова
операнд X (акумулятор):	10010100	94
$\overline{X}$	01101011	6B
Результат в регістрі **A**	01101011	6B

Одним із застосувань команди "**ЗАПЕРЕЧЕННЯ**" (інверсії) є використання її для отримання оберненого, а також доповненого коду числа, що є необхідним для програмної реалізації віднімання багаторозрядних чисел.

Команда "**ЗАПЕРЕЧЕННЯ**" може використовуватись в комбінації з командами "**І**", "**АБО**" та "**Виключне АБО**". В такому випадку будуть реалізовані команди "**І-НЕ**", "**АБО-НЕ**" та "**Виключне АБО-НЕ**".

Команда порозрядного "**ЗАПЕРЕЧЕННЯ**" всього одна:

1	Порозрядна інверсія акумулятора
2	регістрова
3	**CMA**

4	$\overline{A} \rightarrow$ **A**	
5	2Fh	057q
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди кожен із розрядів двійкового числа в акумуляторі **A** змінює своє значення на протилежне. Результат записується у відповідний розряд акумулятора **A**.	
8	Виконання команди не впливає на регістр стану.	

4.6. Команди порозрядного "Порівняння".

Часто виникає необхідність порівняння двох чисел на предмет їх співпадіння, або встановити, яке з них більше чи менше. З попередніх прикладів видно, що співпадіння можна перевірити за допомогою команди "**Виключне АБО**". Такий спосіб перевірки має той недолік, що в результаті порівняння буде втрачене значення акумулятора, оскільки один із операндів повинен бути записаний в акумулятор і туди ж записується результат виконання команди. Якщо цей операнд потрібно використовувати далі в програмі, то його необхідно запам'ятати перед виконанням команди "**Виключне АБО**". А це приводить до додаткових команд програми.

Співпадіння можна перевірити також за допомогою команди "**Віднімання**". В такому випадку від числа, яке перевіряють і яке записане в акумуляторі, віднімають контрольне число. Якщо ці числа рівні, то в акумулятор буде записано 0 в усі розряди, а прапорець нуля встановлено в 1. Якщо контрольне число більше, то в 1 буде встановлено значення прапорця перенесення, оскільки відбудеться запозичення в старший розряд. Значення прапорців Z та C можна використати як ознаку "співпадіння", "менше" чи "більше". Таким чином команда "**Віднімання**" має більш широкі можливості в порівнянні з командою "**Виключне АБО**". Однак їй притаманний такий ж недолік, а саме втрата значення акумулятора **A**.

Від цього недоліку вільна команди "**Порівняння**". Виконання цих команд аналогічне виконанню команди "**Віднімання**", однак результат віднімання не завантажується в акумулятор. Дані команди впливають виключно на прапорці регістра стану. Команди "**Порівняння**" використовують звичайні способи адресації другого операнда: безпосередню, регістрова та непряму. Всього команд"**Порівняння**" є 9:

1	Порівняння числа та акумулятора	
2	безпосередня	
3	**CPI, B_2**	
4	**A - B_2 → регістр стану**	
5	FEh	376q
6	Команда займає 2 байти пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди від значення акумулятора **A** віднімається значення другого байту команди. Результат віднімання впливає тільки на значення прапорців регістра стану. Значення акумулятора	

	не змінюється.
8	Якщо **A** рівне $\mathbf{B_2}$, то Z=1. Якщо **A** < $\mathbf{B_2}$, то C=1, решта прапорців у відповідності з результатом операції віднімання.

1	Порівняння регістра та акумулятора	
2	регістрова	
3	**CMP r**	
4	**A - r → регістр стану**	
5	B8h-BDh, BFh	27Sq (S ≠ 6)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди від значення акумулятора **A** віднімається значення, що міститься в регістрі за номером S. Результат віднімання впливає тільки на значення прапорців регістра стану. Значення акумулятора та регістру не змінюються.	
8	Якщо **A** рівне **r**, то Z=1. Якщо **A** < **r**, то C=1, решта прапорців у відповідності з результатом операції віднімання.	

Номер S береться з Табл.1. Лабораторна робота 2.

1	Порівняння комірки пам'яті та акумулятора	
2	непряма	
3	**CMP M**	
4	**A - M(H, L) → регістр стану**	
5	BEh	276q
6	Команда займає 1 байт пам'яті, виконується за 2 цикли, 7 тактів.	
7	Під час виконання команди від значення акумулятора **A** віднімається значення комірки пам'яті, адреса якої міститься в регістровій парі **HL**. Результат віднімання впливає тільки на значення прапорців регістра стану. Значення акумулятора та комірки пам'яті не змінюються.	
8	Якщо **A** рівне **M(H, L)**, то Z=1. Якщо **A** < **M(H, L)**, то C=1, решта прапорців у відповідності з результатом операції віднімання.	

Перед виконанням команди **CMP, M** необхідно завантажити пару **HL** адресою комірки.

4.7. Команди зміни прапорця перенесення.

В деяких спеціальних випадках необхідно мати можливість явним чином (без урахування результату арифметичних чи логічних команд) змінити певні прапорці регістру стану. В мікропроцесорі КР580ВМ80 такий доступ забезпечено тільки до прапорця перенесення. Існують дві команди: встановлення прапорця в 1 та інверсія поточного його значення:

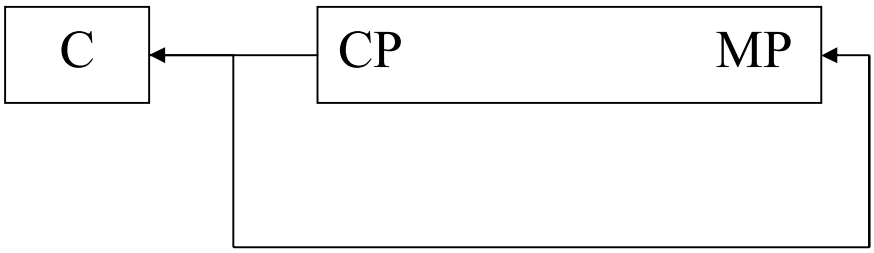
1	Встановлення прапорця перенесення	
2	регістрова	
3	**STC**	
4	**1** → C	
5	37h	067q
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди розряд регістра стану, який відповідає прапорцю перенесення встановлюється в 1.	
8	Виконання команди не змінює попереднього значення впливає на регістр стану.	

1	Інверсія прапорця перенесення	
2	регістрова	
3	**CMC**	
4	$\overline{C}$ → C	
5	3Fh	077q
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди розряд регістра стану, який відповідає прапорцю перенесення міняє своє значення на протилежне	
8	Виконання команди не змінює попереднього значення впливає на регістр стану.	

Сказане вище не означає, що не можна, наприклад програмним чином, змінити значення також і інших прапорців. Для цього необхідно завантажити з пам'яті в мікропроцесор слово стану PSW. Для цього використовуються команди роботи зі стеком, які розглядаються в наступних лабораторних роботах.

4.8. Команди циклічного зсуву.

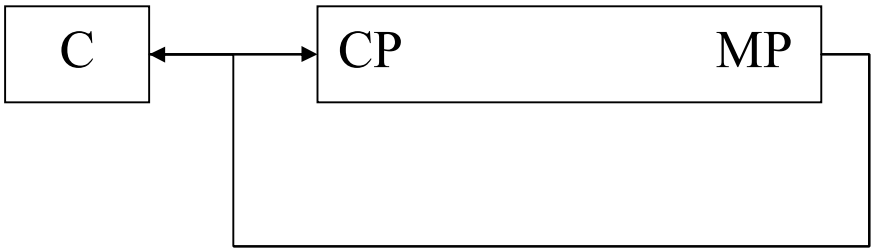
За допомогою команди циклічного зсуву вліво всі дані, що містяться в акумуляторі **A**, зсуваються на одну позицію вліво, тобто значення кожного розряду переміщується в позицію сусіднього зліва розряду. Значення старшого розряду переміщується в прапорець перенесення C та молодший розряд акумулятора:



Таке переміщення можна описати формулою: **A**×2+**A**/128 → **A**, $\mathbf{A_7}$ → **C** оскільки зсув вліво еквівалентний множенню на 2, а перенесення зі старшого в

молодший розряд еквівалентне діленню на 128 (переміщенню на 7 розрядів вправо) з відкиданням дробової частини.

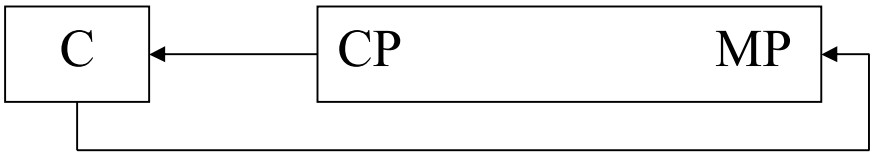
Подібним чином здійснюється команда циклічного зсуву вправо:



Таке переміщення описується формулою: **A**/2+**A**×128 → **A, A_0 → C**

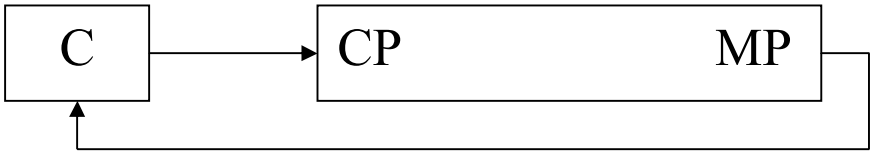
1	Циклічний зсув акумулятора вліво (вправо)	
2	регістрова	
3	**RLC (RRC)**	
4	2×**A** + **A**/128→ **A** (1/2×**A** + 128×**A**→ **A)**	
5	07h (0Fh)	007q (017q)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди розряди акумулятора **A** зсуваються вліво (вправо) на один розряд. Значення прапорця перенесення C і молодшого (старшого) розряду акумулятора замінюються значенням старшого (молодшого) розряду акумулятора	
8	Виконання команди змінює значення тільки прапорця перенесення.	

Команда циклічного зсуву вліво з урахуванням перенесення відрізняється від попередньої команди тим, що значення старшого розряду заноситься в прапорець перенесення C, а попереднє значення прапорця поміщається в молодший розряд акумулятора:



Таке переміщення описується, як **A**×2+C → **A, A_7 → C**

В випадку циклічного зсуву вправо аналогічно маємо:



та **A**/2+C×128 → **A, A_0 → C.**

1	Циклічний зсув акумулятора вліво (вправо) з урахуванням перенесення.
2	регістрова

3	**RAL (RAR)**	
4	2×**A** + C→ **A** (1/2×**A** + 128×C→ **A)**	
5	17h (1Fh)	027q (037q)
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 4 такти.	
7	Під час виконання команди розряди акумулятора **A** зсуваються вліво (вправо) на один розряд. Значення прапорця перенесення C заноситься в молодший (старший) розряд акумулятора. Значення старшого (молодшого) розряду акумулятора заноситься в прапорець перенесення C.	
8	Виконання команди змінює значення тільки прапорця перенесення.	

Команди циклічного зсуву використовуються для реалізації множення двійкових чисел. Наприклад, множення двох 8-ми розрядних чисел можна представити як послідовність виконання накопичення числа в 16 розрядному регістрі в результаті послідовного виконання операцій додавання та зсуву.

```
                0 1 1 0 1 0 1 0
                1 0 0 1 0 1 0 1
                ---------------
                0 1 1 0 1 0 1 0
              0 0 0 0 0 0 0 0
            0 1 1 0 1 0 1 0
          0 0 0 0 0 0 0 0
        0 1 1 0 1 0 1 0
      0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
  0 1 1 0 1 0 1 0
---------------------------------
  0 1 1 1 1 0 1 1 0 1 1 0 0 1 0
```

Для зсуву чисел, розрядність яких перевищує 8, спеціальних команд немає, одна їх можна реалізувати програмно. Для цього можна до кожного із байтів, які містять число, починаючи з молодшого застосувати команду циклічного зсуву вліво з перенесенням. В такому випадку старший біт молодшого байту буде занесено в прапорець перенесення, а наступною командою буде поміщено в молодший біт наступного байту. Для того, щоби в молодший біт молодшого байту не було занесено значення прапорця перенесення, можна попередньо встановити прапорець перенесення в 1 командою **STC**, а потім інвертувати його до значення 0 командою **CMC**. Інший спосіб полягає в маскуванні молодшого біту за допомогою команди "**I**", наприклад **ANI**, FEh.

Розділ. 5. КОМАНДИ УМОВНИХ ТА БЕЗУМОВНИХ ПЕРЕХОДІВ

Широкі можливості мікропроцесора в значній мірі визначаються його здатністю приймати рішення в ході виконання програми у відповідності з отриманим результатом. Для забезпечення таких можливостей використовуються команди умовних та безумовних переходів, які дають змогу міняти послідовність виконання програм. З їх допомогою в програмах влаштовують цикли та розгалуження, тому вони інколи ще називаються командами розгалуження.

5.1. Загальні властивості команд переходів

Як витікає з їх назви, всі розглядувані команди поділяються на дві групи: безумовних та умовних переходів. Якщо на виконання команд першої групи не впливають результати попередніх обчислень, то виконання команд другої групи здійснюється якраз саме у відповідності з цими результатами.

За винятком команди **PCHL**, яка є однобайтовою, всі інші є трибайтовими. Це означає, що другий та третій байт команди містять повну 16-розрядну адресу деякої комірки пам'яті. В цій комірці пам'яті записана команда програми, яку необхідно виконати наступною в випадку, якщо задовільняється умова переходу**. Перехід здійснюється шляхом запису адреси цієї комірки в лічильник команд PC та передачі керування команді, яка в ній записана.** Якщо ж умова не задовільняється, то ніякого переходу не відбувається, а мікропроцесор виконує команду, наступну після команди переходу, тобто лічильник команд збільшує своє значення на 3. Оскільки адреса комірки, де міститься код команди, безпосередньо завантажується в **PC**, то вважається, що використовується безпосередня адресація. Єдина команда переходу, що використовує регістрову адресацію є однобайтова команда **PCHL**. Необхідно звернути увагу на те, що в **другому байті команди міститься молодший байт адреси, а в третьому - старший.** Тому, наприклад, команда **JMP** A7, 00 означає перехід за адресою 00A7h.

Слід пам'ятати, що після виконання команди переходу не залишається жодних даних про те, з якої точки програми було здійснено цей перехід, тому всі умовні та безумовні переходи необхідно застосовувати якомога рідше, оскільки наявність великої кількості галужень програми сильно ускладнює її написання та відлаштування.

5.2. Команди безумовного переходу

Під час виконання цих команд ніякі перевірки не здійснюються. В лічильник команд мікропроцесора автоматично завантажується значення другого та третього байту команди, і починається виконання команди за цією адресою. Команди безумовного переходу відрізняються від усіх команд переходів тим, що їх можна розглядати також як команди завантаження 16-розрядного регістра **PC**.

Команди безумовного переходу використовують в тому випадку, якщо необхідно здійснити перехід на чітко визначену команду програми, а також для зв'язування різних частин програми. Інколи їх використовують під час написання та відлаштування програми для переходів з так званих "заготовок" та "заглушок".

За допомогою команд безумовного переходу реалізують оператори типу GOTO мов високого рівня. Оскільки велика кількість безумовних переходів негативно відображається на надійності та "читабельності" програм, слід обмежувати використання команд такого виду.

В мікропроцесорі КР580ВМ80 передбачено 2 команди безумовного переходу:

1	Безумовний перехід	
2	безпосередня	
3	**JMP B_2, B_3**	
4	**B_3 B_2 → PC**	
5	C3h	303q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Під час виконання команди в молодший байт лічильника команд **PC** записується другий байт команди **B_2**, а в старший байт **PC** - третій байт команди **B_3**. Керування передається команді, яка буде занесена в **PC** (за адресою **B_3 B_2).**	
8	Виконання команди не впливає на регістр стану.	

1	Завантаження лічильника команд **PC** (непрямий перехід)	
2	регістрова / регістрова	
3	**PCHL**	
4	**HL → PC**	
5	E9h	351q
6	Команда займає 1 байт пам'яті, виконується за 1 цикл, 5 тактів.	
7	Під час виконання команди дані, які містяться в регістрі **L** пересилаються в молодший байт лічильника команд **PC**, а дані, які містяться в регістрі **H** - в старший байт. Керування передається команді, яка буде занесена в **PC** (за адресою **B_3 B_2)**.	
8	Виконання команди не впливає на регістр стану.	

5.3. Команди умовних переходів

На виконання всіх команд умовних переходів впливає значення окремих прапорців регістру стану, однак використовуються не всі прапорці. Наприклад, прапорець додаткового перенесення не впливає на жодну із команд переходу. Кожен прапорець може бути встановлений в одне із двох значень, тому всього команд умовного переходу є 8.

Всі команди умовного переходу мають такий формат: якщо **умова виконується**, то **перейти на вказану адресу**, якщо **ні** - **виконувати наступну команду** програми.

Команди ПЕРЕХІД, ЯКЩО НУЛЬ та ПЕРЕХІД, ЯКЩО НЕ НУЛЬ перевіряють значення прапорця нуля. Якщо його значення рівне 1, то це свідчить, що в результаті попередньої дії було набуто нульовий результат. Тому в результаті виконання команди ПЕРЕХІД, ЯКЩО НУЛЬ буде здійснено перехід за адресою, яка міститься в другому та третьому байтах програми, а в випадку нульового значення прапорця такий перехід здійснено не буде. Якщо прапорець встановлено в 0, то навпаки, команда ПЕРЕХІД, ЯКЩО НЕ НУЛЬ буде виконуватись з переходом, а команда ПЕРЕХІД, ЯКЩО НУЛЬ - без.

Команду ПЕРЕХІД, ЯКЩО НУЛЬ часом називають командою ПЕРЕХІД, ЯКЩО РІВНЕ. Таку назву команда отримала тому, що її часто застосовують одразу ж після команди ПОРІВНЯННЯ. Якщо два числа, що порівнюються, рівні між собою, то прапорець нуля встановлюється в 1 (див. виконання команд **CMP** або **CPI,** лабораторна робота №4). В такому випадку в програмі буде здійснено перехід. Аналогічно, команду ПЕРЕХІД, ЯКЩО НЕ НУЛЬ можна вважати командою ПЕРЕХІД, ЯКЩО НЕ РІВНЕ.

1	Перехід, якщо результат нуль	
2	безпосередня	
3	**JZ B_2, B_3**	
4	**B_3 B_2 → PC**	
5	CAh	312q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Якщо прапорець нуля встановлено в 1, то під час виконання команди в молодший байт лічильника команд **PC** записується другий байт команди **B_2**, а в старший байт **PC** - третій байт команди **B_3**. Керування передається команді, яка буде занесена в **PC** (за адресою **B_3 B_2).** Якщо прапорець нуля встановлено в 0, то керування передається наступній команді. (1 цикл, 5 тактів).	
8	Виконання команди не впливає на регістр стану.	

1	Перехід, якщо результат не нуль	
2	безпосередня	
3	**JNZ B_2, B_3**	
4	**B_3 B_2 → PC**	
5	C2h	302q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Якщо прапорець нуля встановлено в 0, то під час виконання команди в молодший байт лічильника команд **PC** записується другий байт команди **B_2**, а в старший байт **PC** - третій байт команди **B_3**. Керування передається команді, яка буде занесена в **PC** (за адресою **B_3 B_2).** Якщо прапорець нуля встановлено в 1, то керування	

	передається наступній команді. (1 цикл, 5 тактів).
8	Виконання команди не впливає на регістр стану.

Команди ПЕРЕХІД, ЯКЩО ПЕРЕНЕСЕННЯ та ПЕРЕХІД, ЯКЩО НЕМА ПЕРЕНЕСЕННЯ перевіряють значення прапорця перенесення. Якщо його значення рівне 1, то це свідчить, що в результаті попередньої дії появився біт перенесення із старшого 7-го розряду в наступний розряд (або біт запозичення в 7-й розряд). Тому в результаті виконання команди ПЕРЕХІД, ЯКЩО ПЕРЕНЕСЕННЯ буде здійснено перехід за адресою, яка міститься в другому та третьому байтах програми, а в випадку нульового значення прапорця такий перехід здійснено не буде. Якщо прапорець встановлено в 0, то навпаки, команда ПЕРЕХІД, ЯКЩО НЕМА ПЕРЕНЕСЕННЯ буде виконуватись з переходом, а команда ПЕРЕХІД, ЯКЩО ПЕРЕНЕСЕННЯ - без.

Команди ПЕРЕХІД, ЯКЩО ПЕРЕНЕСЕННЯ та ПЕРЕХІД, ЯКЩО НЕМА ПЕРЕНЕСЕННЯ використовують, наприклад, для порівняння двох чисел A та B. Якщо A < B, то в результаті віднімання A - B появиться біт запозичення в старший розряд, і прапорець перенесення буде встановлено в 1. Якщо прапорець перенесення встановиться в 0, то це означає, що A ≥ B. Якщо ж одночасно прапорець нуля рівний 0 (ненульовий результат віднімання) тоді A > B. Таке віднімання можна здійснити за допомогою команди ПОРІВНЯННЯ. (див. виконання команд **CMP** або **CPI,** лабораторна робота №4).

1	Перехід, якщо є перенесення	
2	безпосередня	
3	**JC B_2, B_3**	
4	**B_3 B_2 → PC**	
5	DAh	332q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Якщо прапорець перенесення встановлено в 1, то під час виконання команди в молодший байт лічильника команд **PC** записується другий байт команди **B_2**, а в старший байт **PC** - третій байт команди **B_3**. Керування передається команді, яка буде занесена в **PC** (за адресою **B_3 B_2).** Якщо прапорець перенесення встановлено в 0, то керування передається наступній команді. (1 цикл, 5 тактів).	
8	Виконання команди не впливає на регістр стану.	

1	Перехід, якщо нема перенесення	
2	безпосередня	
3	**JNC B_2, B_3**	
4	**B_3 B_2 → PC**	
5	D2h	322q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Якщо прапорець перенесення встановлено в 0, то під час виконання команди в молодший байт лічильника команд **PC** записується другий	

	байт команди **B_2**, а в старший байт **PC** - третій байт команди **B_3**. Керування передається команді, яка буде занесена в **PC** (за адресою **B_3 B_2).** Якщо прапорець перенесення встановлено в 1, то керування передається наступній команді. (1 цикл, 5 тактів).
8	Виконання команди не впливає на регістр стану.

Команди ПЕРЕХІД, ЯКЩО МІНУС та ПЕРЕХІД, ЯКЩО ПЛЮС перевіряють значення прапорця знаку. Якщо його значення рівне 1, то це свідчить, що в результаті попередньої дії старший 7-й розряд результату встановлено в 1. В загальному випадку це не означає, що результат - дійсно від'ємне число. Наприклад, результат додавання додатних чисел 01110000b (70h) та 00010000b (10h) є "від'ємне" число 10000000b (80h). Однак, якщо програма оперує з числами, записаними в оберненому та доповненому кодах, 7-й розряд байта результату вказує на знак числа і може бути використаний програмістом. Тому, якщо прапорець знаку встановлено в 1, в результаті виконання команди ПЕРЕХІД, ЯКЩО МІНУС буде здійснено перехід за адресою, яка міститься в другому та третьому байтах програми, а в випадку нульового значення прапорця такий перехід здійснено не буде. Якщо прапорець встановлено в 0, то навпаки, команда ПЕРЕХІД, ЯКЩО ПЛЮС буде виконуватись з переходом, а команда ПЕРЕХІД, ЯКЩО МІНУС - без.

Команди ПЕРЕХІД, ЯКЩО МІНУС та ПЕРЕХІД, ЯКЩО ПЛЮС можна використовувати для порівняння двох чисел A та B **тільки в тому випадку**, якщо ці числа записані в прямому або оберненому кодах, оскільки в цьому випадку в старшому 7-му розряді додатне число повинно містити 0, а від'ємне - 1.

1	Перехід, якщо результат від'ємний	
2	безпосередня	
3	**JM B_2, B_3**	
4	**B_3 B_2 → PC**	
5	F2h	372q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Якщо прапорець знаку встановлено в 1, то під час виконання команди в молодший байт лічильника команд **PC** записується другий байт команди **B_2**, а в старший байт **PC** - третій байт команди **B_3**. Керування передається команді, яка буде занесена в **PC** (за адресою **B_3 B_2).** Якщо прапорець знаку встановлено в 0, то керування передається наступній команді (1 цикл, 5 тактів).	
8	Виконання команди не впливає на регістр стану.	

1	Перехід, якщо результат додатній
2	безпосередня
3	**JP B_2, B_3**
4	**B_3 B_2 → PC**

5	FAh	362q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Якщо прапорець знаку встановлено в 0, то під час виконання команди в молодший байт лічильника команд **PC** записується другий байт команди $\mathbf{B_2}$, а в старший байт **PC** - третій байт команди $\mathbf{B_3}$. Керування передається команді, яка буде занесена в **PC** (за адресою $\mathbf{B_3\ B_2}$**).** Якщо прапорець знаку встановлено в 1, то керування передається наступній команді. (1 цикл, 5 тактів).	
8	Виконання команди не впливає на регістр стану.	

Команди ПЕРЕХІД, ЯКЩО ПАРНІСТЬ та ПЕРЕХІД, ЯКЩО НЕ ПАРНІСТЬ перевіряють значення прапорця парності. Якщо його значення рівне 1, то це свідчить, що в результат попередньої дії має **парну** кількість розрядів, встановлених в 1. Тому в результаті виконання команди ПЕРЕХІД, ЯКЩО ПАРНІСТЬ буде здійснено перехід за адресою, яка міститься в другому та третьому байтах програми, а в випадку нульового значення прапорця такий перехід здійснено не буде. Якщо прапорець встановлено в 0, то навпаки, команда ПЕРЕХІД, ЯКЩО НЕ ПАРНІСТЬ буде виконуватись з переходом, а команда ПЕРЕХІД, ЯКЩО ПАРНІСТЬ - без.

Перевірка на парність використовується для виявлення помилок під час передавання чи приймання даних через зовнішній порт. В такому випадку, після передавання даних обов'язково передається так звана контрольна сума, а в спрощеному варіанті - біт парності. Цей біт встановлюється таким чином, щоб загальна сума одиниць в усіх байтах повідомлення, включаючи біт парності була парна. Якщо пристрій - приймач виявить непарну кількість бітів, то це свідчить про помилку передавання. Таким чином перевірку на парність можна використовувати, здійснюючи введення даних з накопичувачів чи мережі.

1	Перехід, якщо результат парний	
2	безпосередня	
3	**JPE** $\mathbf{B_2, B_3}$	
4	$\mathbf{B_3\ B_2 \rightarrow}$ **PC**	
5	EAh	352q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Якщо прапорець парності встановлено в 1, то під час виконання команди в молодший байт лічильника команд **PC** записується другий байт команди $\mathbf{B_2}$, а в старший байт **PC** - третій байт команди $\mathbf{B_3}$. Керування передається команді, яка буде занесена в **PC** (за адресою $\mathbf{B_3\ B_2}$**).** Якщо прапорець парності встановлено в 0, то керування передається наступній команді. (1 цикл, 5 тактів).	
8	Виконання команди не впливає на регістр стану.	

1	Перехід, якщо результат не парний
2	безпосередня

3	**JPO B_2, B_3**	
4	**B_3 B_2 → PC**	
5	E2h	342q
6	Команда займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів.	
7	Якщо прапорець парності встановлено в 0, то під час виконання команди в молодший байт лічильника команд **PC** записується другий байт команди **B_2**, а в старший байт **PC** - третій байт команди **B_3**. Керування передається команді, яка буде занесена в **PC** (за адресою **B_3 B_2).** Якщо прапорець парності встановлено в 1, то керування передається наступній команді. (1 цикл, 5 тактів).	
8	Виконання команди не впливає на регістр стану.	

Команди умовних переходів використовуються для реалізації операторів типу:

1. **for i=n to m do** тіло циклу,
2. **if** умова **then** дія1 **else** дія2,
3. **case** операнд **of** значення 1: дія1;
   значення 2: дія 2;
   .....
   значення n: дія n;
4. **repeat** тіло циклу **until** умова,
5. **while** умова **do** тіло циклу,

та інших, які використовуються в мовах програмування високого рівня.

Розділ 6. ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

6.1 Загальні вимоги до виконання лабораторних робіт.

Виконання лабораторної роботи передбачає:

1) вивчення теоретичного матеріалу;
2) виконання завдань до лабораторної роботи;
3) написання звіту;
4) підготовку відповідей на контрольні запитання.

1) Студент повинен опрацювати теоретичний матеріал, який необхідний для виконання лабораторної роботи. Пункти теоретичного матеріалу можуть бути вибрані в довільному порядку, хоча в більшості випадків вони співпадають з порядком викладу.

2) Для кожного студента передбачено індивідуальне завдання. Номер варіанту співпадає з порядковим номером студента в списку групи.

3) Звіт повинен містити інформацію про студента (група, Ф.І.Б., № лабораторної роботи, № завдання), індивідуальне завдання до лабораторної роботи (повністю) та його рішення. В окремих випадках звіт може містити висновки до лабораторної роботи.

Звіт необхідно оформити **від руки** на чистому аркуші формату А4. Всі звіти вкладаються в єдиний файл, який студент в кінці семестру здає викладачу. Допускається використовувати звичайний учнівський зошит в клітинку для запису звітів всіх виконаних лабораторних робіт.

4) Відповіді на контрольні запитання повинні виявити як глибоке теоретичне знання матеріалу, так і вміння його застосувати на практиці.

Оцінка студента включає оцінювання виконання практичного завдання та відповідей на запитання до лабораторної роботи. Невиконання практичного завдання чи повне незнання теоретичного матеріалу означає незадовільну оцінку та не зараховування лабораторної роботи.

6.2 Лабораторна робота №1

Тема роботи: Вивчення будови та принципів роботи восьмирозрядного мікропроцесора КР580ВМ80 (Intel 8080).

Мета роботи: Вивчити основні функціональні вузли восьмирозрядного мікропроцесора, послідовність виконання ним команд програми, навчитися користуватися програмою - емулятором мікропроцесорної системи на базі КР580ВМ80.

Теоретичні відомості: Розділ 1 повністю.

Порядок виконання експериментів.

Запустіть емулятор.

Експеримент 1. Занесення початкових даних в регістри і пам'ять та виконання простої програми.

Виконання: Задайте в полі 6 емулятора початкові значення регістрів загального призначення: **A**=0F; **B**=F1; **C**=12; **D**=55; **E**=66; **H**=00; **L**=0E.

Введіть в полі 5 емулятора таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3C	INR A	A+1 -> A
0001	06	MVI B, d8	дані -> B
0002	1F	RAR	дані 1F
0003	05	DCR B	B-1 -> B
0004	81	ADD C	A+C -> A
0005	82	ADD D	A+D -> A
0006	93	SUB E	A-E -> A
0007	77	MOV M,A	A -> пам'ять (000D)
0008	76	HLT	зупинка
0009	00	NOP	команда відсутня
000A	00	NOP	команда відсутня
000B	00	NOP	команда відсутня
000C	00	NOP	команда відсутня
000D	00	NOP	команда відсутня
000E	00	NOP	команда відсутня

Задайте значення регістра PC=0000. Запустіть програму в полі 9 кнопкою "Выполнить программу". Запишіть значення регістрів загального призначення після виконання програми :

A= ; **B**= ; **C**= ; **D**= ; **E**= ; **H**= ; **L**= ; PC= .

Запишіть значення комірки пам'яті за адресою 000E: ОЗП(000E)= .

Порівняйте значення регістрів та комірок пам'яті до і після виконання програми. Зробіть висновок про арифметичні дії над числами, які були занесені в регістри.

Експеримент 2. Виконання простої програми в покомандному та потактовому режимах

Обнуліть ОПЗ та регістри шляхом натискання на кнопку "Сброс ОЗУ" та "Сброс регистров" в полі 7.

Задайте значення регістрів та внесіть програму аналогічно, як і в Експерименті 1.

Запустіть виконання програми в спочатку в покомандному, а потім в потактовому режимі шляхом натискання кнопок "Выполнить команду" та "Выполнить такт" в полі 9, відповідно.

Результати виконання програми занесіть в таблицю:

Адреса комірки ОЗП	Команда	цикл	такт	Лічильник команд	регістр команд	регістр стану (PSW, тип циклу)	регістр ознак ZSPCAC	Акумулятор

Зробіть висновок про порядок виконання програми та команд.

Запишіть програму в файл: "Файл" -> "Сохранить как..."

Експеримент 3. Вивчення способів адресації мікропроцесора КР580ВМ80.

Обнуліть пам'ять та регістри.

Занесіть та виконайте наведені нижче програми в потактовому режимі.

Результати виконання програм занесіть в відповідні таблиці типу табл.4.

Безпосередня адресація.

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП -> регістр А
0001	AA	XRA D	дані
-	-	-	-

Пряма адресація.

Занесіть початкове значення комірки ОЗП (000B)=AA

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3A	LDA adr	дані комірки ОЗП -> регістр A
0001	0B	DCX B	молодший байт адреси комірки ОЗП
0002	00	NOP	старший байт адреси комірки ОЗП
-	-	-	-
-	-	-	-
000B	AA	XRA D	дані
-	-	-	-

Регістрова адресація.

Занесіть в регістр B значення B=AA.

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	78	MOV A, B	регістр B -> регістр A
-	-	-	-

Непряма адресація.

Занесіть в регістри H та L значення H=00, L=0B; в комірку ОЗП (000B)=AA.

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	7E	MOV A, M	дані комірки ОЗП за адресою HL (H-старший байт адреси комірки ОЗП; L-молодший байт адреси комірки ОЗП) -> регістр A
-	-	-	-
-	-	-	-
000B	AA	XRA D	дані
-	-	-	-

Зробіть висновок про об'єм оперативної пам'яті який займають команди та час їх виконання для різних способів адресації.

Контрольні запитання.

1. Назвіть основні складові частини мікропроцесора.
2. Що таке регістр, і регістри яких типів входять в склад мікропроцесора?
3. Які регістри є доступними програмісту, а які - ні?
4. Чи можна змінити конфігурацію регістрів мікропроцесора?
5. Яка довжина машинного слова мікропроцесора КР580ВМ80?
6. Для чого використовуються регістри загального призначення?
7. Призначення регістра А. Дані якої довжини можна завантажувати в акумулятор?
8. Для чого використовується лічильник команд?
9. Що таке стек, і призначення вказівника стеку?
10. Поясніть призначення регістра стану, та значення окремих його прапорців.
11. Яким чином мікропроцесор виконує команду?
12. Що таке машинний цикл і чим він відрізняється від машинного такту?
13. Скільки і які типи команд може виконувати мікропроцесор КР580ВМ80?
14. Формати команд: однобайтовий, двобайтовий, трибайтовий.
15. Види адресації мікропроцесора КР580ВМ80? Який з видів адресації є найбільш економним з точки зору об'єму оперативної пам'яті, що він займає; з точки зору часу виконання?
16. Назвіть типи команд щодо функціональної ознаки.
17. В чому полягає робота програми-емулятора?
18. Який максимальний об'єм пам'яті може адресувати мікропроцесор КР580ВМ80?

6.3. Лабораторна робота №2

Тема роботи: Команди пересилання даних восьмирозрядного мікропроцесора КР580ВМ80 (Intel 8080).

Мета роботи: Вивчити основні типи команд пересилання даних та послідовність їх виконання восьмирозрядним мікропроцесором, навчитися створювати програми з їх використанням.

Теоретичні відомості: Розділ 2 повністю.

Порядок виконання експериментів.

Запустіть емулятор.

Експеримент 1. Виконання простих команд завантаження та пересилання даних.

Виконайте наведені нижче приклади програм в потактовому режимі. Зафіксуйте, скільки циклів та тактів займає виконання кожної із програм.

1. Завантаження регістра **B** числом 3Fh за допомогою команди **MVI B,** 3Fh

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	06	MVI B, d8	дані наступної комірки ОЗП → регістр **B**
0001	3F	CMC	дані 3Fh → регістр **B**
-	-	-	-

2. Завантаження регістрової пари **DE** двобайтним числом 3A76h за допомогою команди **LXI D,** 76h, 3Ah

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	11	MVI B, d8	дані наступних двох комірок ОЗП → регістр **DE**
0001	76	HLT	дані 76h → регістр **E**
0002	3A	LDA adr	дані 3Ah → регістр **D**
-	-	-	-

3. Завантаження комірки пам'яті за адресою 000Bh числом A6h за допомогою команди **MVI M,** A6h

Задайте початкові значення регістрів **H**=00; **L**=0B.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	36	MVI M, d8	дані наступної комірки ОЗП → комірка ОЗП за адресою 000Bh
0001	A6	AHA M	дані A6h → регістр **E**
-	-	-	-

4. Пересилання даних з одного регістра в інший (наприклад з **B** в **E**) за допомогою команди **MOV E, B**

Задайте початкові значення регістра **B**=55

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	58	MOV E, B	дані регістру **B** → регістр **E**
-	-	-	-

5. Обмін даними між регістрами **HL** та **DE** за допомогою команди **XCHG.** Задайте початкові значення регістрів **D**=55; **E**=66; **H**=07; **L**=0E.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	EB	XCHG	регістр **H** ↔ регістр **D** регістр **L** ↔ регістр **E**
-	-	-	-

Зробіть висновки.

Експеримент 2. Створення масиву даних в пам'яті за допомогою команд пересилання даних.

Завдання: Створити в пам'яті масив даних згідно таблиці 2.:

Таблиця 2.

Адреса	Значення	Адреса	Значення	Адреса	Значення
1020h	AAh	1024h	A2h	1028h	9Ah
1021h	A8h	1025h	A0h	1029h	98h
1022h	A6h	1026h	9Eh		
1023h	A4h	1027h	9Ch		

Звісно, можна створити програму, яка буде записувати дані в комірку пам'яті:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр А
0001	AA		дані АА
0002	32	STA	регістр А → комірка ОЗП за адресою 1020
0003	10	20	молодший байт адреси
0004	2E	10	старший байт адреси
цей фрагмент програми необхідно повторити для всіх значень таблиці			
-	-	-	-

Така програма має той недолік, що фрагмент запису числа в регістр А та перезапису його в пам'ять буде повторюватись стільки разів, скільки даних містить масив.

Неважко зауважити, що послідовність значень являє собою арифметичну прогресію з показником -2. Тому необхідно виконати такі пересилання:

AA-2×i → **M(HL+i)** для всіх i = 0...10.

Для написання програми нам потрібні деякі команди, які детальніше будуть розглядатися в наступних лабораторних роботах. Це арифметичні команди:

DCR A (DCR B) - зменшення значення регістру **A (B)** на одиницю;

INX H - збільшення значення регістрової пари **HL** на одиницю;

та команди передачі керування:

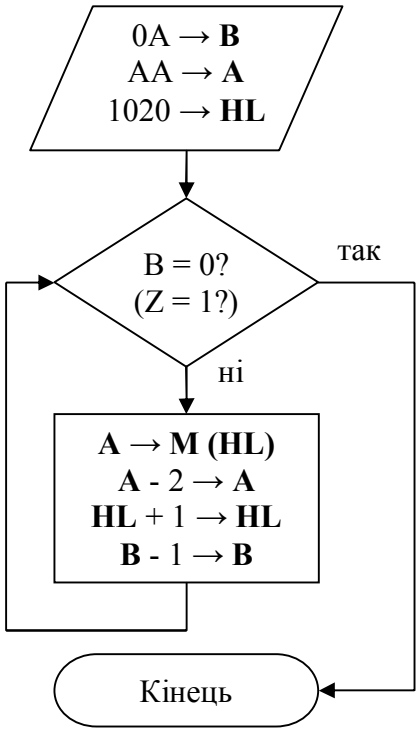
JZ adr - перехід на команду програми за адресою **adr** в випадку, якщо в результаті арифметичних дій встановлено прапорець стану **Z=1** (був нульовий результат попередніх математичних чи логічних операцій);

JMP adr - безумовний перехід на команду програми за адресою **adr**.

Команди **DCR A, DCR B, INX H** - однобайтові, результат їхнього виконання впливає на значення прапорців стану. Команди **JZ adr, JMP adr** - трибайтові, в другому і третьому байтах записана адреса комірки пам'яті, в якій міститься команда, що буде виконуватись наступною. Якщо **Z=1** не виконана, то виконується команда, що записана наступною. Умова **Z=1** може бути

виконана, якщо в результаті арифметичних дій над вмістом регістрів загального призначення чи комірки пам'яті було отримано нульовий результат.

Алгоритм програми можна представити в вигляді:



Спочатку в регістри заносяться початкові дані: в **A -** значення AA**,** в **B** значення *i* - індексу масиву та в **HL** - адреса першої комірки масиву. Далі перевіряється значення прапорця **Z.** Якщо **Z=0** тоді програма припиняє роботу, якщо ні - тоді значення регістру **A** записується в комірку пам'яті **M (HL)** значення регістру **A** зменшується на дві одиниці, значення регістрової пари **HL** збільшується на одиницю, значення індексу в регістрі **B** зменшується на одиницю. Здійснюється перехід програми на команду перевірки значення прапорця **Z** і т.д.

Занесіть програму в пам'ять та виконайте її.

	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000h	06h	MVI B, d8	дані наступної комірки ОЗП → регістр **B**
0001h	0Ah	LDAX B	дані 0A
0002h	3Eh	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0003h	AAh	XRA d	дані AA
0004h	21h	LXI H, d16	**adr** → регістрова пара **HL**
0005h	20h	-	молодший байт адреси **adr**
0006h	10h	-	старший байт адреси **adr**
0007h	CAh	JZ adr	якщо **Z**=1 , то перейти на

			адресу **adr (adr → PC)**
0008h	12h	STAX D	
0009h	00h	NOP	команда відсутня
000Ah	77h	MOV M, A	**A → M(HL)**
000Bh	3Dh	DCR A	**A** -1 → **A**
000Ch	3Dh	DCR A	**A** -1 → **A**
000Dh	23h	INX H	**HL** + 1 → **HL**
000Eh	O5h	DCR B	**B** -1 → **B**
000Fh	C3h	JMP adr	перейти на адресу **adr (adr → PC)**
0010h	07h	RLC	молодший байт адреси **adr**
0011h	00h	NOP	старший байт адреси **adr**
0012h	76h	HLT	Зупинка програми
-	-	-	-

Перевірте значення комірок пам'яті починаючи з адреси 1020h. Запишіть значення комірок. Зробіть висновки.

Контрольні запитання.

1. Призначення команд пересилання даних та їх основні типи.
2. Які із видів адресації використовують команди пересилання даних?
3. В яких випадках використовується безпосередня адресація? Її основні переваги та недоліки.
4. В яких випадках використовується регістрова адресація? Її основні переваги та недоліки
5. В яких випадках використовується пряма адресація? Її основні переваги та недоліки
6. В яких випадках використовується непряма адресація? Її основні переваги та недоліки
7. Опишіть основні типи машинних циклів.
8. Команда **LXI D, B_2, B3** займає 3 байти пам'яті, виконується за 3 цикли, 10 тактів. Опишіть, які дії виконуються в кожному із машинних циклів цієї команди.
9. За допомогою яких команд пересилання можна завантажити дані в регістр **A**?
10. Перечисліть основні команди збереження даних в пам'ять.
11. Чому команди мікропроцесора інколи зручніше записувати в вісімковій системі числення?
12. Чи міняються значення прапорців стану, якщо пересилається число 00h?
13. Команди якого типу використовуються при створенні масивів даних та роботи з ними?
14. Яким чином можна значення комірки пам'яті переписати в іншу комірку пам'яті? Які команди для цього використовуються?
15. Чому, на Вашу думку, відсутні команди пересилання даних, які використовують такі види адресації: пряму/пряму, непряму/непряму, пряму/непряму та непряму/пряму? Відповідь обґрунтуйте.

6.4. Лабораторна робота №3

Тема роботи: Арифметичні команди восьмирозрядного мікропроцесора КР580ВМ80 (Intel 8080).

Мета роботи: Вивчити основні типи арифметичних і логічних команд та послідовність їх виконання восьмирозрядним мікропроцесором, навчитися створювати програми з їх використанням.

Теоретичні відомості: Розділ 3 повністю.

Порядок виконання експериментів.

Запустіть емулятор.

Експеримент 1. Виконання простих команд 8-ми розрядного додавання даних. Розглянемо приклади до розділу 3.1.1.

Виконайте наведені нижче приклади програм в потактовому режимі. Зафіксуйте, скільки циклів та тактів займає виконання кожної із програм. Зафіксуйте значення прапорця перенесення після виконання кожної із команд.

1. Додавання без врахування попереднього перенесення. Сума доданків не перевищує 256. Додати числа 12d та 41d.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	0C	INR C	дані 0Ch → регістр **A**
0002	06	MVI B, d8	дані наступної комірки ОЗП → регістр **B**
0003	29	DAD H	дані 29h → регістр **B**
0004	80	ADD B	**A + B → A**
-	-	-	-

2. Додавання без врахування попереднього перенесення. Сума доданків не перевищує 256. Додати числа 55d та 67d.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	37	STC	дані 37h → регістр **A**

0002	0E	MVI C, d8	дані наступної комірки ОЗП → регістр **C**
0003	43	MOV B, E	дані 43h → регістр **C**
0004	81	ADD C	**A + C → A**
-	-	-	-

3. Додавання без врахування попереднього перенесення. Сума доданків перевищує 256. Додати числа 156d та 189d.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	9C	SBB H	дані 9Ch → регістр **A**
0002	16	MVI D, d8	дані наступної комірки ОЗП → регістр **D**
0003	BD	CMP L	дані BDh → регістр **D**
0004	82	ADD C	**A + D → A**
-	-	-	-

4. Віднімання без врахування попереднього перенесення. Зменшуване більше від'ємника. Від числа 156d відняти 41d.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	9C	SBB H	дані 9Ch → регістр **A**
0002	1E	MVI E, d8	дані наступної комірки ОЗП → регістр **E**
0003	29	DAD H	дані 29h → регістр **E**
0004	93	SUB E	**A - E → A**
-	-	-	-

5. Віднімання без врахування попереднього перенесення. Зменшуване менше від'ємника. Від числа 12d відняти 41d.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**

0001	0C	INR C	дані 0Ch → регістр **A**
0002	26	MVI H, d8	дані наступної комірки ОЗП → регістр **H**
0003	29	DAD H	дані 29h → регістр **H**
0004	82	ADD H	**A - H → A**
-	-	-	-

Зробіть висновки.

Експеримент 2. Виконання простих команд 16-ти розрядного додавання даних.

Попередньо занесіть такі числа в регістри:

B = 11, C = 22, D = 33, E = 44, H = 55, L = 66, SP = 0080.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	09	DAD B	HL + BC → HL
0001	19	DAD D	HL + DE → HL
0002	29	DAD H	HL + HL → HL
0003	39	DAD SP	HL + SP → HL
-	-	-	-

Прослідкуйте за значеннями прапорця перенесення C.

Зробіть висновки.

Експеримент 3. Додавання з урахуванням попереднього перенесення.

Завдання: необхідно додати 2 чотирибайтні числа 12889513h та 32C2B0AAh. Оскільки регістрів загального призначення не вистачить, то розмістимо дані в пам'яті. Туди ж будемо записувати суму. Нехай 1-й доданок займає в пам'яті адреси починаючи з 0030h, 2-й доданок - починаючи з 0035 h, суму будемо записувати починаючи з 003Ah. В такому разі адресу першого доданку запишемо в регістровій парі **BC**, другого - в **HL**, суми - в **DE**.

Попередньо необхідно занести числа в пам'ять.

Адреса комірки ОЗП	1-й доданок	Адреса комірки ОЗП	2-й доданок
0030	13	0035	AA
0031	95	0036	B0
0032	88	0037	C2
0033	12	0038	32

Байти заносяться в пам'ять згідно принципу: Старші байти в комірки з більшою адресою.

Введемо та виконаємо програму.

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	01	LXI B, d16	**0030h → BC**
0001	30	-	
0002	00	NOP	
0003	11	LXI D, d16	**003Ah → HL**
0004	3A	LDA adr	
0005	00	NOP	
0006	21	LXI H, d16	**0035h → DE**
0007	35	DCR M	
0008	00	NOP	
0009	0A	LDAX B	**M(BC) → A**
000A	86	ADD M	**A + M(HL) → A**
000B	12	STAX D	**A → M(DE)**
000C	03	INX B	**BC + 1 → BC**
000D	13	INX D	**DE + 1 → DE**
000E	23	INX H	**HL + 1 → HL**
000F	0A	LDAX B	**M(BC) → A**
0010	8E	ADC M	**A + M(HL) + C→ A**
0011	12	STAX D	**A → M(DE)**
0012	03	INX B	**BC + 1 → BC**
0013	13	INX D	**DE + 1 → DE**
0014	23	INX H	**HL + 1 → HL**
0015	0A	LDAX B	**M(BC) → A**
0016	8E	ADC M	**A + M(HL) + C→ A**
0017	12	STAX D	**A → M(DE)**
0018	03	INX B	**BC + 1 → BC**
0019	13	INX D	**DE + 1 → DE**
001A	23	INX H	**HL + 1 → HL**
001B	0A	LDAX B	**M(BC) → A**
001C	8E	ADC M	**A + M(HL) + C→ A**
001D	12	STAX D	**A → M(DE)**
001E	76	HLT	Зупинка програми
-	-	-	-

Перевірте результат виконання програми в комірках пам'яті починаючи з адреси 003Ah. Перевірте правильність додавання та врахування перенесень. Зробіть висновки.

Контрольні запитання.

1. Основні типи арифметичних команд та їх призначення.
2. Які із видів адресації використовують арифметичні команди?
3. Перерахуйте прапорці регістра стану. За яких умов вони встановлюються в 0 або 1?
4. На які прапорці впливає результат виконання арифметичних команд?
5. Які із видів адресації використовують арифметичні команди?
6. З якою метою використовуються команди додавання з перенесенням та віднімання з запозиченням. На скільки циклів та тактів вони довші ніж аналогічні арифметичні команди без урахування перенесення. Чому?
7. В яких випадках використовують команди 16-ти розрядного додавання? Чи можна їх використовувати в випадку додавання більш, ніж двобайтних чисел? Відповідь обґрунтуйте.
8. Де повинні знаходитись операнди та результат в випадку використання команд 8-ми розрядного додавання та віднімання. В випадку 16-ти розрядного додавання?
9. Яким чином можна здійснити віднімання 16-ти розрядних чисел?
10. Яким чином здійснюється представлення чисел в двійково-десятковому коді? Наведіть приклади використання такого коду.
11. З якою метою використовується команда десяткової корекції?
12. Чи можна застосовувати команду десяткової корекції до результату віднімання двох чисел? Відповідь обґрунтуйте.
13. Які команди інкременту - декременту здійснюються в мікропроцесорі КР580ВМ80? Яку адресацію використовують ці команди, та на які прапорці стану впливає результат їх виконання?
14. Наведіть приклади використання команди інкременту.
15. З якою метою, як правило, використовують команди від'ємного приросту?
16.* Складіть програму додавання двох двобайтних чисел в двійково-десятковому коді без використання команди десяткової корекції

6.5. Лабораторна робота №4

Тема роботи: Логічні команди восьмирозрядного мікропроцесора КР580ВМ80 (Intel 8080).

Мета роботи: Вивчити основні типи логічних команд та послідовність їх виконання восьмирозрядним мікропроцесором, навчитися створювати програми з їх використанням.

Теоретичні відомості: Розділ 4 повністю.

Порядок виконання експериментів.

Запустіть емулятор.

Експеримент 1. Виконання простих команд "**І**", "**АБО**", "**Заперечення**", "**Виключне АБО**".

Виконайте наведені нижче приклади програм в потактовому режимі. Зафіксуйте, скільки циклів та тактів займає виконання кожної із програм. Зафіксуйте значення прапорців після виконання кожної із команд. На прикладі команд "**І**" та "**АБО**" продемонстровано маскування чисел, а на прикладі команди "**АБО**" також і способи адресації другого операнда.

1. Маскування молодших чотирьох бітів. Над числами CAh та F0h виконати порозрядне "**І**".

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	CA	JZ adr	дані CAh → регістр **A**
0002	06	MVI B, d8	дані наступної комірки ОЗП → регістр **B**
0003	F0	RP	дані F0h → регістр **B**
0004	A0	ANA B	**A & B → A**
-	-	-	-

2. Виділення молодшого біта числа. Над числами CAh та 01h виконати порозрядне "**І**".

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**

0001	CA	JZ adr	дані CAh → регістр **A**
0002	06	MVI B, d8	дані наступної комірки ОЗП → регістр **B**
0003	01	LXI B, d16	дані 01h → регістр **B**
0004	A0	ANA B	**A & B → A**
-	-	-	-

3. Над числами BAh та 0Fh виконати порозрядне "**АБО**" використовуючи безпосередню адресацію.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	BA	CMP D	дані BAh → регістр **A**
0002	F6	ORI, d8	**A** ∨ значення наступного байту програми → **A**
0003	0F	RRC	дані 0Fh
-	-	-	-

4. Над числами BAh та 0Fh виконати порозрядне "**АБО**" використовуючи регістрову адресацію.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	BA	CMP D	дані BAh → регістр **A**
0002	06	MVI B, d8	дані наступної комірки ОЗП → регістр **B**
0003	0F	RRC	дані 0Fh → регістр **B**
0004	B0	ORA B	**A** ∨ **B** → **A**
-	-	-	-

5. Над числами BAh та 0Fh виконати порозрядне "**АБО**" використовуючи непряму адресацію.

Попередньо задайте: H = 00h; L = 10h; M(0010) = 0F.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки

			ОЗП → регістр **A**
0001	BA	CMP D	дані BAh → регістр **A**
0002	B6	ORA M	**A** ∨ **M(HL)** → **A**
-	-	-	-

6. Над числом A3h виконати порозрядне "**Заперечення**".

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	A3	ANA E	дані A3h → регістр **A**
0002	2F	CMA	$\overline{A}$ → **A**
-	-	-	-

7. Над числами BAh та 3Ah виконати порозрядне "**Виключне АБО**" використовуючи регістрову адресацію.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	BA	CMP D	дані BAh → регістр **A**
0002	16	MVI D, d8	дані наступної комірки ОЗП → регістр **D**
0003	3A	LDA adr	дані 3Ah → регістр **D**
0004	AA	XRA D	**A** ⊕ **D** → **A**
-	-	-	-

8. Перевірка співпадіння чисел. Перевірити, чи в акумуляторі дійсно знаходиться число BBh, виконавши порозрядне "**Виключне АБО**".

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	BB	CMP E	дані BBh → регістр **A**
0002	EE	XRI, d8	**A** ⊕ значення наступного байту програми → **A**
0003	BB	CMP E	дані BBh
-	-	-	-

9. Обнулення акумулятора виконанням порозрядного "**Виключне АБО**". Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	5F	MOV E, A	дані 5Fh → регістр **A**
0002	AF	XRA A	**A** ⊕ **A** → **A**
-	-	-	-

Зробіть висновки стосовно кожної із використаних функцій.

Експеримент 2. Вивчення команд порівняння чисел. Особливу увагу звертайте на значення прапорців регістру стану після виконання команди порівняння.

1. Порівняти число BAh з 0Fh використовуючи безпосередню адресацію. Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	BA	CMP D	дані BAh → регістр **A**
0002	FE	CPI, d8	**A -** значення наступного байту програми → регістр стану
0003	0F	RRC	дані 0Fh
-	-	-	-

2. Порівняти число 0Fh з BAh використовуючи регістрову адресацію. Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	0F	RRC	дані 0Fh → регістр **A**
0002	06	MVI B, d8	дані наступної комірки ОЗП → регістр **B**
0003	BA	CMP D	дані BAh → регістр **B**
0004	B8	CMP B	**A - B** → регістр стану
-	-	-	-

3. Порівняти числа 7Ah та 7Ah використовуючи непряму адресацію.
Попередньо задайте: H = 00h; L = 10h; M(0010) = 7A.
Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	7A	MOV A, D	дані 7Ah → регістр **A**
0002	BE	CMP M	**A - M(HL)** → регістр стану
-	-	-	-

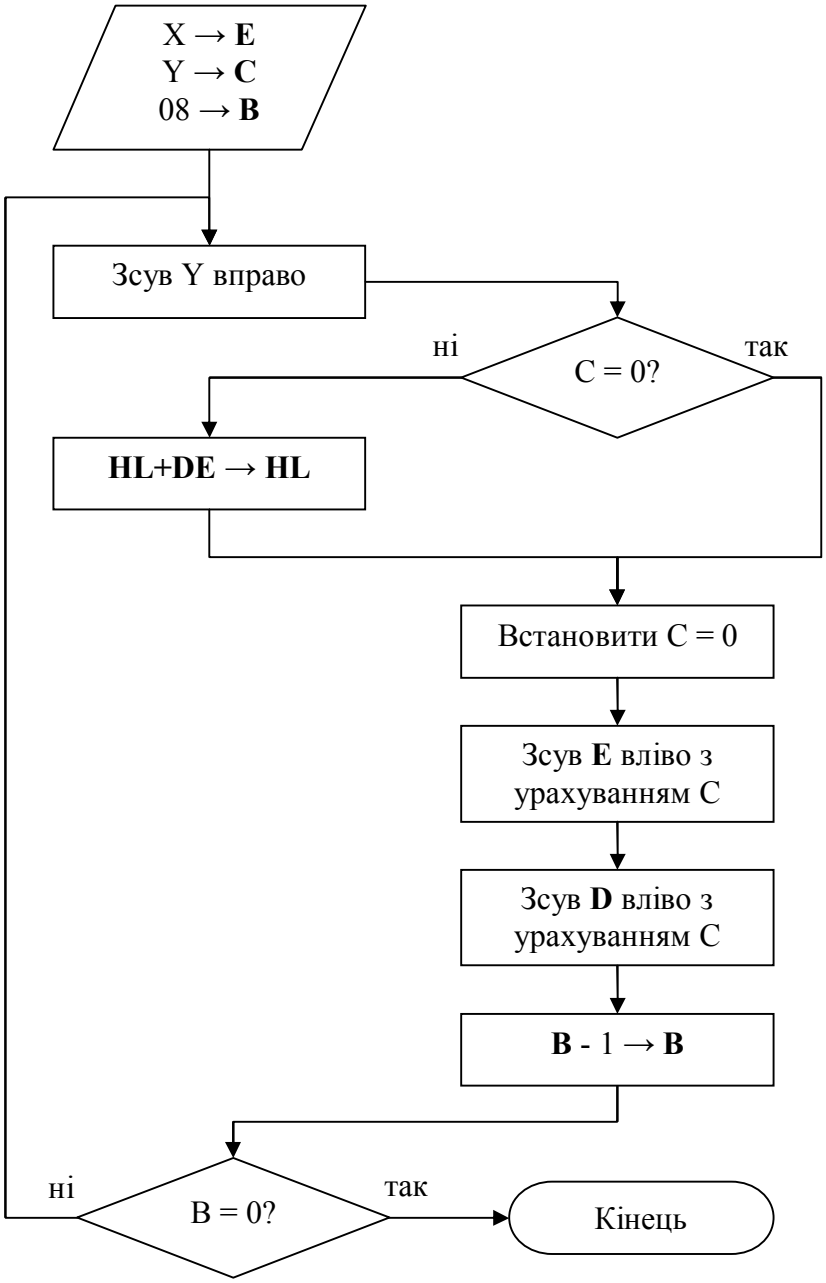
Зробіть висновки.

Експеримент 3. Вивчення команд циклічного перенесення та використання їх для реалізації множення 8-ми розрядних двійкових чисел.

Завдання: необхідно перемножити два 8-ми розрядні числа X та Y. Як показано вище в розділі 3.1.7., множення двох 8-ми розрядних чисел можна реалізувати шляхом накопичення числа в 16 розрядному регістрі в результаті послідовного виконання операцій додавання та зсуву. Таким чином, необхідно один 16-ти розрядний регістр для накопичення суми, другий 16-ти розрядний регістр для формування доданків, один 8-розрядний регістр для збереження циклічно зсунутого числа Y на певне число розрядів. Ще один 8-розрядний регістр необхідно для запису максимального індексу циклу. Для збереження всіх необхідних кінцевих і проміжних результатів виконання програми достатньо регістрів загального призначення.

Нехай X записано в регістр **E** (фактично в регістровій парі **DE**), а Y - в регістр **C**. В регістр **B** запишемо максимальну кількість доданків - 8. Це і буде максимальний індекс циклу. В регістровій парі **HL** будемо накопичувати добуток, який буде формуватись таким чином: число X, помножене на 2^i (що відповідає зсуву вліво на i розрядів) буде додаватись до добутку тоді, коли вага i−того розряду числа Y становить 1. Якщо цей розряд рівний нулю, то i-те накопичення добутку буде відсутнє.

Алгоритм множення чисел може бути, наприклад, таким:



В представленому алгоритмі для визначання ваги і–того розряду числа Y використовується і–тий зсув числа Y вправо. Тоді і–тий розряд числа Y перейде в прапорець перенесення C, і його вагу можна визначити. Якщо C=1, то виконується накопичення добутку, в іншому випадку відбувається множення числа X на 2, що еквівалентне його зсуву вліво на 1 розряд. Далі індекс циклу зменшується на 1 і, якщо він буде рівним нулю, програма завершиться. Якщо індекс циклу не рівний нулю весь цикл буде виконано знову. Зверніть увагу на те, що перед початком зсуву числа в регістровій парі **DE** необхідно встановити C=0, оскільки внаслідок попередніх арифметичних дій прапорець C міг бути встановлений в 1. В результаті зсуву числа вліво завжди в наймолодший розряд записується значення або старшого розряду (команда **RLC**) або прапорця перенесення C (команда **RAL**). Тому значення прапорця C необхідно обнулити.

На основі даного алгоритму можна написати таку програму:

Адреса комірки ОЗП	Код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	1E	MVI E, d8	X → **E**
0001	0F	RRC	
0002	0E	MVI C, d8	Y → **C**
0003	0F	RRC	
0004	06	MVI B, d8	08h → **B**
0005	08	-	
0006	00	NOP	
0007	79	MOV A, C	Зсув числа в регістрі **C** на 1 розряд вправо
0008	0F	RRC	
0009	4F	MOV C, A	
000A	D2	JNC adr	Якщо C = 0, тоді перейти до виконання команди за адресою 000Eh
000B	0E	MVI C, d8	
000C	00	NOP	
000D	19	DAD D	Накопичення добутку в регістровій парі **HL**
000E	37	STC	Обнулення прапорця C.
000F	3F	CMC	
0010	7B	MOV A, E	Зсув числа в регістрі **E** на 1 розряд вліво
0011	17	RAL	
0012	5F	MOV E, A	
0013	7A	MOV A, D	Зсув числа в регістрі **D** на 1 розряд вліво враховуючи перенесення з регістру **E**.
0014	17	RAL	
0015	57	MOV D, A	
0016	05	DCR B	**B**-1 → **B**
0017	C2	JNZ adr	Якщо **B** ≠ 0, тоді перейти до виконання команди за адресою 0007h
0018	07	RLC	
0019	00	NOP	
001A	76	HLT	Кінець програми
001B	-	-	-

Введіть та виконайте програму. Перевірте результат виконання програми шляхом множення двох 8-ми розрядних чисел в стовпчик. Зробіть висновки.

Увага: Дана програма перемножує два числа X = 0Fh та Y = 0Fh. Для множення інших чисел необхідно заносити X в комірку 0001 = 0Fh та Y в комірку 0003h.

Програму можна модифікувати наступним чином:

1. В комірки пам'яті 0000h-0003h запишіть нульові команди. Це означатиме, що:

2. Перед запуском програми числа X та Y необхідно заносити в регістри **E** та **C**, відповідно.

Зробіть висновки.

Контрольні запитання.

1. Які основні логічні функції реалізують логічні команди? З якою метою їх використовують?
2. Назвіть основні відмінності між логічними та арифметичними командами.
3. Чи можна команди порозрядного "**І**" чи "**АБО**" застосовувати до сусідніх розрядів регістру **А**?
4. На які прапорці регістра стану впливає результат виконання логічних команд?
5. В чому полягає сутність порозрядного маскування? За допомогою яких логічних команд його реалізують? З якою метою його використовують?
6. Які із способів адресації операндів використовують логічні команди? Де повинен знаходитися перший операнд?
7. Опишіть логічну функцію "**Виключне АБО**". Наведіть приклади використання команд, які її реалізують.
8. З якою метою використовують команди "**Порівняння**"? Які переваги вони мають над командами "**Виключне АБО**"?
9. Наведіть три способи порівняння чисел. Опишіть переваги і недоліки кожного із способів.
10. З якою метою використовують команди зміни прапорця перенесення С?.
11. Опишіть різновидності команд циклічного зсуву. Наведіть приклади використання цих команд. На які прапорці регістра стану впливає результат виконання цих команд?
12. Яким чином реалізується функція множення двох чисел в мікропроцесорі КР580ВМ80?
13. Чому операція циклічного зсуву вліво еквівалентна множенню числа в акумуляторі на 2? Якій арифметичній дії еквівалентна операція циклічного зсуву вправо? Відповідь обґрунтуйте.
14. Яким чином в мікропроцесорі КР580ВМ80 реалізувати функцію ділення двох чисел?
15. Чи можна за допомогою програми експерименту 3 здійснити множення чисел в оберненому та доповненому кодах? Яким чином необхідно модифікувати дану програму для реалізації таких функцій?

6.6. Лабораторна робота №5

Тема роботи: Команди умовних та безумовних переходів восьмирозрядного мікропроцесора КР580ВМ80 (Intel 8080).

Мета роботи: Вивчити основні типи команд умовних та безумовних переходів, основні способи їх застосування, навчитися створювати програми з їх використанням.

Теоретичні відомості: Розділ 5 повністю.

Порядок виконання експериментів.

Запустіть емулятор.

Виконайте наведені нижче приклади програм в потактовому режимі. Зафіксуйте, скільки циклів та тактів займає виконання кожної із програм. Зафіксуйте значення прапорців після виконання кожної із команд.

Експеримент 1. Вивчення команд безумовного переходу.

1. Вивчення команди безумовного переходу **JMP adr.**

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	01	LXI B, d16	дані 01h → регістр **A**
0002	3C	INR A	**A** + 1 → регістр **A**
0003	C3	JMP adr	перехід на адресу 000Ah
0004	0A	LDAX B	молодший байт адреси
0005	00	NOP	старший байт адреси
0006	3D	DCR A	**A -** 1 → регістр **A**
0007	47	MOV B, A	**A** → регістр **B**
0008	80	ADD B	**A** + **B** → регістр **A**
0009	A0	ANA B	**A & B** → регістр **A**
000A	C3	JMP adr	перехід на адресу 0002h
000B	02	STAX B	молодший байт адреси
000C	00	NOP	старший байт адреси
000D	76	HLT	зупинка програми
-	-	-	-

Складіть блок-схему алгоритму програми. Покажіть, які із команд програми будуть виконуватися, а які - ні. За яких умов програма закінчить своє виконання, і який результат буде отримано?

2. Вивчення команди безумовного переходу **PCHL.**

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	3E	MVI A, d8	дані наступної комірки ОЗП → регістр **A**
0001	01	LXI B, d16	дані 01h → регістр **A**
0002	26	MVI H, d8	дані наступної комірки ОЗП → регістр **H**
0003	00	NOP	старший байт адреси
0004	2E	MVI L, d8	дані наступної комірки ОЗП → регістр **L**
0005	0A	LDAX B	молодший байт адреси
0006	3C	INR A	**A** + 1 → регістр **A**
0007	E9	PCHL	**HL** → регістр **PC**
0008	24	INR H	**H** + 1 → регістр **H**
0009	2C	INR L	**L** + 1 → регістр **L**
000A	3D	DCR A	**A** - 1 → регістр **A**
000B	87	ADD A	**A** + **A** → регістр **A**
000C	C3	JMP adr	перехід на адресу 0002h
000D	02	STAX B	молодший байт адреси
000E	00	NOP	старший байт адреси
000F	76	HLT	зупинка програми
-	-	-	-

Які з команд програми ніколи не будуть виконані? Чим відрізняється командами **PCHL** від команди **JMP adr**? Зробіть висновки.

Експеримент 2 Вивчення команд умовного переходу.

1. Вивчення команд умовного переходу **JZ adr** та **JNC adr.**

Постановка задачі: В регістрах B та C записані числа, які необхідно порівняти. Після виконання програми в регістрі B повинно бути записане більше число. Якщо числа одинакові, то регістри B та C треба обнулити.

Введіть таку програму:

Адреса комірки ОЗП	код команди	Мнемонічний запис команди	Команда мікропроцесора
0000	78	MOV A, B	**B** → регістр **A**

0001	91	SUB C	**A - C** → регістр **A**
0002	CA	JZ adr	ПЕРЕХІД, ЯКЩО НУЛЬ на адресу 000Eh
0003	0E	MVI C, d8	молодший байт адреси
0004	00	NOP	старший байт адреси
0005	D2	JNC adr	ПЕРЕХІД, ЯКЩО НЕМА ПЕРЕНЕСЕННЯ на адресу 0011h
0006	11	LXI D, d16	молодший байт адреси
0007	00	NOP	старший байт адреси
0008	78	MOV A, B	**B** → регістр **A**
0009	41	MOV B, C	**C** → регістр **B**
000A	4F	MOV C, A	**A** → регістр **C**
000B	C3	JMP adr	ПЕРЕХІД на адресу 0011h
000C	11	LXI D, d16	молодший байт адреси
000D	00	NOP	старший байт адреси
000E	AF	XRA A	обнулення **A**
000F	47	MOV B, A	**A** → регістр **B**
0010	4F	MOV C, A	**A** → регістр **C**
0011	76	HLT	зупинка програми
-	-	-	-

Задайте початкові значення B=50h, C=20h. Запустіть програму на виконання в покроковому режимі. Запишіть значення прапорців стану після виконання команди **SUB C**. Запишіть всі команди, які будуть виконані наступними. Задайте інші значення регістрів: спочатку B=35h, C=67h, а потім B=33h, C=33h. Запустіть програму для кожного із наборів. Прослідкуйте за значеннями прапорців стану, та послідовністю виконання команд. Складіть блок-схему алгоритму програми. Зробіть висновки.

2. Вивчення команд **JPE adr** та **JPO adr.**

Постановка задачі.

Контрольні запитання.

1. Яке основне призначення команд безумовних та умовних переходів?
2. Яким чином здійснюються безумовні та умовні переходи на вказану адресу?
3. На які прапорці регістра стану впливає результат виконання команд переходів?
4. Яким чином вказується адреса на яку необхідно перейти?
5. Назвіть основні недоліки команд безумовних та умовних переходів?
6. Вкажіть відмінності між командами **PCHL** та **JMP adr**. В яких випадках застосовуються дані команди?
7. Чим відрізняються команди безумовних та умовних переходів між собою?
8. З якою метою використовують команди безумовних переходів?
9. Які прапорці регістру стану використовуються командами умовних переходів?
10. Перерахуйте всі команди умовних переходів? Чому таких команд є вісім? Чи існує команда переходу, яка враховує прапорець додаткового перенесення?
11. З якою метою використовують команди переходу, які залежать від значення прапорця парності?
12. Яким чином здійснюється порівняння чисел в мікропроцесорі КР580ВМ80? Які прапорці стану використовуються при цьому?
13. В яких випадках використовують команди ПЕРЕХІД, ЯКЩО МІНУС та ПЕРЕХІД, ЯКЩО ПЛЮС?
14. Яка команда в експерименті 2.1 є лишньою?
15. Яким чином за допомогою команд умовних переходів можна реалізувати оператор **if** умова **then** дія1 **else** дія2?
16. Яким чином за допомогою команд умовних переходів можна реалізувати оператор **repeat** тіло циклу **until** умова?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Мельник А.О. Архітектура комп'ютера. - Луцьк: Волинська обласна друкарня, 2008. - 470 с.
2. Таненбаум Э. Архитектура компьютера. 5-е изд. - СПб.: Питер, 2007. - 844 с.
3. Гилмор Ч. Введение в микропроцессорную технику. - М.: Мир, 1984. - 334 с.
4. Шауман А.М. Основы машинной арифметики. - Л.: Из-во Ленингр. ун-та, 1979. 312 с.
5. Справочник по микропроцессорным устройствам. / А.А. Молчанов, В.И. Корнейчук, В.П. Тарасенко, Д.А. Россошинский. - К.: Техніка - 1987. - 288 с.
6. Морисита М. Аппаратные средства микроЭВМ. - М.: Мир, 1988. - 288 с.
7. Буреев Л.Н., Дудко А.Л., Захаров В.Н. Простейшая микро-ЭВМ: Проектирование. Наладка. Использование. - М.: Энергоатомиздат, 1989. - 216 с.

Терлецький Андрій Іванович
Фрик Оксана Богданівна

БУДОВА ТА ПРОГРАМУВАННЯ 8-РОЗРЯДНОГО МІКРОПРОЦЕСОРА

методичні рекомендації до виконання лабораторних робіт
з дисципліни "Архітектура комп'ютерів" (2-й семестр)
для студентів напряму "Комп'ютерна інженерія"

Підписано до друку хх.хх.2012 р. Формат 60×84/16
Папір офсетний, друк цифровий.
Умовн. др. арк. 9,0.
Тираж 100 пр. Зам. № ххх від хх.хх.2012 р.

Віддруковано:
Приватний підприємець Голіней О.М.,
76008, м. Івано-Франківськ, вул. Галицька, 128, тел.:(0342)580432