Головні терміни і поняття

Комп`ютер, Алгоритм, Програма, Архітектура

Комп`ютер – сукупність електронних технічних засобів для автоматизованої алгоритмічної обробки дискретних ***даних***.

Алгоритм – скінченний впорядкований набор правил для рішення завдання (ISO 2382/1-93).

Відповідно до алгоритму реалізується *обчислювальний процес*, який задається *програмою*.

Програма формується із ***команд*** (ISO 2382/1-93).

Архітектура комп'ютерів – загальні принципи побудови комп'ютерних систем та положення із реалізації взаємодії функціональних компонентів з виконання **команд** обробки **даних**.

Призначення системного програмування

Прикладна програма призначена для вирішення задач у визначеній області застосування засобів перетворення даних.

Системна програма призначена для:

- підтримки роботоспроможності засобів перетворення даних,
- підвищення ефективності використання комп’ютерних засобів.

Системне програмування - процес розробки системних програм.

Нейманівська архітектура

Ідея - в статті фон Неймана [*Von Neumann, J., First Draft of a Report on the EDVAC, Moore School, University of Pennsylvania, 1945*].

Основні принципи:

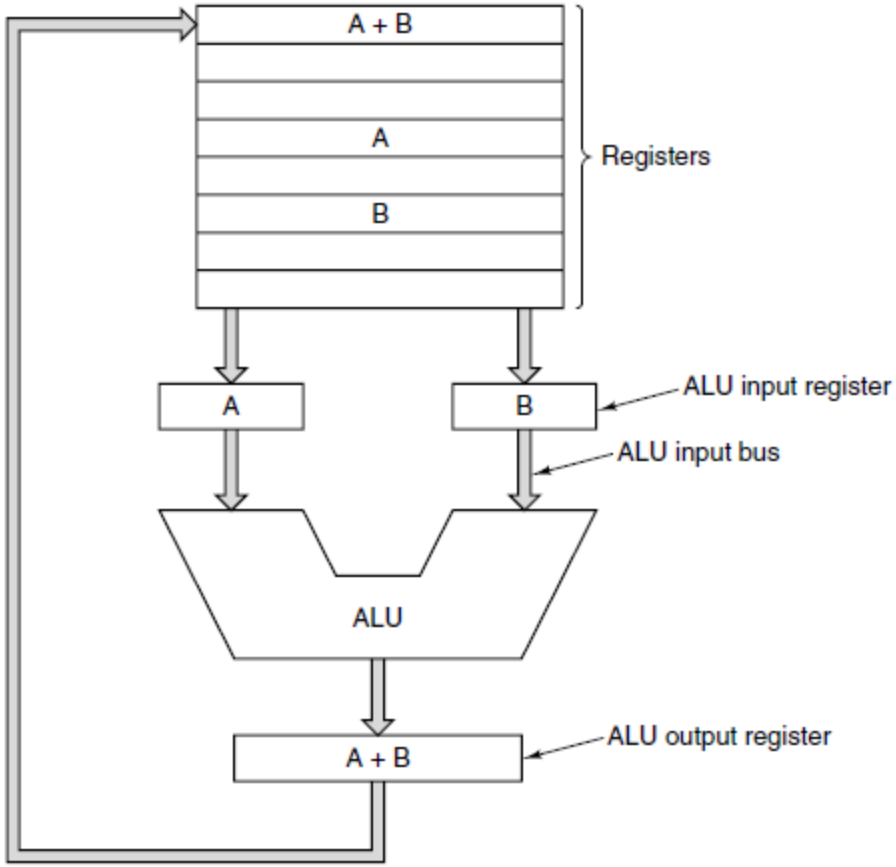
- двійкове кодування – команди і операнди кодуються двійковими цифрами 0 та 1 і мають відповідний *формат*;
- програмне керування – обчислення за алгоритмом вирішення завдання представлено у формі програми із команд, які вказують на операції для виконання відповідних дій;
- одноманітність пам'яті – команди та операнди розміщуються в спільній пам'яті і відрізняються за методом використання;
- адресуємость пам’яті – пам’ять формується із комірок з номерами (адресами).

Така архітектура має назву «*нейманівська*» або «*принстонська*».

Інша архітектура: «*гарвардська архітектура*» використовує розділену пам’ять для команд і даних.

Типова структура комп'ютера принстонської архітектури

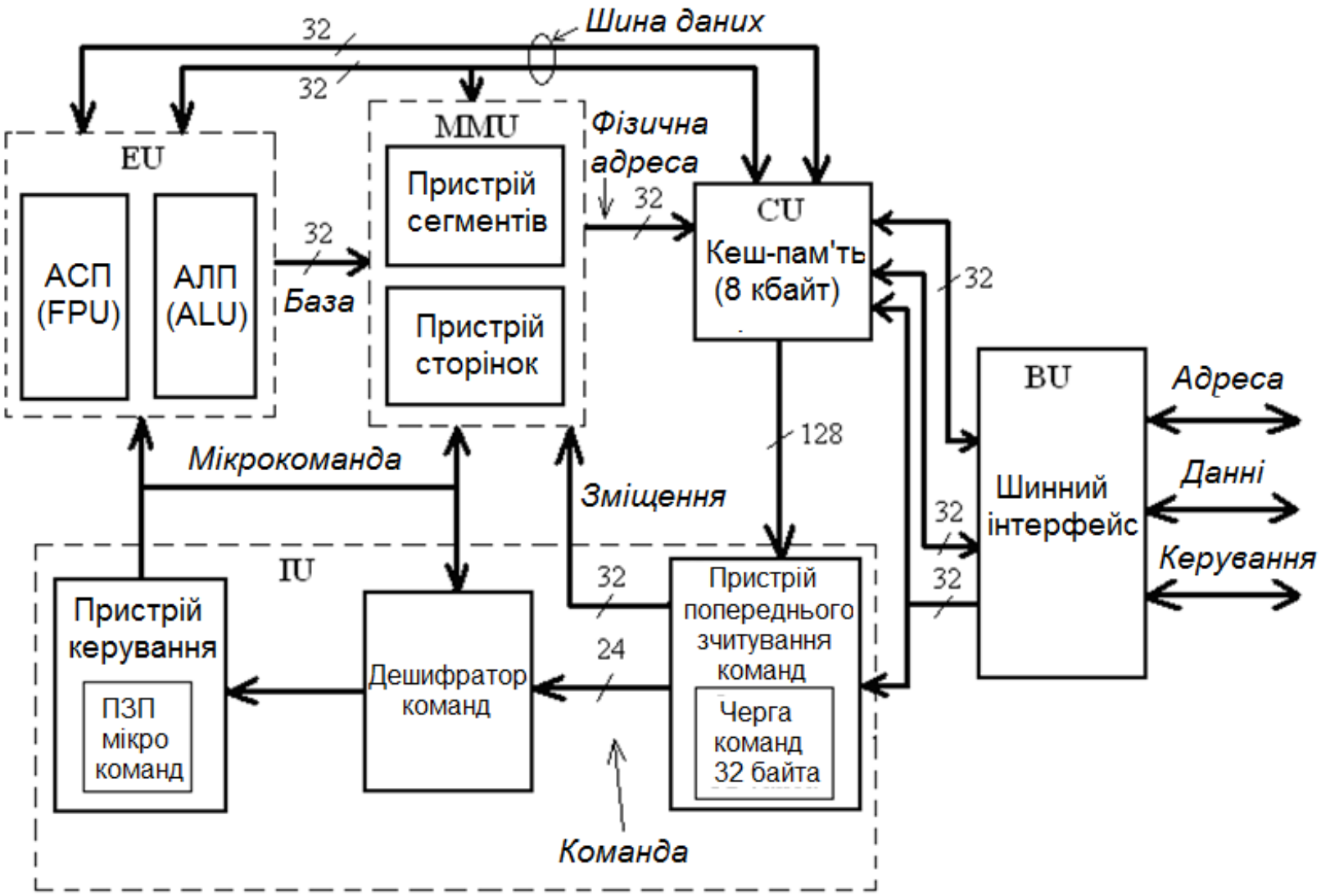
Потік даних (неймановська архітектура)



Типова послідовність дій при виконанні команд

1. Зчитування команди із пам’яті в процесор відповідно до значення покажчика адреси команд.
2. Декодування команди.
3. Формування адрес операндів і зчитування операндів із пам’яті в регістри процесору (за необхідністю).
4. Виконання операції над операндами із формуванням ознак.
5. Запис в пам’ять результату виконання операції (за необхідністю).

Типова структура процесора IA-32



Типова структура процесора IA-32 (пояснення)

Основні функціональні пристрої:

- пристрій шинного інтерфейсу (BU),
- пристрій команд (IU) – попереднє зчитування команд, дешифрація команд, мікропрограмне керування,
- виконавчий пристрій (EU) – арифметико-логічний пристрій (ALU), пристрій обчислень із рухомою точкою (FPU),
- пристрій керування пам’яттю (MMU) - керування сегментацією, перетворення сторінок адрес,
- кеш-пам’ять (CU – Cache Unit).

Архітектура IA-32

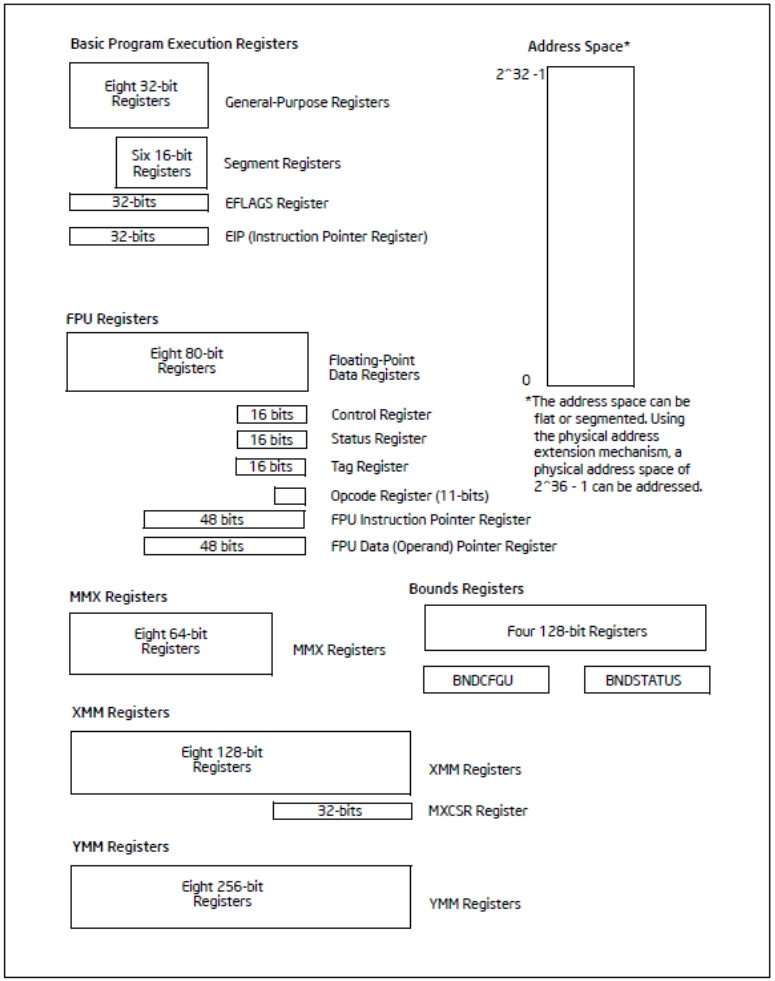
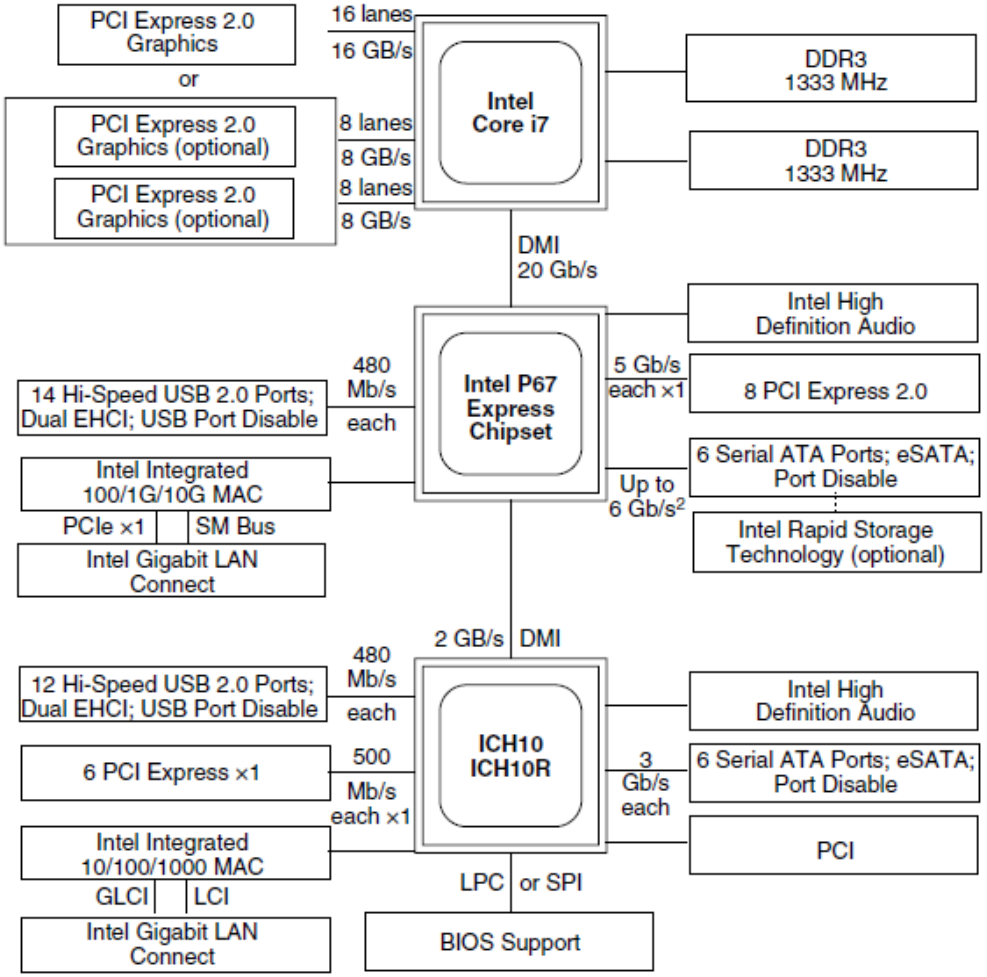
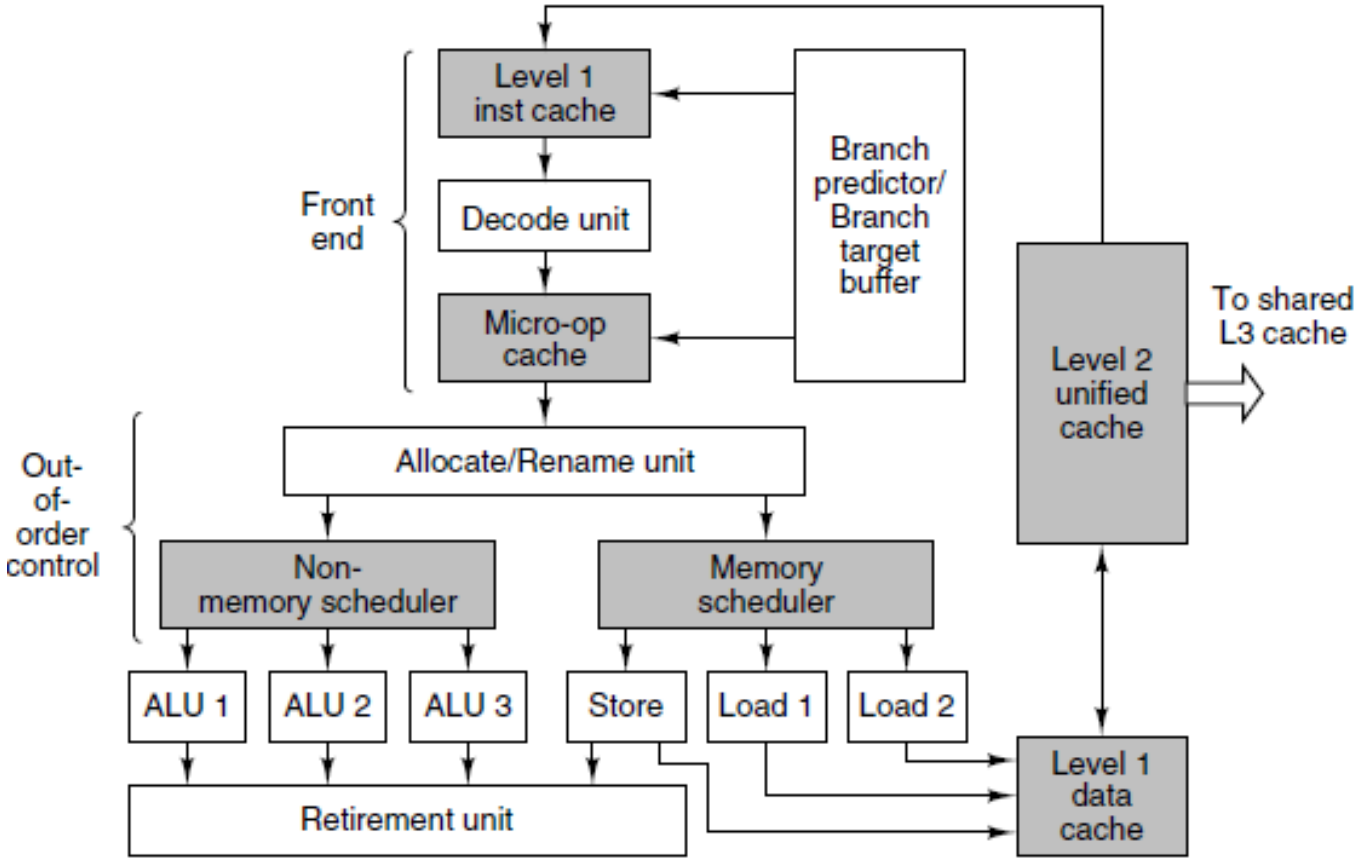


Figure 3-1. IA-32 Basic Execution Environment for Non-64-bit Modes

Структура процесору INTEL 64



Потік даних (процесор INTEL Core i7)



Архітектура INTEL64

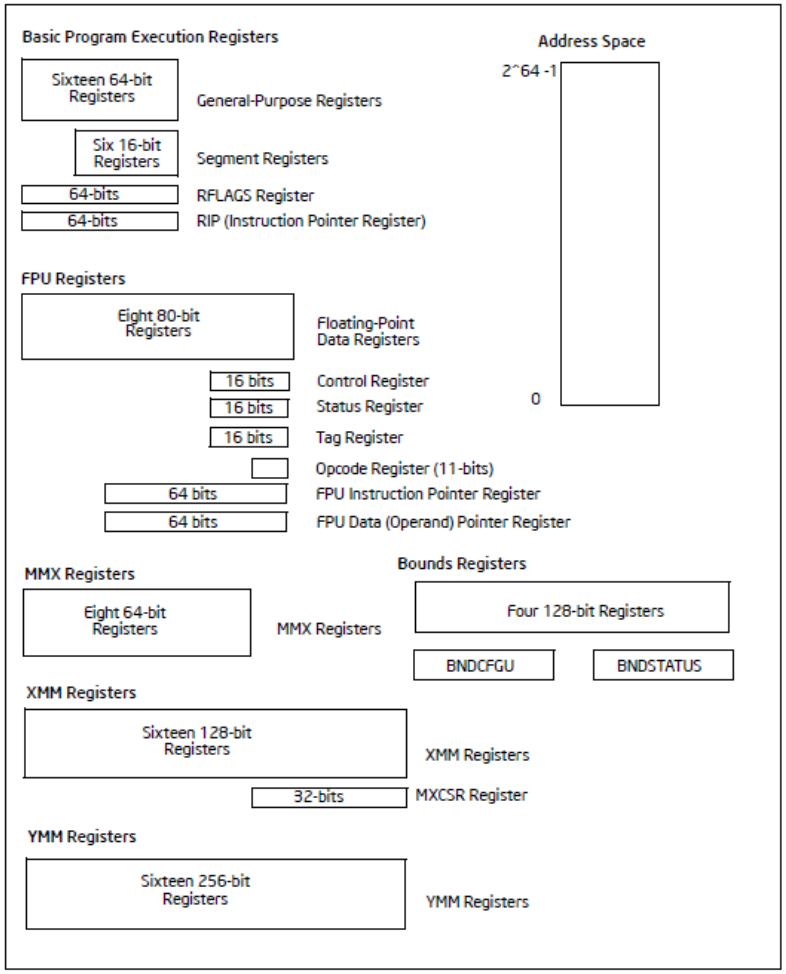
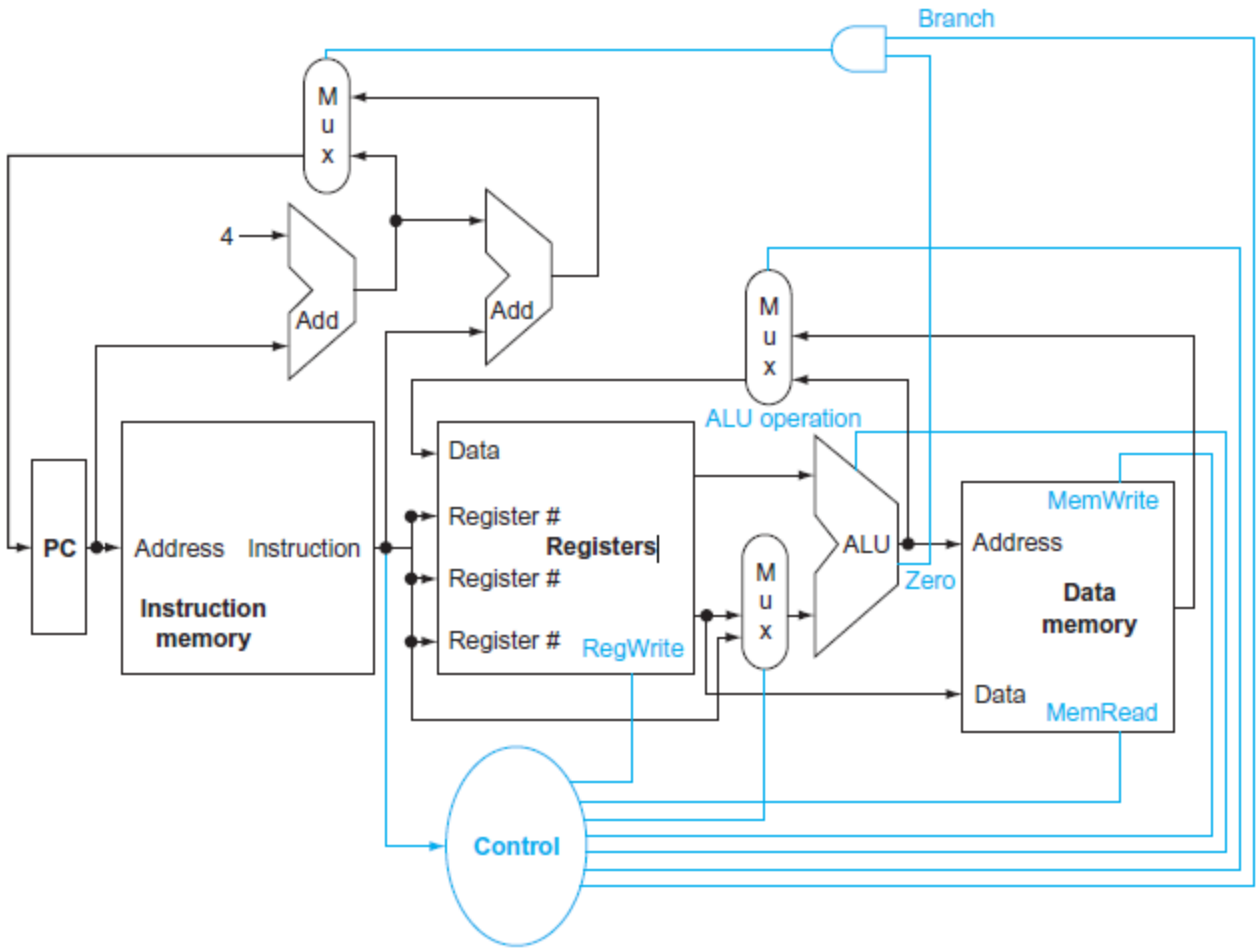
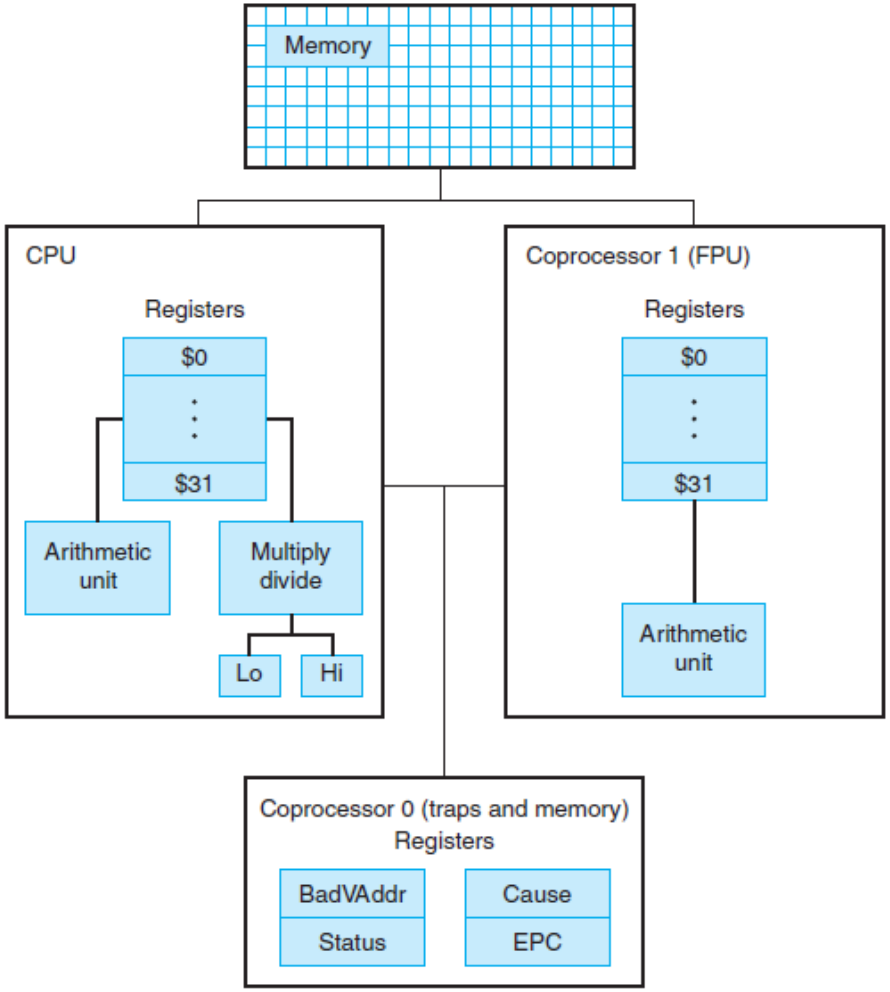


Figure 3-2. 64-Bit Mode Execution Environment

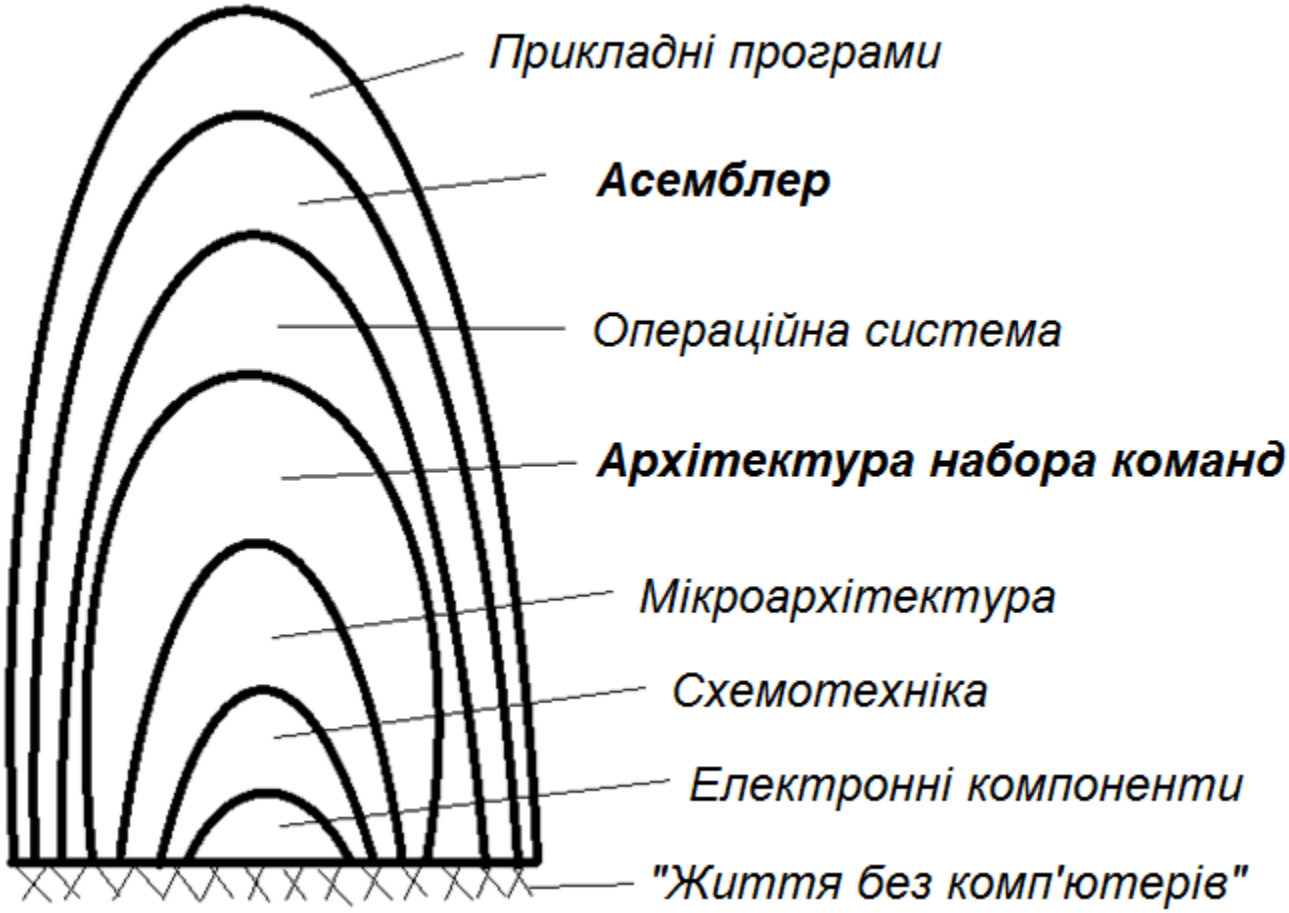
Потік даних (процесор MIPS)



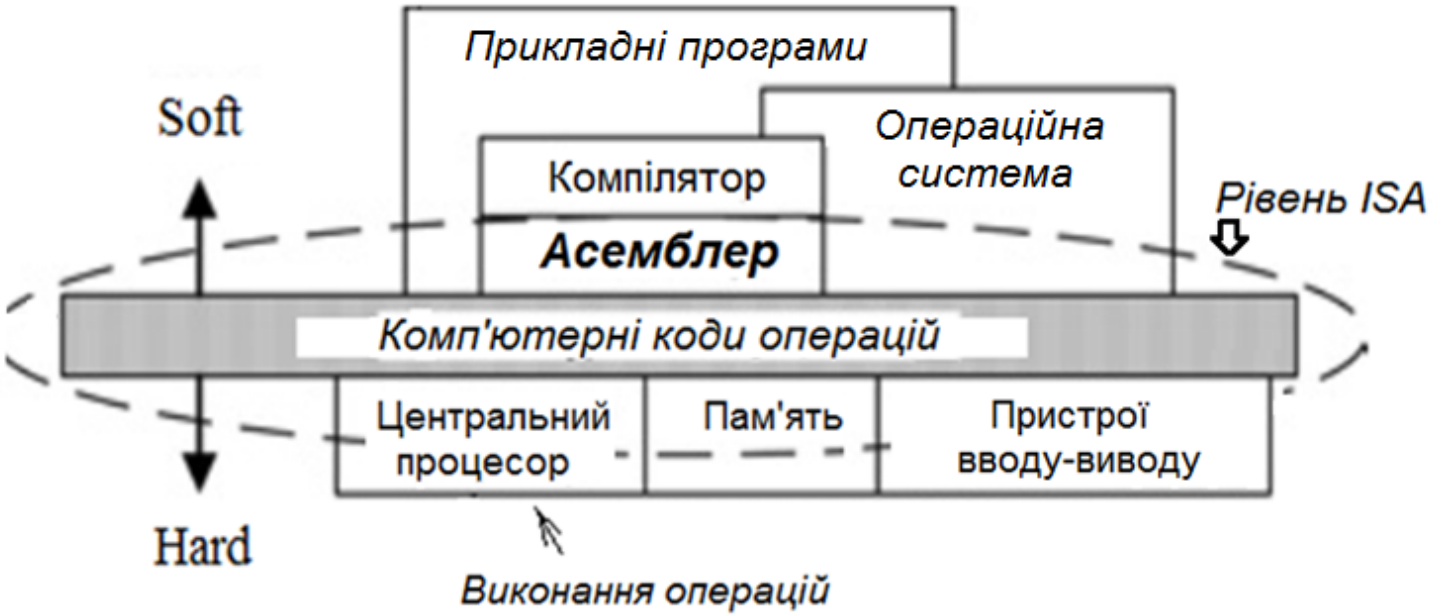
Архітектура MIPS R2000



Багаторівневе представлення комп’ютерів



Структурне розташування рівня ISA



Рівень архітектури набора команд (ISA – Instruction Set Architecture) узгоджує роботу програмного та апаратного забезпечення.

Всі початкові програми транслюються в єдину форму команд рівня ISA, а апаратне забезпечення їх виконує.

Асемблер - програма для перетворення програми з мови асемблера в машинний (комп'ютерний) код

Рівень архітектури набора команд (ISA – Instruction Set Architecture)

Архітектура системи команд (ISA) характеризує:

1. Тип і формати даних.
2. Адресацію даних.
3. Доступ до операндів.
4. Формат команд.
5. Адресацію команд.
6. Кодування команд.
7. Набор команд.

Рівень ISA має опис в технічній документації (IA32, IA64, ARMv7).
Приклади ISA – x86, ARM (Acorn/Advanced RISC Machine), AVR (Alf and Vegard`s RISC processor), MIPS (Microprocessor without Interlocked Pipeline Stages).
Рівень ISA – це рівень, на який орієнтується компілятор при формуванні вихідного коду.

Співвідношення асемблеру x86, мови високого рівня і машинного коду

Приклад коду на C++:

```
int Y;
int X = (Y + 7) * 3;
```

Еквівалентний машинний код:

```
A1 00204000
83C0 07
BB 03000000
F7EB
A3 04204000
8915 08204000
```

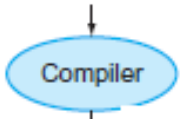
Еквівалент на асемблері:

```
mov eax,Y    ; копіювання із пам'яті значення змінної із ім'ям Y в регістр EAX
             ; (завантаження в регістр EAX двійкового коду відповідно до значення
             ; числа, яке розташоване в пам'яті за адресою відповідно до імені Y)
add eax,7    ; додавання константи 7 до числа, розташованого в регістрі EAX,
             ; та збереження результату в регістрі EAX)
mov ebx,3    ; завантаження числа із значенням 3 в регістр EBX
imul ebx     ; перемноження чисел, розташованих в регістрах EBX і EAX,
              ; та збереження результату (добутку) в регістрі EAX та EDX
mov X,eax     ; завантаження (копіювання в пам'ять) значення числа із регістру EAX
              ; та присвоєння числу імені X
mov X+4,edx  ; завантаження (копіювання в пам'ять) значення числа
              ; із регістру EDX та присвоєння числу імені X+4
```

Співвідношення асемблеру MIPS, мови високого рівня і машинного коду

```
A[300] = h + A[300];
```

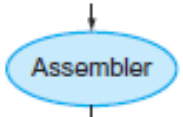
High-level language program (in C)



If `$t1` has the base of the array `A` and `$s2` corresponds to `h`

```
lw   $t0,1200($t1) # Temporary reg $t0 gets A[300]
add  $t0,$s2,$t0   # Temporary reg $t0 gets h + A[300]
sw   $t0,1200($t1) # Stores h + A[300] back into A[300]
```

Assembly language program (for MIPS)



100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Binary machine language program (for MIPS)

Op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

The `lw` instruction is identified by 35 in the first field (op). The base register 9 (`$t1`) is specified in the second field (rs), and the destination register 8 (`$t0`) is specified in the third field (rt). The offset to select `A[300]` ($1200 = 300 \times 4$) is found in the final field (address).

Напрями практичного використання асемблеру

- Вбудовані програми (економічне використання пам'яті).
- Програми реального часу - моделювання та моніторинг стану обладнання (точність запитів і відповідей).
- Комп'ютерні ігрові консолі (оптимізація для невеликого розміру коду і швидкого виконання).
- Аналіз процесів взаємодії комп'ютерних апаратно-програмних засобів («глибина діагностування»).
- Керування елементами комп’ютерного обладнання на самому низькому рівні («маніпулювання бітами»).
- Розробка драйверів (безпосередній доступ до елементів керування).

Початкові вимоги

Базові навички студентів:
- Вміння програмувати хоча б на одній мові високого рівня (Java, C, Python, C ++).
- Знання правил використання операторів, масивів і функцій для вирішення завдань програмування.

Апаратно-програмна підтримка:
комп'ютер з встановленою 32- або 64-розрядною версію Microsoft Windows.

Завдання до самостійної роботи

1. Засвоїти терміни і поняття.
2. Засвоїти головні відмінності між формами подання програмного коду.
3. Визначити ознаки комп’ютерної структури і архітектури для формування системи машинних команд.

Література

1. Intel® 64 and IA-32 Architectures Software Developer's Manual. Order Number: 325462-067US, May 2018
2. Andrew S. Tanenbaum, Todd Austin. Structured Computer Organization. 6th ed. University of Michigan, Ann Arbor, Michigan, United States, 2013
3. David A. Patterson, John L. Hennessy. Computer organization and design: the hardware/software interface. 5th ed. The Morgan Kaufmann series in computer architecture and design, 2014
4. William Stallings. Computer organization and architecture: designing for performance. 10th ed. Pearson Education, Inc, 2016
5. Irvine K.R. Assembly Language for x86 Processors. Florida International University School of Computing and Information Sciences. 7th ed, 2014.
6. Randall Hyde. The Art of Assembly Language. 2th ed, 2010

Регістрова архітектура системи команд

Формат команд

Структура і мікропрограми блоку виконання арифметичних операцій

Огляд IA-32

Формат команд на мові асемблера IA-32

Формат команд

Типова команда має операційну і адресну частини із наступною інформацією:

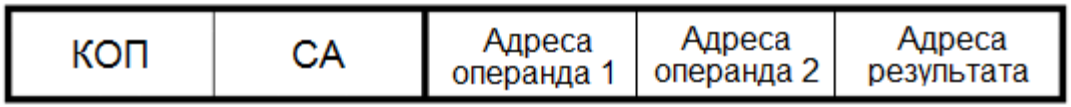
- Дія, яка виконується,
- Місцерозташування операндів,
- Місцерозташування результату.

Формат команд вказує на:

- Тип операцій в системі команд та їх кількість,
- Розрядність команд,
- Тип фрагментів команд («полів команди») та їх розрядність,
- Спосіб декодування команд,
- Кількість адрес в команді («адресність»),
- Способи доступу до даних («способи адресації»).

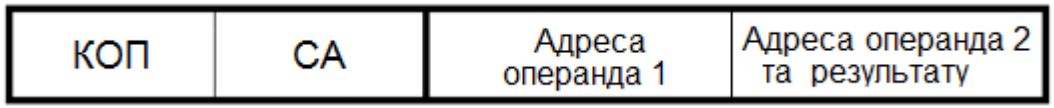
Адресність команд (3А, 2А)

Очевидний варіант адресності – наявність в команді трьох адрес:



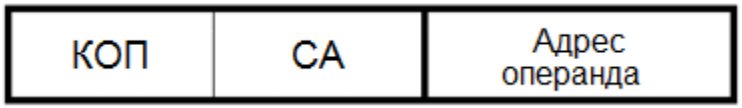
Раніше, наприклад в EDVAC (1952р), в команді була також четверта адреса для вказання номеру наступної команди. Необхідність в 4-й адресі зникла після введення в процесор «покажчика (лічильника) адреси команд» і використання впорядкованого розташування команд в пам’яті.

Використання адреси одного із операндів (наприклад, другого) в якості адреси результату дозволяє перейти до 2-адресних команд (після виконання команди операнд втрачений, тому що дані по відповідній адресі заміщуються на результат):



Адресність команд (1А, 0А)

Одноадресні команди приписують розташування одного із операндів та результату за фіксованою адресою (зазвичай, в одному із процесорних регістрів - «акумуляторі»):



Безадресні («нульадресні») команди приписують використання даних за фіксованими адресами розташування операндів та результату:

Критерії визначення адресності команд:
- Ємність пам’яті для розташування даних,
- Швидкодія виконання програми,
- Ефективність використання комірок пам’яті.

Приклад оцінки ефективності команд різної адресності

Необхідно обчислити: ***y*** = ***a*** x ***b*** + ***(c – d)*** x ***e*** / ***f***.

Для оцінки використовуються коефіцієнти:

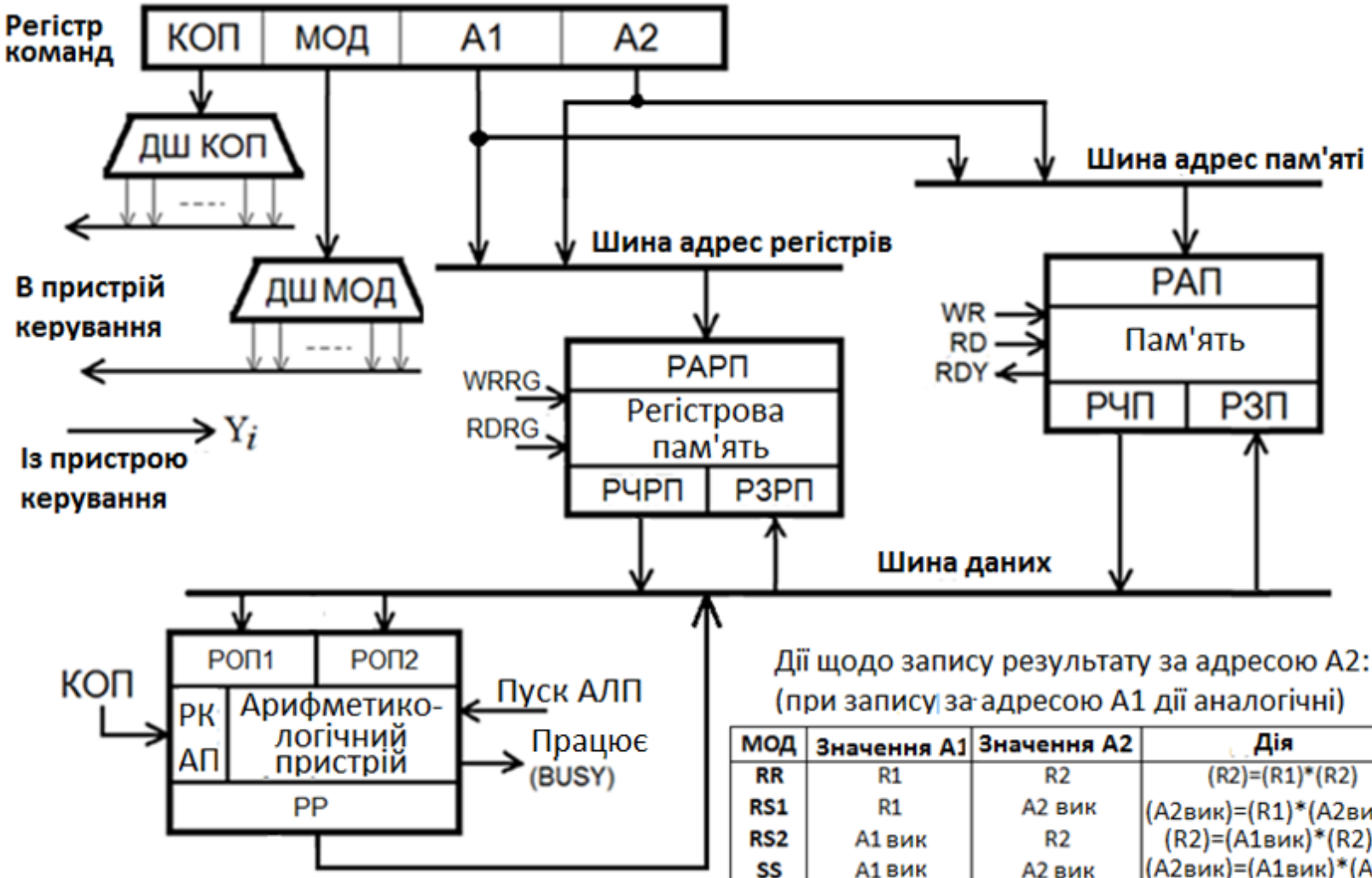
1. Використання адрес в команді: K = Ae/As (Ae – кількість адрес, що вказані в програмі, As – кількість адресних полів у всіх командах програми).
2. Звертань до пам'яті: T = Top + Tin (Top – кількість звертань до пам'яті для передавання даних, Tin - … команд).

Для прикладу: K3=9/15, K2=9/12, K1=9/9, T3=14(9+5), T2=15(9+6), T1=18(9+9).

В загальному випадку, ефективність залежить від прикладної області.

А) Команди 3А	Б) Команди 2А	В) Команди 1А
(***s*** - адреса комірки пам'яті, ***Acc*** - акумулятор):		
МН ***a, b, s ; a*** x ***b*** -> ***s***	МН ***a, b ; a*** x ***b*** -> ***Acc***	ЧТ ***a ; a*** -> ***Acc***
ВД ***c, d, - ; c – d*** -> ***Acc***	ЗП ***s, - ; Acc*** -> ***s***	МН ***b ; Acc*** x ***b*** -> ***Acc***
МН ***- , e, - ; Acc*** x ***e*** -> ***Acc***	ВД ***c, d ; c – d*** -> ***Acc***	ЗП ***s ; Acc*** -> ***s***
ДЛ ***- , f , - ; Acc / f*** -> ***Acc***	МН ***e, - ; Acc*** x ***e*** -> ***Acc***	ЧТ ***c ; c*** -> ***Acc***
ДД ***- , s, y ; Acc + s*** -> ***y***	ДЛ ***f , - ; Acc / f*** -> ***Acc***	ВД ***d ; Acc - d*** -> ***Acc***
	ДД ***s, y ; Acc + s*** -> ***y***	МН ***e ; Acc*** x ***e*** -> ***Acc***
		ДЛ ***f ; Acc / f*** -> ***Acc***
		ДД ***s ; Acc + s*** -> ***Acc***
		ЗП ***y ; Acc*** -> ***y***

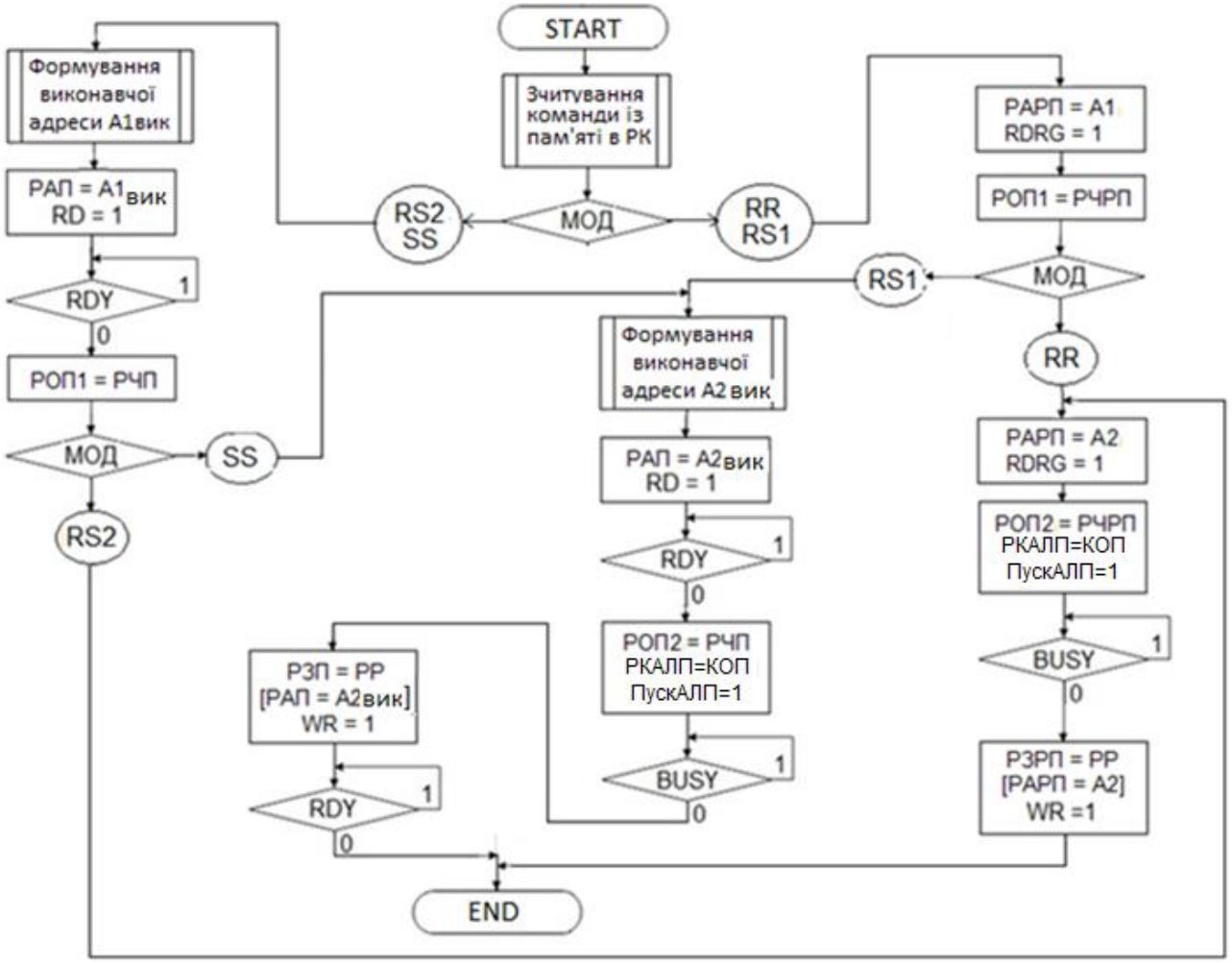
Структура блоку виконання арифметичних операцій адресності «2А» (із регістровою пам’яттю)



Дії щодо запису результату за адресою А2:
(при запису за адресою А1 дії аналогічні)

МОД	Значення А1	Значення А2	Дія
RR	R1	R2	(R2)=(R1)*(R2)
RS1	R1	А2 вик	(А2вик)=(R1)*(А2вик)
RS2	А1 вик	R2	(R2)=(А1вик)*(R2)
SS	А1 вик	А2 вик	(А2вик)=(А1вик)*(А2вик)

Мікропрограма виконання арифметичних операцій адресності «2А» (із регістровою пам’яттю)

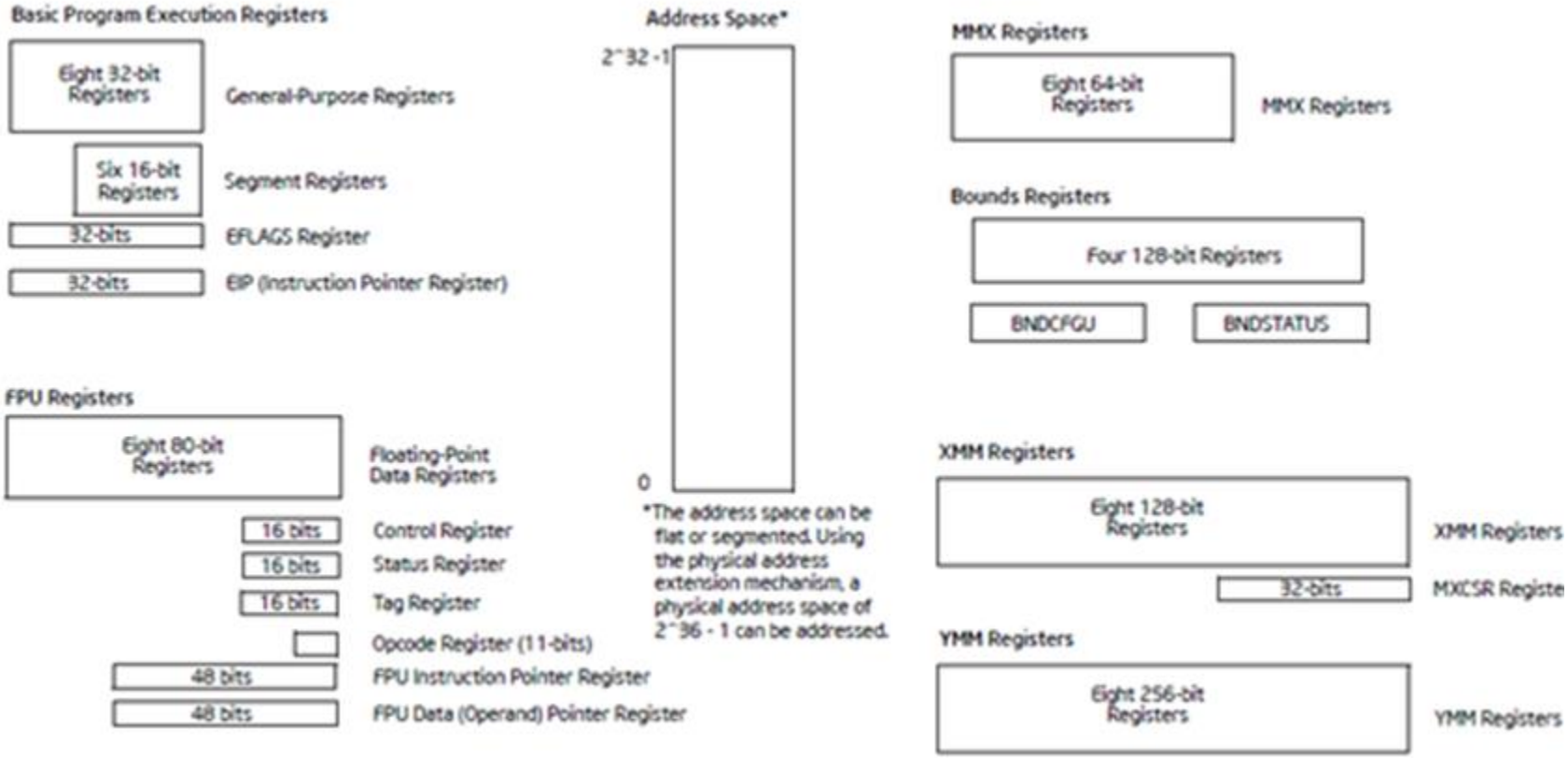


Дії блоку виконання арифметичних операцій адресності «2А»

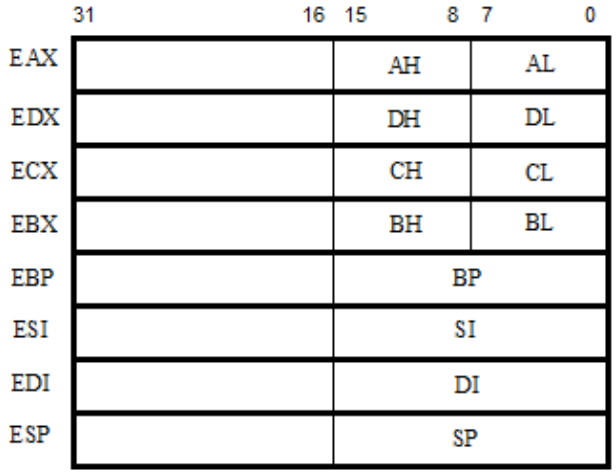
-Виявлення місцерозташування першого операнда,
-Формування виконавчої адреси (якщо перший операнд розташований в загальній пам’яті),
-Зчитування першого операнда (із регістра або пам’яті),
-Виявлення місцерозташування другого операнда,
-Формування виконавчої адреси (якщо другий операнд розташований в загальній пам’яті),
-Зчитування другого операнда (із регістра або пам’яті),
-Пересилання операндів в АЛП,
-Виконання операції в АЛП,
-Виявлення місцерозташування результату,
-Запис (занесення) результату (в регістр або пам’ять).

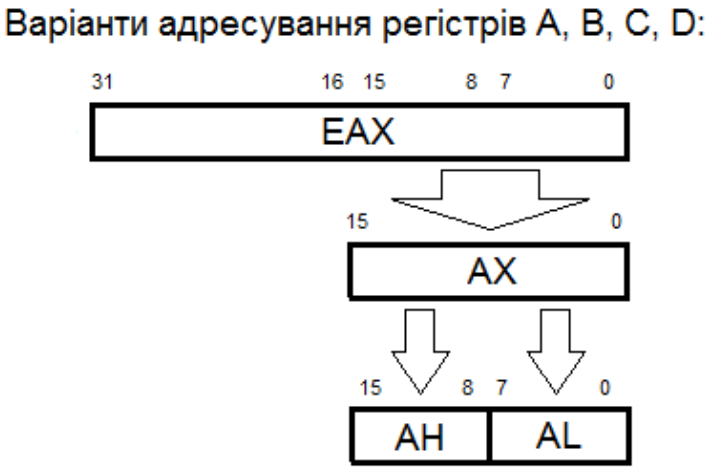
Виконавчий (ефективний) адрес – двійковий код номеру комірки пам’яті (що вказана в команді) для збереження операнда та/або результату.

Архітектура IA-32



Регістри загального призначення (General Purpose Registers)



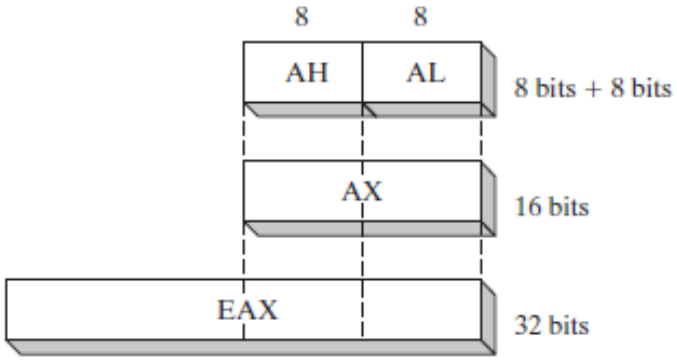


Використання регістрів в деяких командах (за умовчанням):

- EAX - акумулятор, джерело операнду або приймач результату;
- EBX - покажчик на дані в сегменті даних (DS);
- ECX - лічильник в рядкових (ланцюгових (MOVS)) і циклічних (із префіксом REP) командах;
- EDX - частина даних в арифметичних діях, адреса порту вводу-виводу в IN/INS, OUT/OUTS;
- ESI - покажчик на джерело операнду (індексний регістр джерела даних);
- EDI - покажчик на приймач операнду (індексний регістр приймача даних);
- EBP - покажчик на фрагмент даних в сегменті стеку (SS).

Регістр ESP завжди вказує на відносний адрес вершини стеку.

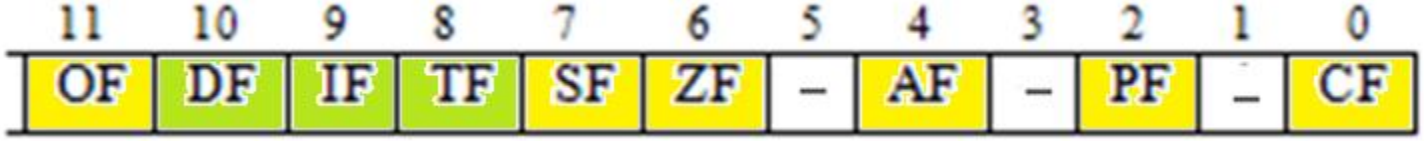
Розрядність і позначення регістрів



32-Bit	16-Bit	8-Bit (High)	8-Bit (Low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

32-Bit	16-Bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

Регістр ознак/прапорів (EFLAGS Register) (12 молодших розрядів)

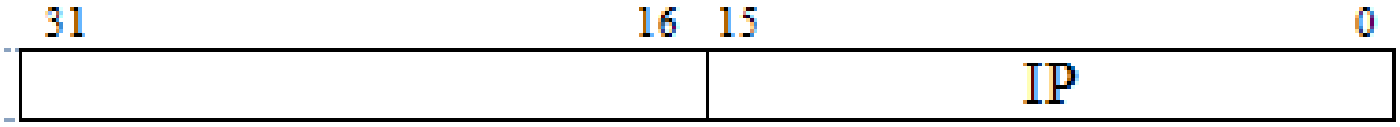


<u>Шість</u> ознак формуються відповідно до результату виконання операції:
- **CF** (carry) - перенесення із старшого розряду результату (із розряду (**n-1**) в неіснуючий розряд **n)**;
- **PF** (parity) – парний результат (кількість одиниць в молодшому байті результату є парним);
- **AF** (auxiliary carry) – перенесення між тетрадами молодшого байту (із 3 в 4 розряд результату);
- **ZF** (zero) - нульовий результат (всі розряди результату нульові);
- **SF** (sign) – знак результату (значення старшого розряду (**n-1**) результату);
- **OF** (overflow) - переповнення (перенесення із розряду (n-2) в розряд (n-1) не співпадає із перенесенням із розряду (n-1) в неіснуючий розряд n).

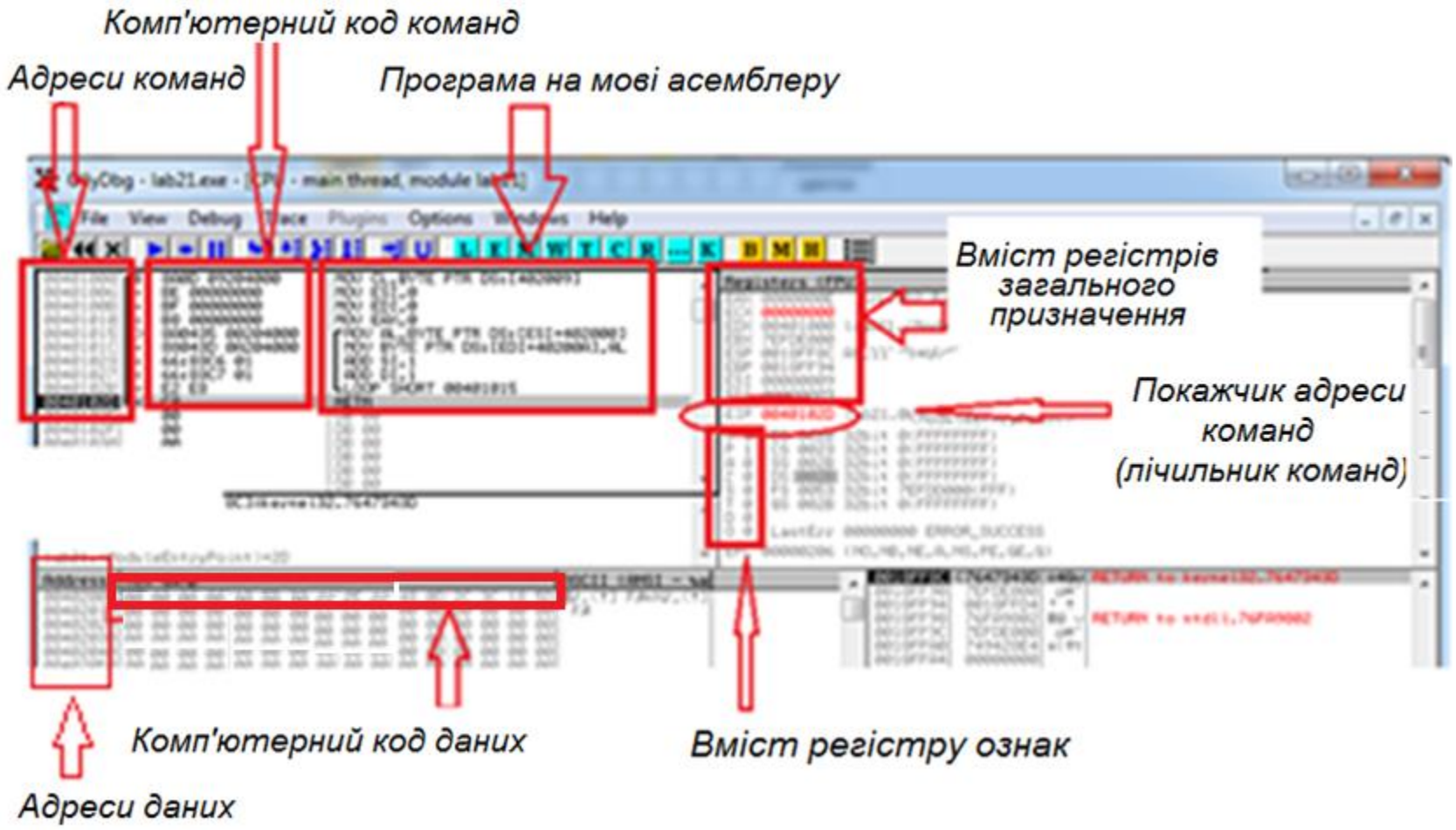
<u>Три</u> ознаки вказують на стан роботи процесору:
- **DF** (direction) - напрям автоадресації джерела/приймача в рядкових (ланцюгових) командах (DF = 0 - автоінкремент, DF = 1 - автодекремент);
- **IF** (interrupt enable) – наявність дозволу на переривання (IF =1 – дозволено);
- **TF** (trap) – наявність дозволу крокового (одна команда) режиму (TF=1 - дозволено).

Регістр адреси команд «покажчик команд» - (EIP)

Покажчик команд (EIP) - 32-розрядний регістр – адреса *наступної* команди на виконання.



Відображення архітектурних елементів в OllyDbg та x32Dbg



Формат команд на мові асемблеру

```
[label:] mnemonic [operands] [;comment]
```

- Мітка (Label) - починається з літери і завершується двокрапкою.
  Приклади - mit18:, ab55:, fah2:

- Мнемокод операції (Instruction mnemonic) – відділяється пробілом.
  Приклади - ADD, MOV, SUB, MUL, JMP

- Операнд/операнди (Operand(s)) – ім’я регістрів, мітки, безпосередні дані, вирази арифметичні, вирази для формування адрес операндів в пам’яті тощо.

  Приклади – eax, 52, 0A4h, [(18+20)/2], [ebx+esi]

- Коментар (Comment) - починається крапкою з комою.

Обов’язковим є вказання на мнемокод операції.

Правила вказання на операнди і результат

В командах на мові асемблеру вказання на послідовність операндів і результату обумовлено за умовчанням:

ADD dst, src

dst (destination) – приймач (приемник (рос))

src (source) – джерело (подавач?) (источник (рос))

Результат завантажується в ***dst***, наприклад, для операції:

- ***ADD***: ***dst = dst + src***
- ***SUB***: ***dst = dst - src***

Така погодженість присутня в більшості команд.

По іншому, наприклад, в рядкових (ланцюгових) командах.

Вимога – однакова розрядність операндів команди

Нотація в командах

Symbol	Description
reg	An 8-, 16-, or 32-bit general register from the following list: AH, AL, BH, BL, CH, CL, DH, DL, AX, BX, CX, DX, SI, DI, BP, SP, EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP.
reg8, reg16, reg32	A general register, identified by its number of bits.
segreg	A 16-bit segment register (CS, DS, ES, SS, FS, GS).
accum	AL, AX, or EAX.
mem	A memory operand, using any of the standard memory-addressing modes.
mem8, mem16, mem32	A memory operand, identified by its number of bits.
shortlabel	A location in the code segment within −128 to +127 bytes of the current location.
nearlabel	A location in the current code segment, identified by a label.
farlabel	A location in an external code segment, identified by a label.
imm	An immediate operand.
imm8, imm16, imm32	An immediate operand, identified by its number of bits.

Команди додавання та віднімання (основні)

ADD (Added) - Додавання

Формат:

```
ADD   reg,reg      ADD   reg,imm
ADD   mem,reg      ADD   mem,imm
ADD   reg,mem      ADD   accum,imm
```

Додає операнд із джерела до операнду із приймача і завантажує результат в приймач.

Розрядності операндів повинні бути однаковими.

A source operand is added to a destination operand, and the sum is stored in the destination.
Operands must be the same size.

Формування ознак:

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

SUB (Subtract) - Віднімання

Формат:

```
SUB   reg,reg      SUB   reg,imm
SUB   mem,reg      SUB   mem,imm
SUB   reg,mem      SUB   accum,imm
```

Віднімає значення операнда джерела із операнда приймача і завантажує результат в приймач.

Розрядності операндів повинні бути однаковими.

Subtracts the source operand from the destination operand.
Operands must be the same size.

Формування ознак:

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

Пересилання (копіювання) даних

MOV (Move) - Пересилання

(Копіювання)

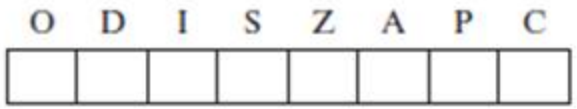
Формат:

```
MOV reg,reg          MOV reg,imm
MOV mem,reg          MOV mem,imm
MOV reg,mem          MOV mem16,segreg
MOV reg16,segreg     MOV segreg,mem16
MOV segreg,reg16
```

Копіювання байт/слово/подвійне слово із джерела в приймач

Copies a byte or word from a source operand to a destination operand.

Формування ознак немає:



Операція комп’ютерної логіки - XOR

XOR — **Exclusive OR**

O	D	I	S	Z	A	P	C
0			*	*	?	*	0

Виключне ЧИ (виключне АБО)

Each bit in the source operand is exclusive ORed with its corresponding bit in the destination. The destination bit is a 1 only when the original source and destination bits are different.

Instruction formats:

```
XOR   reg,reg      XOR   reg,imm
XOR   mem,reg      XOR   mem,imm
XOR   reg,mem      XOR   accum,imm
```

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Використання для обнулення регістрів: XOR *reg, reg*

Приклад (обнулення регістру EAX): `XOR EAX,EAX`

(комп код команди: `33C0` *)*

Альтернативні варіанти обнулення:

```
MOV EAX,0     |  B8 00000000
SUB EAX,EAX   |  2BC0
```

Приклади із 32-біт і 8-біт даними

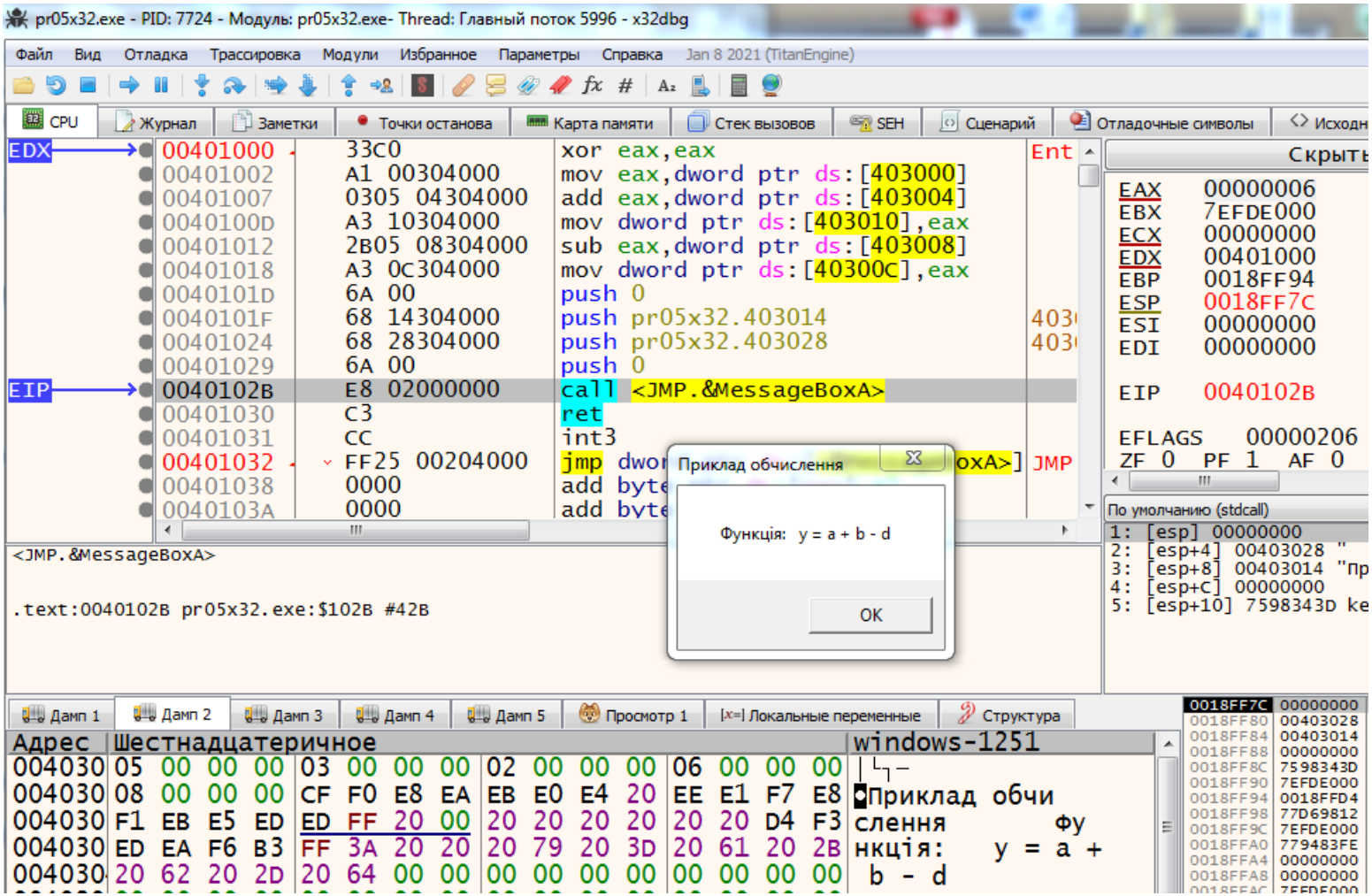
(pr05x32.asm)

```
TITLE < y=a+b-d >
.686
.model flat, stdcall
option casemap: none
.data
   a dd 5
   b dd 3
   d dd 2
   y dd 0Fh
   temp dd ?
.code
sty:
   xor    eax, eax
   mov eax, a
   add eax, b
   mov temp, eax  ; (a+b)
   sub eax, d
   mov y, eax  ; y=(a+b-d)
ret
end sty
```

(pr06x32.asm)

```
TITLE <y=a+b-d>
.686
.model flat, stdcall
option casemap: none
.data
   a db 5
   b db 3
   d db 2
   y db 0Fh
   temp db ?
.code
sty:
  xor    eax, eax
   mov al, a
   add al, b
   mov  temp, al  ; (a+b)
   sub al, d
   mov y, al  ; y=(a+b-d)
ret
end sty
```

Приклад в x32Dbg



Завдання до самостійної роботи (звіт не вимагається)

1. Особливості формату команд різної адресності.
2. Положення з регістрової архітектури команд.
3. Алгоритм виконання арифметичних операцій і операцій пересилання адресності «2А» в архітектурі із регістровою пам’яттю.
4. Архітектура IA-32.
5. Правила вказання на розрядність компонентів IA-32 в командах на мові асемблера.
6. Формат команд на мові асемблера.
7. Редагувати програму, що наведена на попередньому слайді, з іншими значеннями операндів та проаналізувати вміст регістрів і комірок пам’яті.

Література

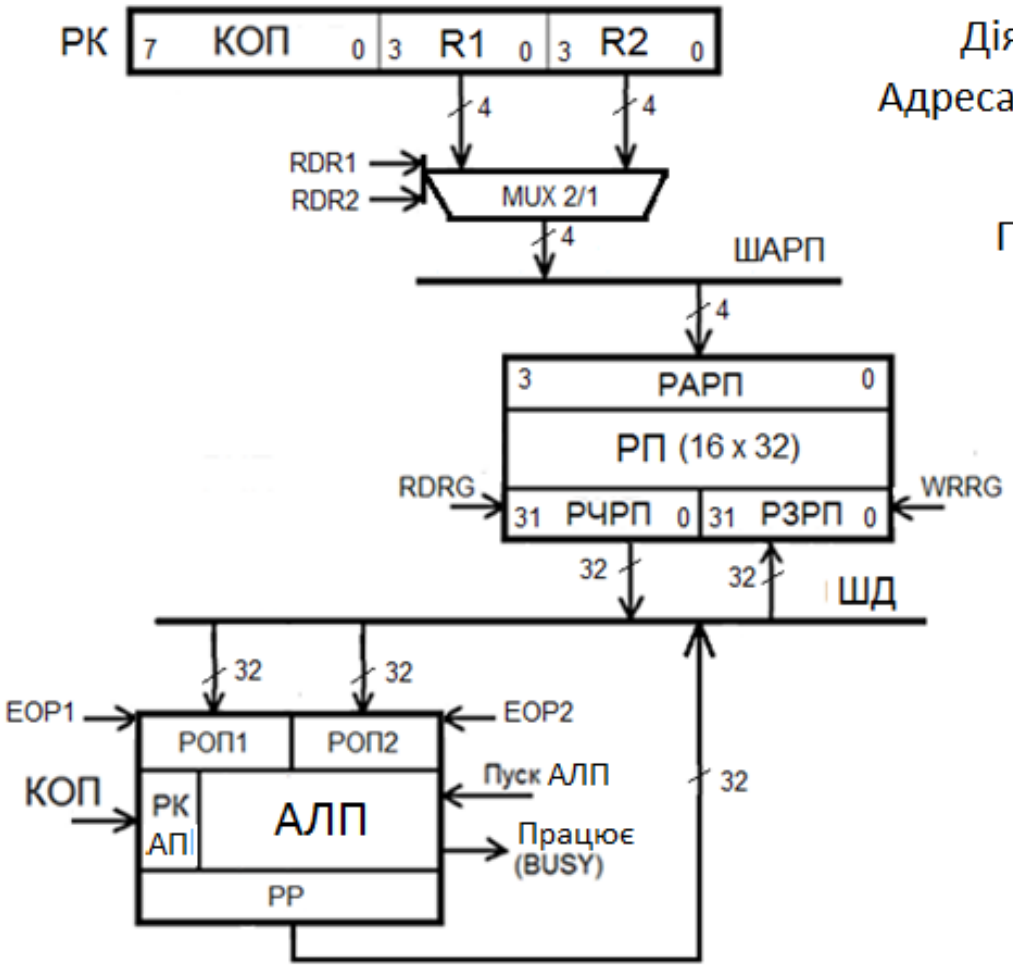
1. Intel® 64 and IA-32 Architectures Software Developer's Manual. - Order Number: 325462-067US, May 2018 (ch.4)
2. Kip R. Irvine. Assembly Language for x86 Processors. Florida International University School of Computing and Information Sciences. 7th Edition, 2014 (ch.2-3,app.A)
3. Andrew S. Tanenbaum, Todd Austin. Structured Computer Organization. – University of Michigan, Ann Arbor, Michigan, United States, 2013 (ch.5)
4. Бабич М.П., Жуков І.А. Комп'ютерна схемотехніка: Навчальний посібник. – К.: «МК-Прес», 2004 (гл.10, с.337-340)

Додаткове завдання до самостійної роботи (не є обов’язковим для виконання)

1. Засвоїти алгоритм виконання арифметичних операцій адресності «2А» із регістровою пам’яттю при форматі команд “RR”, “RS”, “RX” (наступні слайди).
2. Вказати на особливості мікропрограми виконання арифметичних операцій адресності «2А» із регістровою пам’яттю при форматі команд “RR”, “RS”, “RX”.

Barbara J. Burian. A simple approach to S/370 assembly language programming. — New Jersy: Prentice-Hall, Inc, 1977.

Приклад виконання арифметичних операцій адресності «2А» із регістровою пам’яттю
Формат команди “RR”



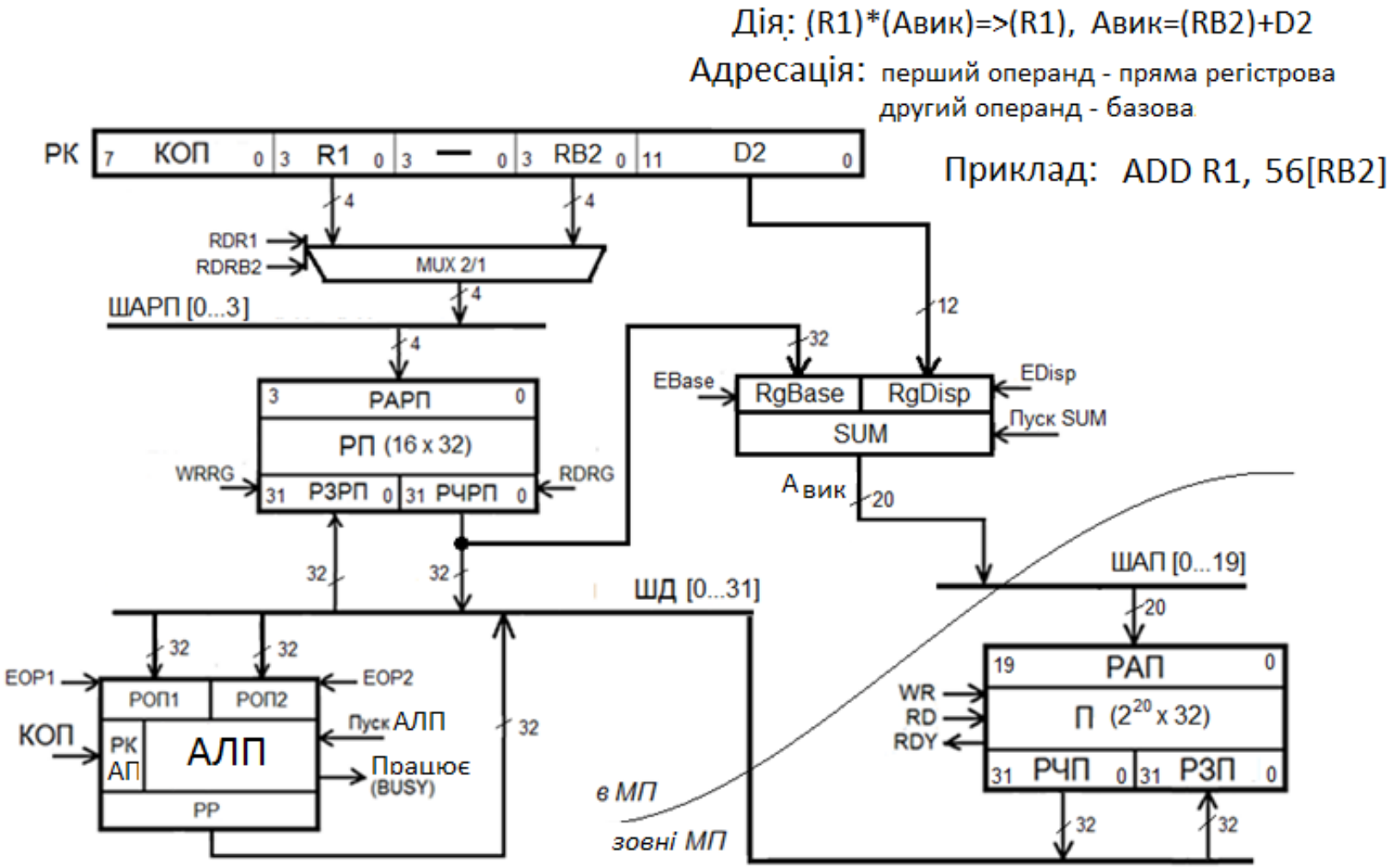
Дія: (R1) * (R2) => (R1)

Адресація: пряма регістрова

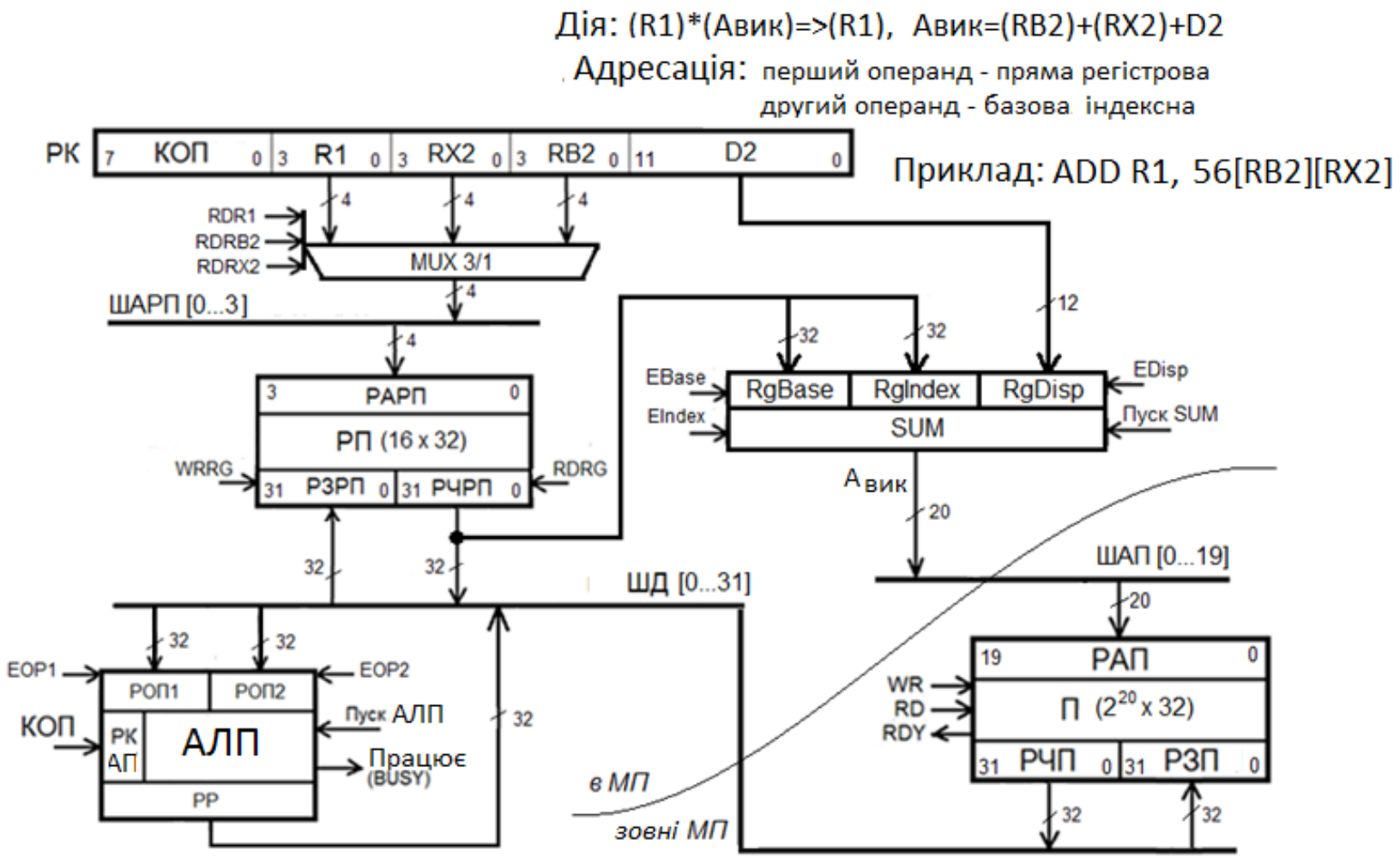
Приклад: ADD R1, R2

Приклад виконання арифметичних операцій адресності «2А» із регістровою пам’яттю

Формат команди “RS”



Приклад виконання арифметичних операцій адресності «2А» із регістровою пам’яттю
Формат команди “RX”



Команди асемблеру процесорів «IA-32»

Інкремент, декремент,
Логічні
Перекодування
Зсув
Цикл
Умовний перехід
Порівняння
Опрацювання окремих бітів

Архітектура IA-32

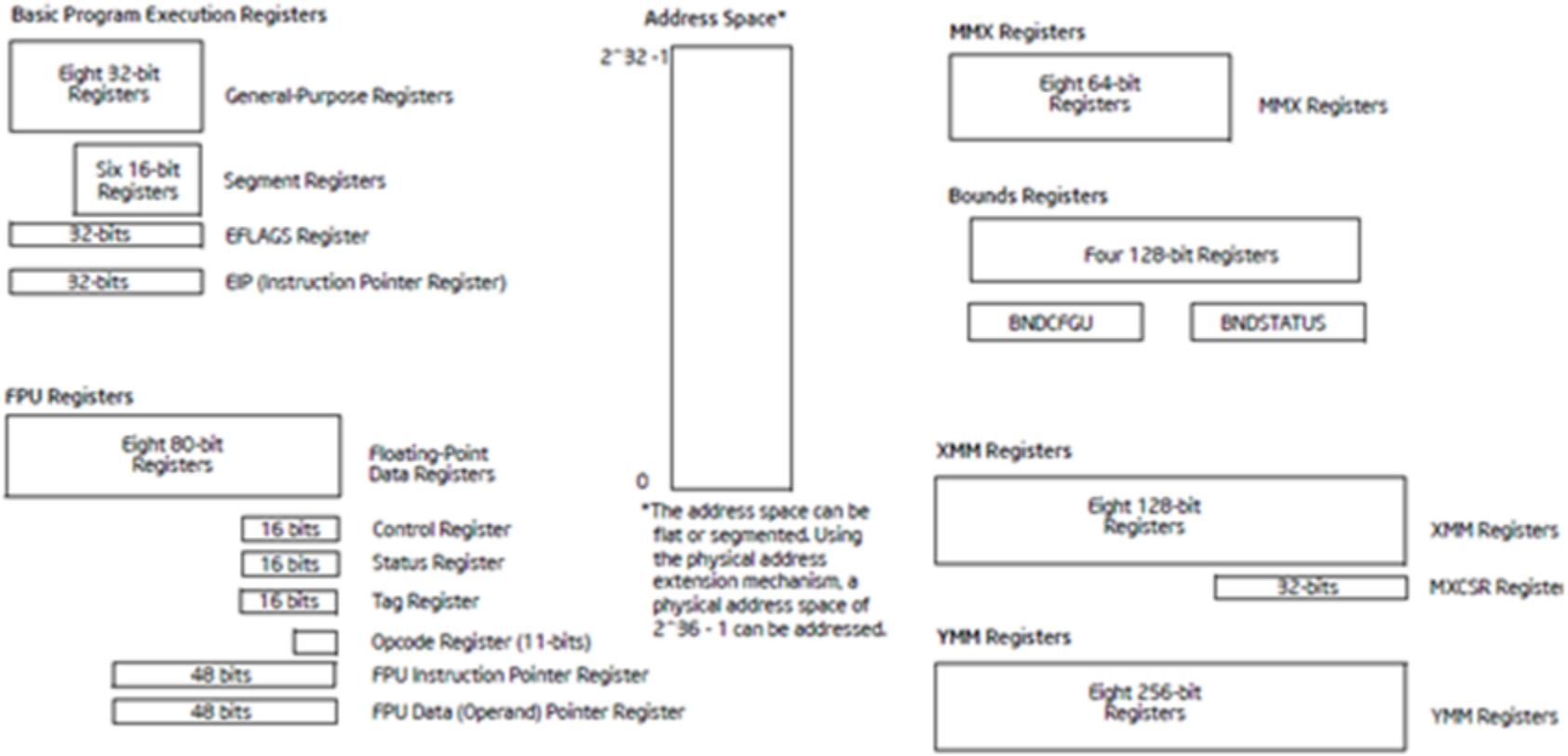


Figure 3-1. IA-32 Basic Execution Environment for Non-64-bit Modes

Зміна значення операнда на одиницю

INC (Increment) - Збільшення на одиницю

Додавання 1 до операнду.

Adds 1 to a register or memory operand.

Формат:
```
INC  reg
INC  mem
```

Формування ознак (ознака CF не формується)

O	D	I	S	Z	A	P	C
*			*	*	*	*	

DEC (Decrement) - Зменшення на одиницю

Віднімання 1 із операнду.

Subtracts 1 from an operand. Does not affect the Carry flag.

Формат:
```
DEC  reg
DEC  mem
```

Формування ознак (ознака CF не формується)

O	D	I	S	Z	A	P	C
*			*	*	*	*	

Операції комп’ютерної логіки

AND — Logical AND

O	D	I	S	Z	A	P	C
*			*	*	?	*	0

Each bit in the destination operand is ANDed with the corresponding bit in the source operand.
Instruction formats:

```
AND   reg,reg
AND   mem,reg
AND   reg,mem
AND   reg,imm
AND   mem,imm
AND   accum,imm
```

OR — Inclusive OR

O	D	I	S	Z	A	P	C
0			*	*	?	*	0

Performs a boolean (bitwise) OR operation between each matching bit in the destination operand and each bit in the source operand.
Instruction formats:

```
OR   reg,reg
OR   mem,reg
OR   reg,mem
OR   reg,imm
OR   mem,imm
OR   accum,imm
```

XOR — Exclusive OR

O	D	I	S	Z	A	P	C
0			*	*	?	*	0

Each bit in the source operand is exclusive ORed with its corresponding bit in the destination. The destination bit is a 1 only when the original source and destination bits are different.
Instruction formats:

```
XOR   reg,reg      XOR   reg,imm
XOR   mem,reg      XOR   mem,imm
XOR   reg,mem      XOR   accum,imm
```

Перетворення кодів

NEG — Negate

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

Calculates the twos complement of the destination operand and stores the result in the destination.
Instruction formats:

`NEG` *reg* `NEG` *mem*

NOT — Not

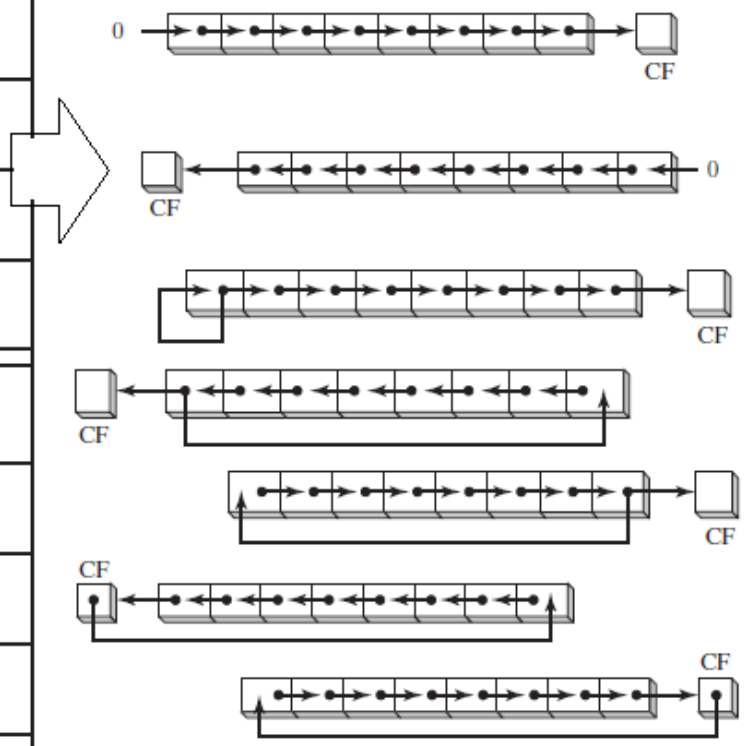
O	D	I	S	Z	A	P	C

Performs a logical NOT operation on an operand by reversing each of its bits.
Instruction formats:

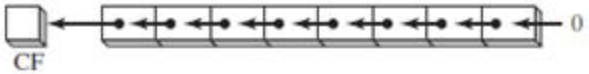
`NOT` *reg* `NOT` *mem*

Операції зсуву

SHR	Shift right
SHL	Shift left
SAL	Shift arithmetic left
SAR	Shift arithmetic right
ROL	Rotate left
ROR	Rotate right
RCL	Rotate carry left
RCR	Rotate carry right
SHLD	Double-precision shift left
SHRD	Double-precision shift right



Зсув вліво



SHL
SAL

Shift Left

Shift Arithmetic Left

`SHL reg,imm8`	`SAL reg,imm8`
`SHL reg,CL`	`SAL reg,CL`
`SHL mem,imm8`	`SAL mem,imm8`
`SHL mem,CL`	`SAL mem,CL`

Shifts each bit in the destination operand to the left, using the source operand to determine the number of shifts. The highest bit is copied into the Carry flag, and the lowest bit is filled with a zero (identical to SAL). The *imm8* operand must be a 1 when using the 8086/8088 processor.

O	D	I	S	Z	A	P	C
*			*	*	?	*	*

Можливе застосування - множення на значення, кратне степіню 2.

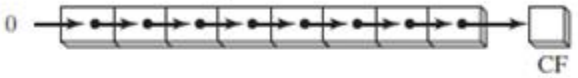
```
mov  dl,10     ; before:  00001010
shl  dl,2      ; after:   00101000
```

Зсув вправо

SHR — **Shift Right**

Shifts each bit in the destination operand to the right, using the source operand to determine the number of shifts. The highest bit is filled with a zero, and the lowest bit is copied into the Carry flag. The *imm8* operand must be a 1 when using the 8086/8088 processor.

```
SHR   reg, imm8      SHR   mem, imm8
SHR   reg,CL         SHR   mem,CL
```



O	D	I	S	Z	A	P	C
*			*	*	?	*	*

SAR — **Shift Arithmetic Right**

Shifts each bit in the destination operand to the right, using the source operand to determine the number of shifts. The lowest bit is copied into the Carry flag, and the highest bit retains its previous value. This shift is often used with signed operands because it preserves the number's sign. The *imm8* operand must be a 1 when using the 8086/8088 processor.

```
SAR   reg, imm8      SAR   mem, imm8
SAR   reg,CL         SAR   mem,CL
```

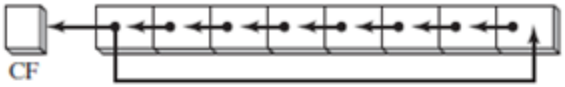
Можливе застосування - ділення на значення, кратне степіню 2.

```
a) mov  al,0F0h          ; AL = 11110000b (-16)
   sar  al,1             ; AL = 11111000b (-8), CF = 0

b) mov  al,01000000b     ; AL = 64
   shr  al,3             ; divide by 8, AL = 00001000b

c) mov  ax,-128          ; EAX = ????FF80h
   shl  eax,16           ; EAX = FF800000h
   sar  eax,16           ; EAX = FFFFFF80h
```

Приклади застосування команди ROL



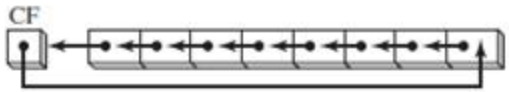
а) Зсуви на тетраду з відтворенням початкового значення:

```
mov  ax,6A4Bh
rol  ax,4          ; AX = A4B6h
rol  ax,4          ; AX = 4B6Ah
rol  ax,4          ; AX = B6A4h
rol  ax,4          ; AX = 6A4Bh
```

б) Перетворення 26h в 62h:

```
mov  al,26h
rol  al,4        ; AL = 62h
```

Приклад застосування команди RCL



Перевірка біту LSB (контроль числа на парність):

```
.data
testval BYTE  01101010b
.code
shr  testval,1          ; shift LSB into Carry flag
jc   exit               ; exit if Carry flag set
rcl  testval,1          ; else restore the number
```

Подвійний зсув вліво SHLD

Операнд-отримувач зміщується вліво на задану кількість бітів. Позиції бітів, відкриті зсувом, заповнюються найбільш значущими бітами операнда-джерела. Значення операнда-джерела не змінюється.

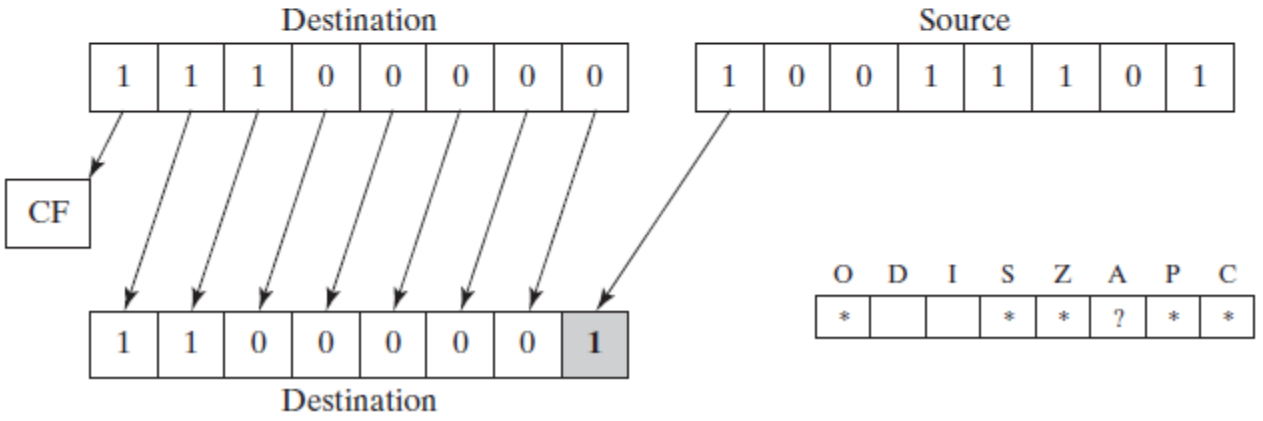
SHLD — **Double-Precision Shift Left (x86)**

Shifts the bits of the second operand into the first operand. The third operand indicates the number of bits to be shifted. The positions opened by the shift are filled by the most significant bits of the second operand. The second operand must always be a register, and the third operand may be either an immediate value or the CL register.

Instruction formats:

```
SHLD dest, source, count

SHLD reg16,reg16,imm8        SHLD mem16,reg16,imm8
SHLD reg32,reg32,imm8        SHLD mem32,reg32,imm8
SHLD reg16,reg16,CL          SHLD mem16,reg16,CL
SHLD reg32,reg32,CL          SHLD mem32,reg32,CL
```



Подвійний зсув вправо SHRD

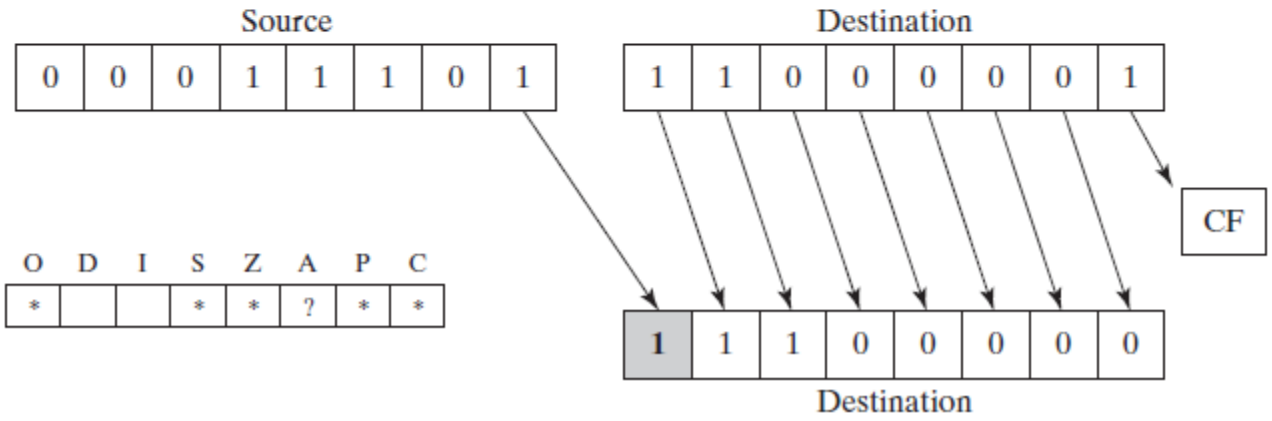
Операнд-отримувач зміщується вправо на задану кількість бітів. Позиції бітів, відкриті зсувом, заповнюються найменш значущими бітами операнда-джерела. Значення операнда-джерела не змінюється.

SHRD — **Double-Precision Shift Right (x86)**

Shifts the bits of the second operand into the first operand. The third operand indicates the number of bits to be shifted. The positions opened by the shift are filled by the least significant bits of the second operand. The second operand must always be a register, and the third operand may be either an immediate value or the CL register.

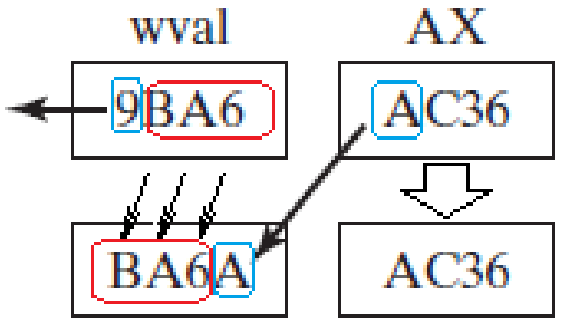
Instruction formats: `SHRD dest, source, count`

```
SHRD reg16,reg16,imm8        SHRD mem16,reg16,imm8
SHRD reg32,reg32,imm8        SHRD mem32,reg32,imm8
SHRD reg16,reg16,CL          SHRD mem16,reg16,CL
SHRD reg32,reg32,CL          SHRD mem32,reg32,CL
```



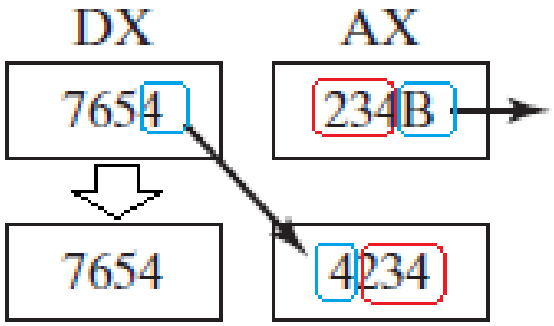
Приклади застосування SHLD та SHRD

```
.data
wval WORD 9BA6h
.code
mov   ax,0AC36h
shld  wval,ax,4
```

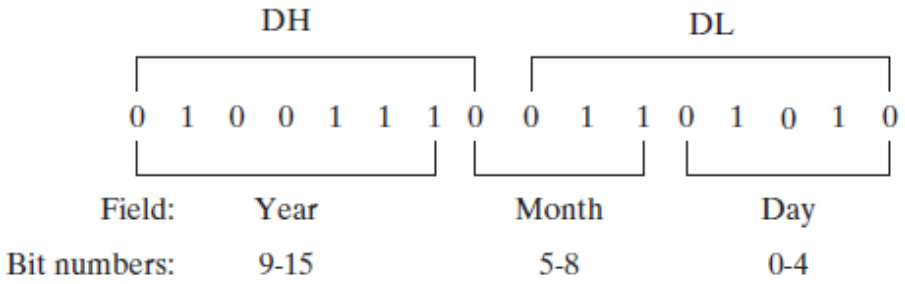



```
; wval = BA6Ah
```

```
mov   ax,234Bh
mov   dx,7654h
shrd  ax,dx,4
```



Приклад отримання частин від коду дати



а) Отримання значення "день"

```
mov  al,dl                  ; make a copy of DL
and  al,00011111b           ; clear bits 5-7
mov  day,al                 ; save in day
```

б) Отримання значення "місяць"

```
mov  ax,dx                  ; make a copy of DX
shr  ax,5                   ; shift right 5 bits
and  al,00001111b           ; clear bits 4-7
mov  month,al               ; save in month
```

в) Отримання значення "рік"

```
mov  al,dh                  ; make a copy of DH
shr  al,1                   ; shift right one position
mov  ah,0                   ; clear AH to zeros
add  ax,1980                ; year is relative to 1980
mov  year,ax                ; save in year
```

Команди циклу

LOOP (Loop) - Цикл
LOOPD (x86)
LOOPW (16-bit Counter)

Формат:
```
LOOP shortlabel
LOOPD shortlabel
LOOPW shortlabel
```

Зменшення значення регістру ECX/CX на 1, перевірка остачі та передача керування за вказаною міткою, якщо значення остачі в регістру ECX/CX більше нуля.
Мітка повинна вказувати на адресу із зміщенням -128...+127 байт відносно адреси наступної команди.
В процесорах IA-32 за умовчанням використовується регістр ECX.

Decrements ECX and jumps to a short label if ECX is not equal to zero. The destination must be −128 to +127 bytes from the current location.

O	D	I	S	Z	A	P	C

Формування ознак немає:

LOOPE, (Loop if Equal (Zero)) - Цикл, якщо нуль
LOOPZ

Формат:
```
LOOPE shortlabel
LOOPZ shortlabel
```

Декремент значення регістру ECX/CX, перевірка остачі та передача керування за вказаною міткою, якщо значення остачі в регістрі ECX/CX більше нуля і ознака ZF дорівнює одиниці.

Decrements (E)CX and jumps to a short label if (E)CX > 0 and the Zero flag is set.

Формування ознак немає

LOOPNE, (Loop if Not Equal (Zero)) - Цикл, якщо не нуль
LOOPNZ

Формат:
```
LOOPNE shortlabel
LOOPNZ shortlabel
```

Декремент значення регістру ECX/CX, перевірка остачі та передача керування за вказаною міткою, якщо значення остачі в регістрі ECX/CX більше нуля і ознака ZF дорівнює нулю.

Decrements (E)CX and jumps to a short label if (E)CX > 0 and the Zero flag is clear.

Формування ознак немає

Умовний і безумовний перехід

Jcondition

Conditional Jump

Умовний перехід

O	D	I	S	Z	A	P	C

Перехід до мітки, якщо вказане значення ознаки є істинним. На процесорах x86 зміщення мітки може бути позитивним або негативним 32-бітовим значенням.

Jumps to a label if a specified flag condition is true. When using a processor earlier than the x86, the label must be in the range of −128 to +127 bytes from the current location. On x86 processors, the label's offset can be a positive or negative 32-bit value.

Instruction format: `Jcondition label`

JMP

Jump Unconditionally to Label

Безумовний перехід

O	D	I	S	Z	A	P	C

Jump to a code label. A short jump is within −128 to +127 bytes from the current location. A near jump is within the same code segment, and a far jump is outside the current segment.

Instruction formats:

```
JMP shortlabel      JMP reg16
JMP nearlabel       JMP mem16
JMP farlabel        JMP mem32
```

Умовний перехід за ознаками (прапорами)

Mnemonic	Comment	Mnemonic	Comment
JA	Jump if above	JE	Jump if equal
JNA	Jump if not above	JNE	Jump if not equal
JAE	Jump if above or equal	JZ	Jump if zero
JNAE	Jump if not above or equal	JNZ	Jump if not zero
JB	Jump if below	JS	Jump if sign
JNB	Jump if not below	JNS	Jump if not sign
JBE	Jump if below or equal	JC	Jump if carry
JNBE	Jump if not below or equal	JNC	Jump if no carry
JG	Jump if greater	JO	Jump if overflow
JNG	Jump if not greater	JNO	Jump if no overflow
JGE	Jump if greater or equal	JP	Jump if parity
JNGE	Jump if not greater or equal	JPE	Jump if parity equal
JL	Jump if less	JNP	Jump if no parity
JNL	Jump if not less	JPO	Jump if parity odd
JLE	Jump if less or equal	JNLE	Jump if not less than or equal

Команди зіставлення операндів

CMP

Compare

Зіставлення

O	D	I	S	Z	A	P	C
*			*	*	*	*	*

Зіставлення операндів за результатами виконання неявної операції віднімання

Compares the destination to the source by performing an implied subtraction of the source from the destination.

Instruction formats:

```
CMP   reg,reg      CMP   reg,imm
CMP   mem,reg      CMP   mem,imm
CMP   reg,mem      CMP   accum,imm
```

TEST

Test

Тестування

O	D	I	S	Z	A	P	C
0			*	*	?	*	0

Порозрядна неявна операція ТА

Tests individual bits in the destination operand against those in the source operand. Performs a logical AND operation that affects the flags but not the destination operand.

Instruction formats:

```
TEST  reg,reg      TEST  reg,imm
TEST  mem,reg      TEST  mem,imm
TEST  reg,mem      TEST  accum,imm
```

Типи умов у командах передачі керування

Інструкції умовної передачі курування можна розділити на чотири групи відповідно до умов:

- значень прапора;
- рівності між операндами або значення (E) CX;
- порівняння незнакових операндів;
- порівняння знакових операндів.

1. Перехід за умовами відповідно до значень прапорів Zero, Carry, Overflow, Parity, Sign

Mnemonic	Description	Flags / Registers
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

2. Перехід за умовами відповідно до результатів перевірки на рівність

За результатами порівнянь двох операндів, наприклад командою CMP, або на основі значення CX, ECX (RCX).

```
CMP leftOp,rightOp
```

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0
JRCXZ	Jump if RCX = 0 (64-bit mode)

3. Перехід за умовами відповідно до результатів порівняння незнакових операндів

```
CMP leftOp,rightOp
```

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* ≥ *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* ≤ *rightOp*)
JNA	Jump if not above (same as JBE)

4.Перехід за умовами відповідно до результатів порівняння знакових операндів

```
CMP leftOp,rightOp
```

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* ≥ *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* ≤ *rightOp*)
JNG	Jump if not greater (same as JLE)

Приклади команд умовного переходу

```
mov  al,+127    ; hexadecimal value is 7Fh
cmp  al,-128    ; hexadecimal value is 80h
ja   IsAbove    ; jump not taken, because 7Fh < 80h
jg   IsGreater  ; jump taken, because +127 > -128
```

Example 1

```
mov   edx,-1
cmp   edx,0
jnl   L5   ; jump not taken (-1 >= 0 is false)
jnle  L5   ; jump not taken (-1 > 0 is false)
jl    L1   ; jump is taken (-1 < 0 is true)
```

Example 2

```
mov   bx,+32
cmp   bx,-35
jng   L5   ; jump not taken (+32 <= -35 is false)
jnge  L5   ; jump not taken (+32 < -35 is false)
jge   L1   ; jump is taken (+32 >= -35 is true)
```

Example 3

```
mov  ecx,0
cmp  ecx,0
jg   L5    ; jump not taken (0 > 0 is false)
jnl  L1    ; jump is taken (0 >= 0 is true)
```

Example 4

```
mov  ecx,0
cmp  ecx,0
jl   L5    ; jump not taken (0 < 0 is false)
jng  L1    ; jump is taken (0 <= 0 is true)
```

Приклади переходу відповідно до значень контрольних бітів

Example 1

```
mov   al,status
test  al,00100000b   ; test bit 5
jnz   DeviceOffline
```

Example 2

```
mov   al,status
test  al,00010011b   ; test bits 0,1,4
jnz   InputDataByte
```

Example 3

```
mov   al,status
and   al,10001100b   ; mask bits 2,3,7
cmp   al,10001100b   ; all bits set?
je    ResetMachine   ; yes: jump to label
```

Сканування бітів

Сканує операнд з метою пошуку першого встановленого біту. Якщо біт знайдено, то прапорець ZF=0, а в операнді dist вказується на номер біта (індекс) першого біту. Якщо встановлений біт не знайдено, то ZF=1.
BSF сканує від біта LSB до MSB, а BSR - від біта MSB до LSB .

BSF, BSR

Bit Scan (x86)

O	D	I	S	Z	A	P	C
?			?	?	?	?	?

Scans an operand to find the first set bit. If the bit is found, the Zero flag is cleared, and the destination operand is assigned the bit number (index) of the first set bit encountered. If no set bit is found, ZF = 1. BSF scans from bit 0 to the highest bit, and BSR starts at the highest bit and scans toward bit 0.

Instruction formats (apply to both BSF and BSR):

```
BSF reg16,r/m16
BSF reg32,r/m32
```

Тестування бітів

BT, BTC, BTR, BTS

Bit Tests (x86)

Тестування бітів

O	D	I	S	Z	A	P	C
?			?	?	?	?	*

Copies a specified bit (*n*) into the Carry flag. The destination operand contains the value in which the bit is located, and the source operand indicates the bit's position within the destination. BT copies bit *n* to the Carry flag. BTC copies bit *n* to the Carry flag and complements bit *n* in the destination operand. BTR copies bit *n* to the Carry flag and clears bit *n* in the destination. BTS copies bit *n* to the Carry flag and sets bit *n* in the destination.

BT встановлює ознаку CF (Carry) відповідно до значення біту операнда-отримувача, номер якого (n) вказаний в операнді-джерелі.
BTC, BTR, BTS також (відповідно) інвертує, скидає в 0, встановлює в 1 значення біту (n) операнда-отримувача.

Instruction formats:

```
BT r/m16,imm8
BT r/m32,imm8
BT r/m16,r16
BT r/m32,r32
```

Умовне встановлення

SET*condition*	**Set Conditionally** **Умовне встановлення** O D I S Z A P C ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐
	If the given flag condition is true, the byte specified by the destination operand is assigned the value 1. If the flag condition is false, the destination is assigned a value of 0. The possible values for *condition* were listed in Table B-2. Встановлення в одиничне значення байту операнда-отримувача у разі наявності вказаної умови, інакше - в нульове значення. Умови аналогічні умовампереходу за ознаками (J) Instruction formats: `SETcond reg8` `SETcond mem8`

Завдання до самостійної роботи

1. Засвоїти формат команд на мові асемблера архітектури IA-32 і правила виконання відповідних операцій.
2. Виконати в середовищі MASM32 і налагоджувачі усі приклади, які наведено на слайдах, з вказаними даними та іншими довільними операндами.

Література

1. Intel® 64 and IA-32 Architectures Software Developer's Manual. - Order Number: 325462-067US, May 2018 (ch.5)
2. Kip R. Irvine. Assembly Language for x86 Processors. Florida International University School of Computing and Information Sciences. 7th Edition, 2014 (ch.6.1-6.4,7.1,app.A)