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:

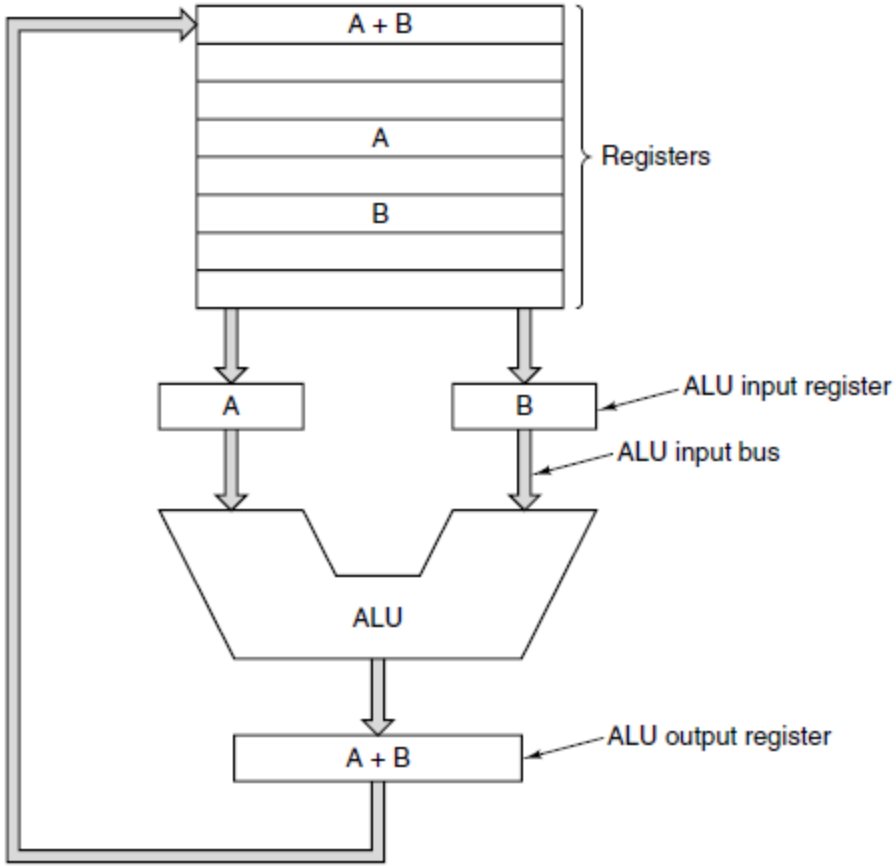
- двійкове кодування – команди і операнди кодуються двійковими цифрами 0 та 1 і мають відповідний *формат*;
- програмне керування – обчислення за алгоритмом вирішення завдання представлено у формі програми із команд, які вказують на операції для виконання відповідних дій;
- одноманітність пам'яті – команди 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

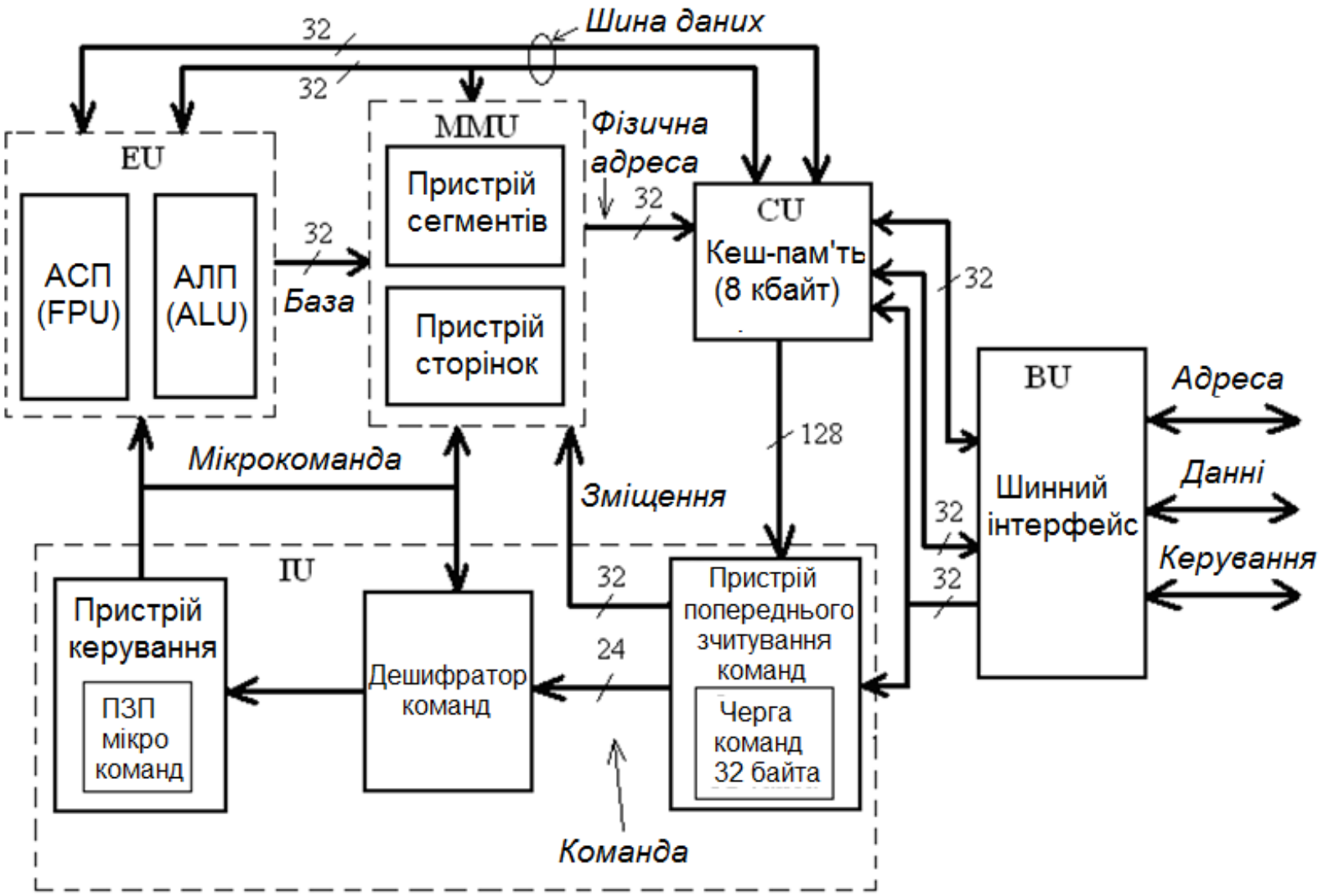
Потік даних (неймановська архітектура)



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

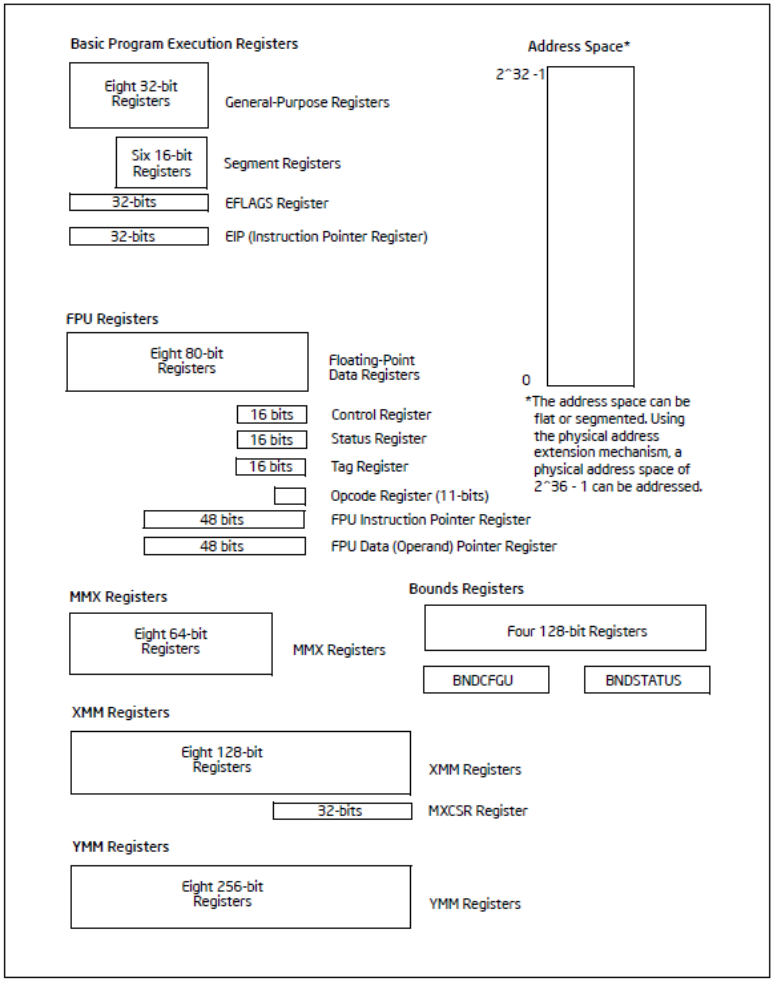
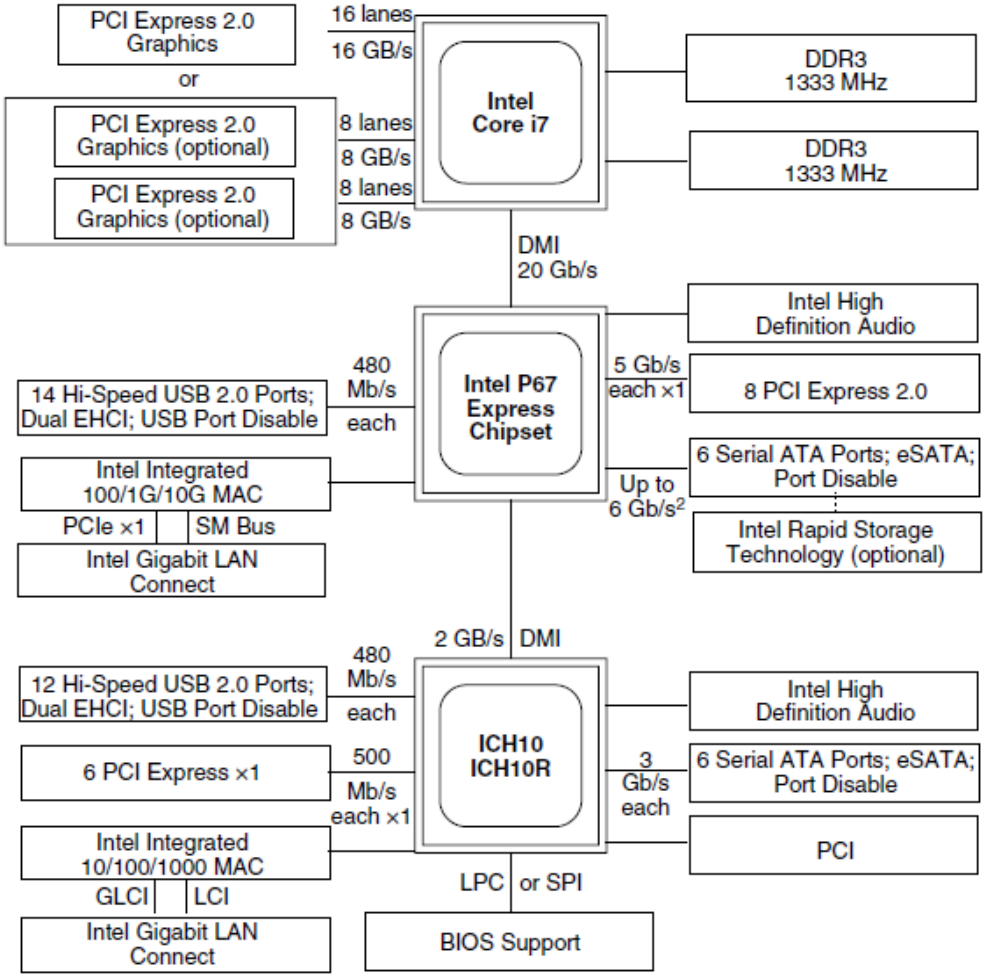
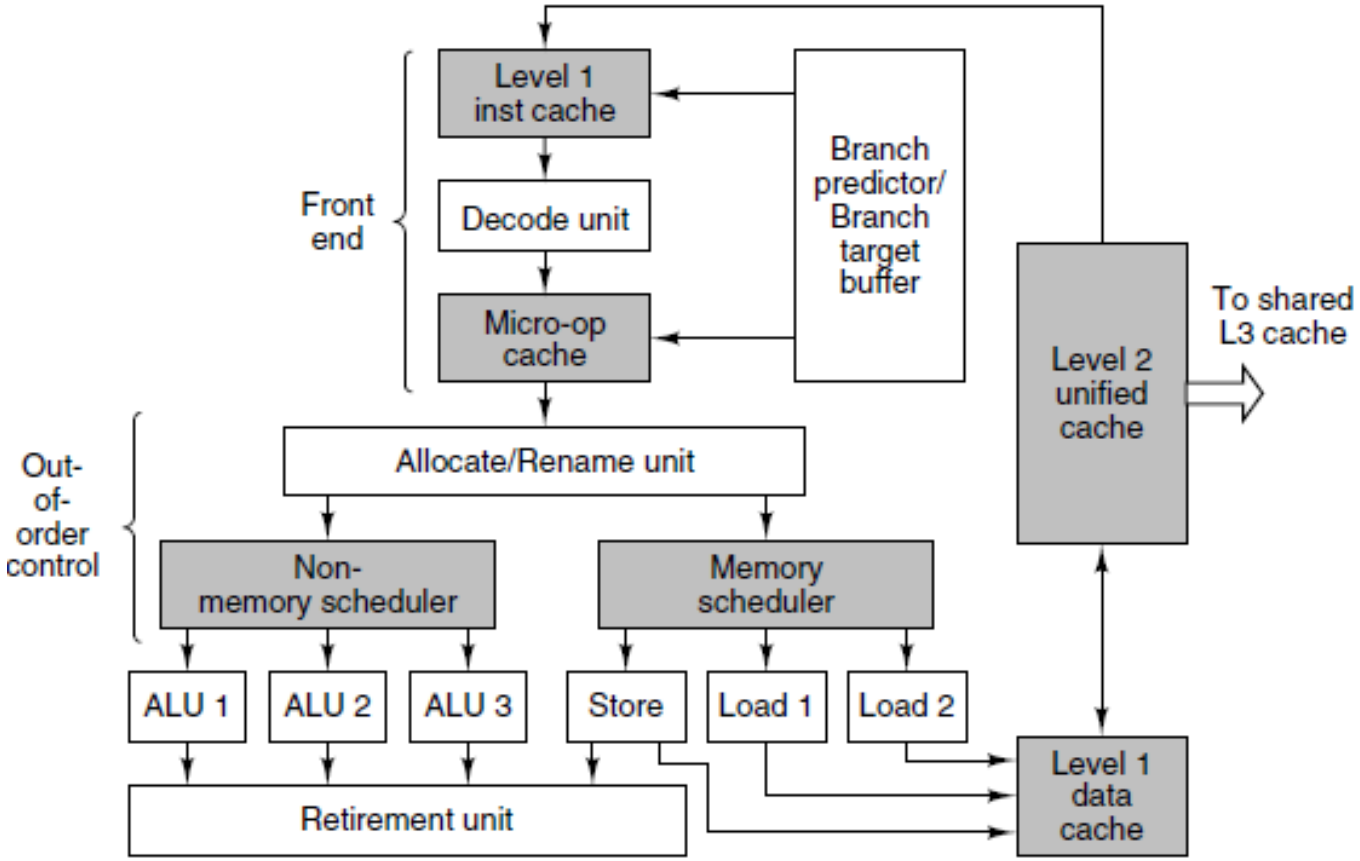


Figure 3-1. IA-32 Basic Execution Environment for Non-64-bit Modes

Структура процесору INTEL 64



Потік даних (процесор INTEL Core i7)



Архітектура INTEL64

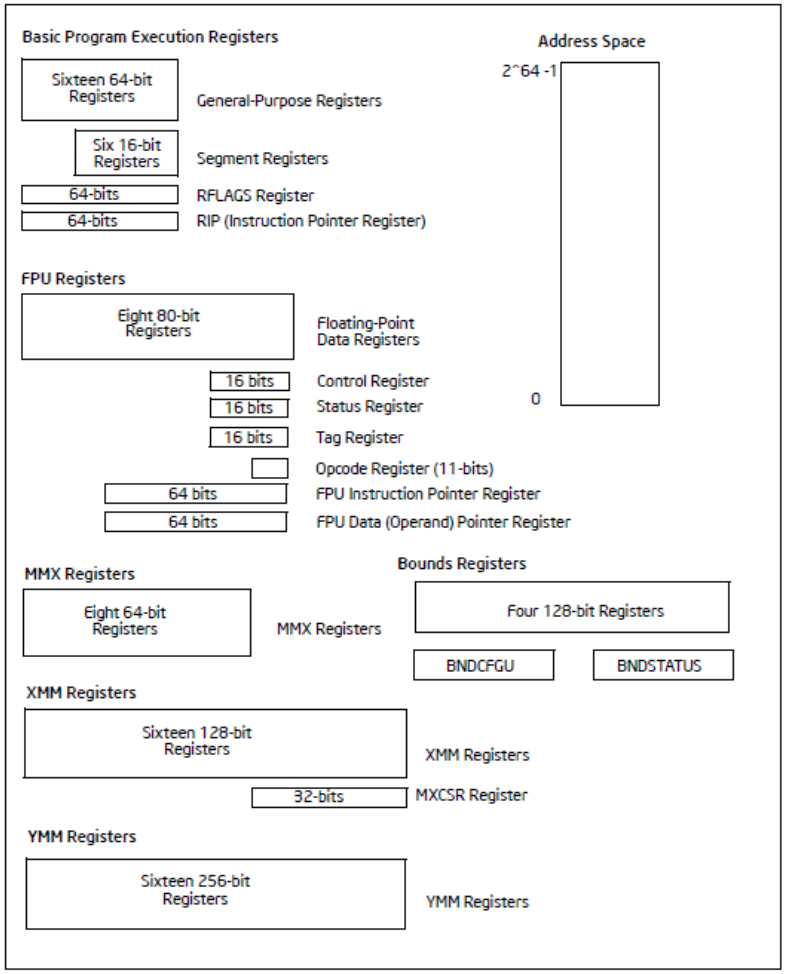
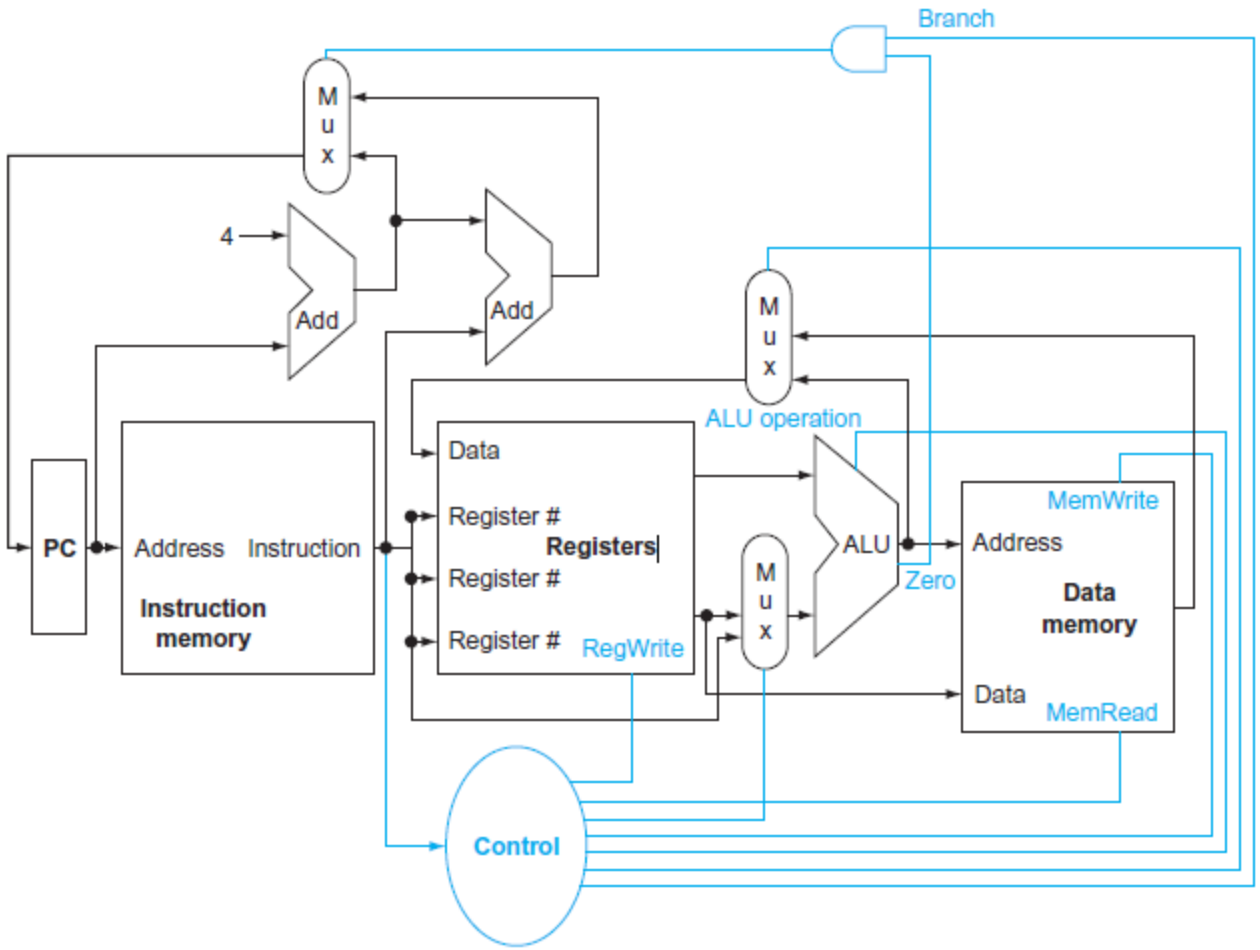
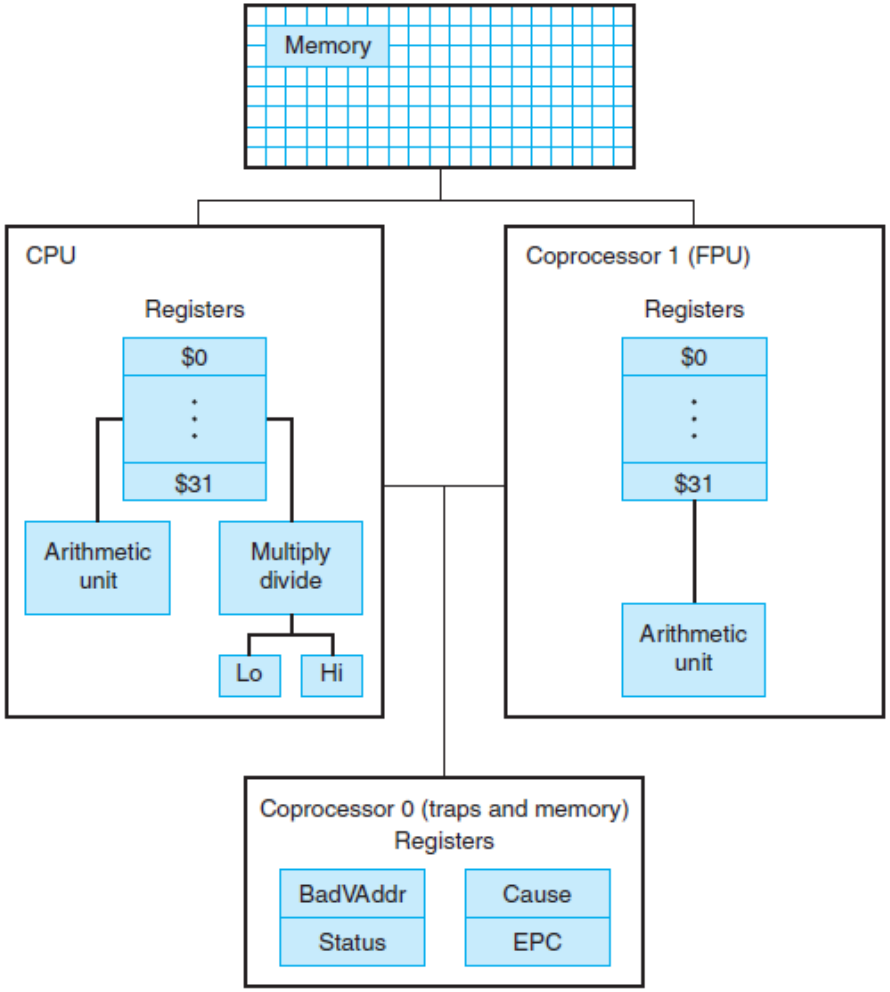


Figure 3-2. 64-Bit Mode Execution Environment

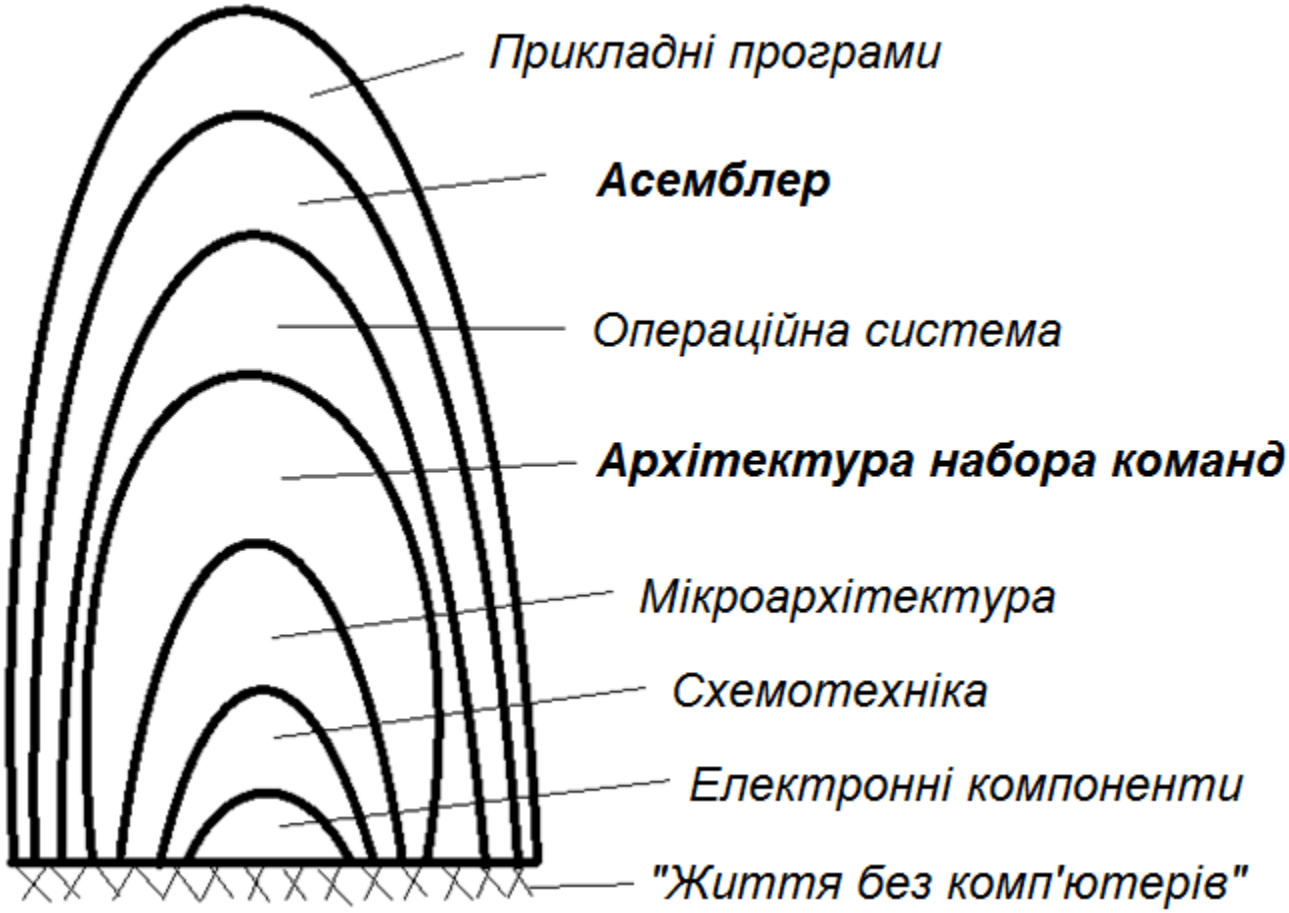
Потік даних (процесор MIPS)



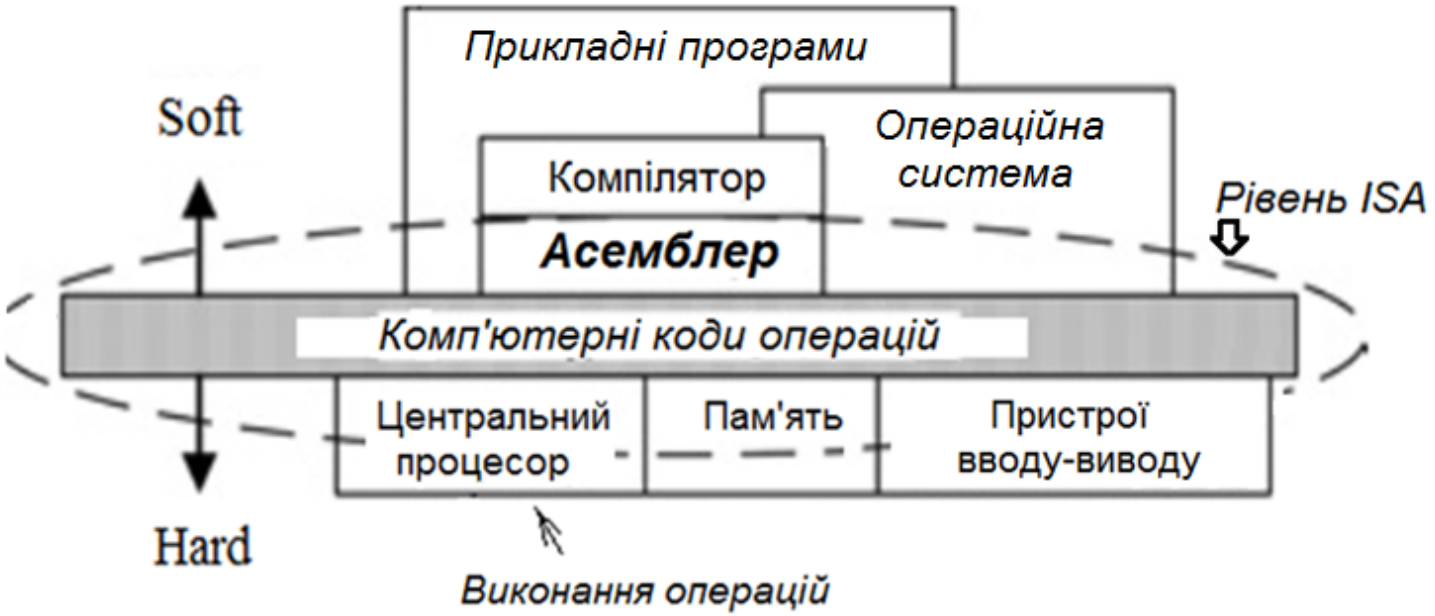
Архітектура MIPS R2000



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
```

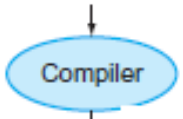
Еквівалент на асемблері:

```
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];
```

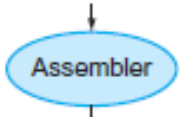
High-level language program (in C)



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