Київський Національний Університет ім. Тараса Шевченко
Фізичний факультет, кафедра ядерної фізики

Основи мікропроцесорної техніки

Програмне забезпечення мікропроцесорів та мікроконтроллерів

Викладач:
канд. фіз.-мат. наук, асистент
Єрмоленко Руслан
Вікторович

http://10.22.1.39/CPU/Lection14.pdf

Зміст

–Рівні архітектури мікропроцесорів
–Мови високого рівня програмування
–Асемблер як мова програмування
–Машинні коди
–Функції операційної системи
–Роль компіляторів
–Методика програмування мікроконтролерів
–Практичне завдання

Література

–Никитюк Н.М. Микропроцессоры и микро ЭВМ: применение в приборостроении и научных исследованиях.

– Финогенов К.Г., Програмирование измерительных систем реального времени

–Ступин Ю.В. Методы автоматизации физических экспериментов и установок на основе ЭВМ

–http://10.22.1.39/CPU/

Рівні архітектури мікропроцесорів

- Цифровий логічний рівень
- Рівень мікроархітектури
- Рівень архітектури набору команд
- Рівень операційної системи
- Рівень машинних команд
- Рівень асемблера
- Мови високого рівня програмування

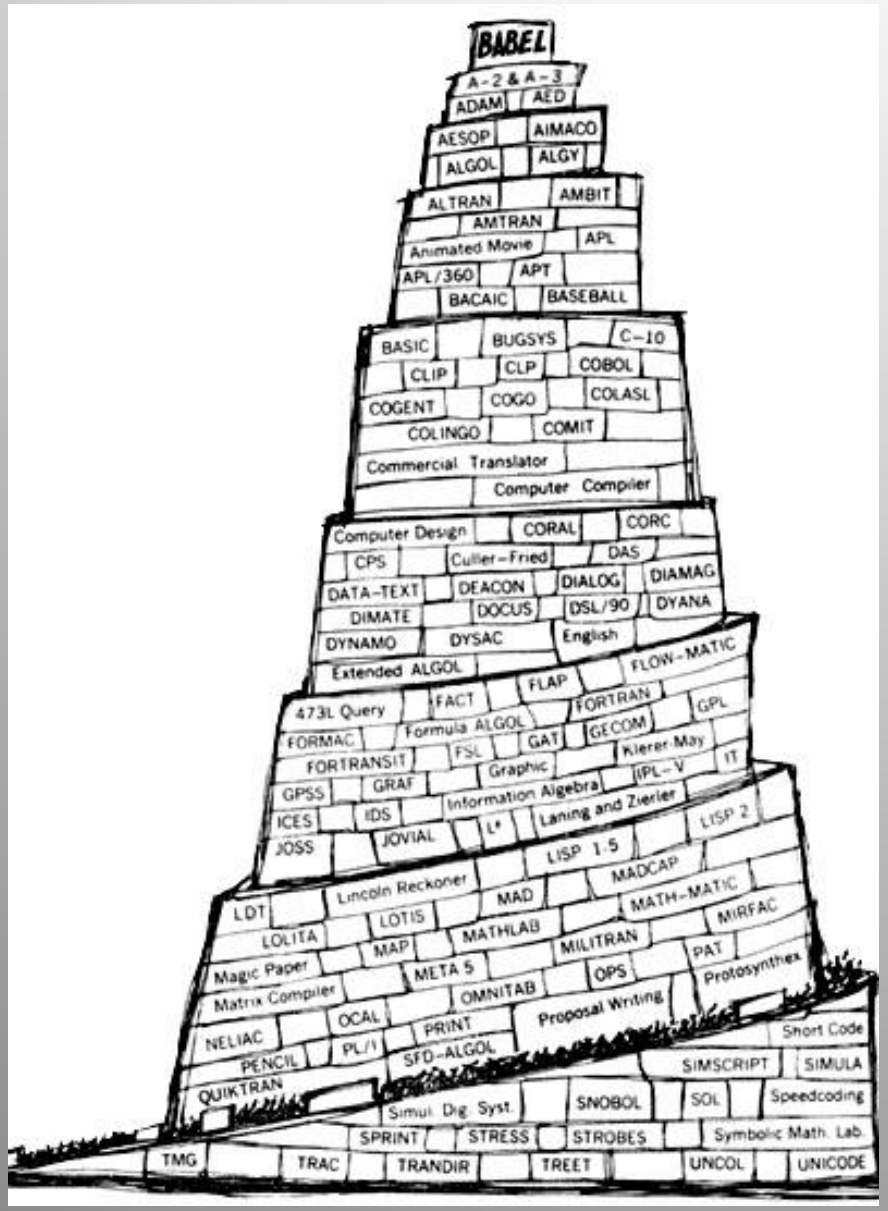
Мови високого рівня програмування

Мо́ва програмува́ння — система позначень для опису алгоритмів та структур даних, певна штучна формальна система, засобами якої можна виражати алгоритми.

Мову програмування визначає набір лексичних, синтаксичних і семантичних правил, що задають зовнішній вигляд програми і дії, які виконує виконавець (комп'ютер) під її управлінням.

З часу створення перших програмованих машин було створено понад дві з половиною тисячі мов програмування. Щороку їх кількість поповнюється новими. Деякими мовами вміє користуватись тільки невелике число їх власних розробників, інші стають відомі мільйонам людей. Професійні програмісти зазвичай застосовують в своїй роботі декілька мов програмування.

Мови високого рівня програмування



Зображення «Вавилонської вежі» з обкладинки книги Джін Семміт «Мови програмування» (1969 р.), яка містила огляд мов програмування того часу

Мови високого рівня програмування

Синтаксис

Синтаксис мови програмування визначає те, як буде виглядати програма на цій мові, зокрема, як пишуться оператори, оголошення і інші мовні конструкції.

Наприклад, оголошення масиву **Vector** з десяти цілочислових елементів в на мові **C++** буде виглядати так:

int Vector [10];

На мові Pascal:
Vector : array [0…9] of integer

Мови високого рівня програмування

Типи даних

Область зберігання даних в апаратній частині комп'ютера (пам'ять, регістри і зовнішні запам'ятовуючі пристрої) зазвичай мають доволі просту структуру в вигляді послідовності бітів, згрупованих в байти або слова.

Проте в віртуальному комп'ютері, як правило, організовано більш складним чином — в різні моменти виконання програми використовуються такі форми зберігання даних, як стеки, масиви, числа, символьні рядки та інші.

Один або декілька однотипних елементів даних, об'єднаних в одне ціле в віртуальному комп'ютері в певний момент виконання програми, прийнято називати об'єктом даних.

При виконанні програми існує багато об'єктів даних різних типів. Тип даних — це деякий клас об'єктів даних разом з набором операцій для створення і роботи з ним.

В кожній мові програмування є певний набір вбудованих примітивних типів даних. Додатково в мові можуть бути передбачені засоби, що дозволяють програмісту визначати нові типи даних.

Мови високого рівня програмування

Класифікація мов програмування

Рівень абстракції

Мови програмування високого рівня оперують сутностями ближчими людині, такими як об'єкти, змінні, функції.

Мови програмування нижчого рівня оперують сутностями ближчими машині: байти, адреси, інструкції.

Текст програми на мові високого рівня зазвичай набагато коротший ніж текст такої самої програми на мові низького рівня, проте програма має більший розмір.

Мови високого рівня програмування

Класифікація мов програмування

Підтримувані парадигми програмування

Об'єктно-орієнтовані, логічні, функційні, структурні…

Імперативні мови базуються на ідеї змінної, значення якої змінюється присвоєнням. Вони називаються імперативними (лат. imperative - наказовий), оскільки складаються із послідовностей команд, які звичайно містять присвоєння змінних <назва_змінної> = <вираз>, де вираз може посилатися на значення змінних присвоєних попередніми командами.

Парадигма програмування — основні принципи програмування (не плутати з розробкою програм), або, парадигмне програмування.
Парадигма програмування надає (та визначає) те, як програміст розглядає роботу програми. Наприклад, в об'єктно-орієнтованому програмуванні, програміст розглядає програму як множину взаємодіючих об'єктів, в той час як у функційному програмуванні програму можна представити як послідовність обчислення функцій без станів.

Мови високого рівня програмування

Класифікація мов програмування

Об'єктно́-орієнтова́не програмува́ння — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. В ній використано декілька технологій від попередніх парадигм, зокрема успадкування, модульність, поліморфізм та інкапсуляцію. Не зважаючи на те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих. На сьогодні багато із мов програмування (зокрема, Java, C#, C++, Python, PHP, Ruby та Objective-C, ActionScript 3) підтримують ООП.

Мови високого рівня програмування

Класифікація мов програмування

Функційне програмування — парадигма програмування, яка розглядає програму як обчислення математичних функцій та уникає станів та змінних даних. Функційне програмування наголошує на застосуванні функцій, на відміну від імперативного програмування, яке наголошує на змінах в стані та виконанні послідовностей команд.

Іншими словами, функційне програмування є способом створення програм, в яких єдиною дією є виклик функції, єдиним способом розбиття програми є створення нового імені функції та задання для цього імені виразу, що обчислює значення функції, а єдиним правилом композиції є оператор суперпозиції функцій. Жодних комірок пам'яті, операторів присвоєння, циклів, ні, тим більше, блок схем чи передачі управління.

Приклад: Програмний пакет **Mathematica (символьні обчислення)**

Мови високого рівня програмування

Мови програмування високого рівня можна сказати є більш зрозумілими людині, ніж комп'ютеру. Особливості конкретних комп'ютерних архітектур в них не враховуються, тому створені програми легко переносяться з комп'ютера на комп'ютер. Здебільшого достатньо просто перекомпілювати програму під певну комп'ютерну архітектурну та операційну систему. Розробляти програми на таких мовах значно простіше і помилок допускається менше. Значно скорочується час розробки програми, що особливо важливо при роботі над великими програмними проектами.

Приклади:
Фортран, Кобол, Алгол, Pascal, Java, C, C++, C#, Delphi, Basis

Недоліком мов високого рівня є більший розмір програм порівняно з програмами на мові низького рівня. Тому в основному мови високого рівня використовуються для розробок програмного забезпечення комп'ютерів, і пристроїв, які мають великий обсяг пам'яті. А різні підвиди асемблеру застосовуються для програмування інших пристроїв, де критичним є розмір програми.

Мови програмування низького рівня

Перші комп'ютери доводилось програмувати двійковими **машинними кодами.**
Проте програмувати таким чином — доволі трудомістка і важка задача.

Для спрощення цієї задачі почали з'являтися мови програмування низького рівня, які дозволяли задавати машинні команди в більш зрозумілому для людини вигляді.

Для перетворення їх у двійковий код були створені спеціальні програми — транслятори (компілятори або інтерпретатори).

Маши́нний код — набір команд, що може зрозуміти й обробити центральний процесор комп'ютера без транслятора.

Кожен тип центрального процесора має власний машинний код. Оскільки машинний код складається повністю з двійкових чисел (бітів), більшість програмістів пишуть програми на мовах програмування високого рівня. Програми, написані такими мовами повинні транслюватися в машинний код, що здійснює компілятор або інтерпретатор програм, до того, як комп'ютер починає їх виконувати.

Мови програмування низького рівня

Машинний код		Асемблер	
0005	B4 09	7	mov AH, 09h
0007	BA0000r	8	mov DX, offset msg
00A	CD 21	9	int 21 h

Мови програмування низького рівня

Прикладом мови низького рівня є **асемблер.**

Мова асемблера - це система запису програми з деталізацією до окремої машинної команди, яка дозволяє використовувати мнемонічне позначення команд і символічне завдання адрес.

Оскільки в різних апаратних архітекторах використовуються різні програмно-доступні компоненти (система команд, регістри, засоби адресації), то мова асемблера є апаратно-залежною. Програми, написані мовою асемблера, можуть бути перенесені тільки на обчислювальну систему відповідної архітектури.

Програмування мовою асемблера дозволяє максимально використовувати особливості архітектури обчислювальної системи. Донедавна сприймалося як аксіома, що асемблерна програма завжди є більш ефективною і у розумінні швидкодії, і у розумінні вимог до пам'яті. Для Intel-архітектури це й зараз так. Але вже не так для RISC-архітектур. Для того щоб програма могла ефективно виконуватися в обчислювальному середовищі з розпаралелюванням на рівні команд, вона має бути певним чином оптимізована, тобто команди мають бути розташовані у певному порядку, який допускає їх паралельне виконання. Програміст просто не зможе покомандно оптимізувати всю свою програму. З такою оптимізацією більш ефективно справляються компілятори.

Мови програмування низького рівня

У загальних випадках речення мови асемблера складаються з таких компонентів:

–**мітка або ім'я;** // адреса памяті, на яку передається керування)
–**мнемоніка;** // символічне позначення команди/псевдокоманди)
–**операнди;** // один чи декілька компонентів команди, що зазвичай розділяються комами)
–**коментарі.**

Мови програмування низького рівня

Константами, в команді зокрема, можна подавати безпосередні операнди та абсолютні адреси пам'яті. Застосовуються десяткові, вісімкові, шістнадцяткові та двійкові коди і символьні константи.

Імена змінних - це адреси комірок пам'яті. Під час асемблювання асемблер перетворює імена на адреси. Спосіб перетворення імені на значення залежить від прийнятого способу адресації. Одним із часто вживаних способів адресації в машинних мовах є відносна адресація: адреса у команді задається у вигляді зміщення відносно якоїсь базової адреси, значення якої міститься у деякому базовому регістрі. Як базовий можуть застосовуватися або спеціальні регістри (*DS*, *CS* - фірма *Intel*), або регістри загального призначення (*S/390*).

Літерали - це записані в особливій формі константи. Концептуально літерали - ті ж імена. При появі в програмі літерала асемблер виділяє комірку пам'яті та записує до неї задану в літералі константу. Далі всі появлення цього літерала асемблер замінює на звертання за адресою цієї комірки. Таким чином, літеральні константи зберігаються в пам'яті в одному екземплярі незалежно від числа звернень до них.

Мови програмування низького рівня

```
$MOD52                    ; use 8052 predefined symbols

LED    EQU    P3.4        ; P3.4 is red LED on eval board
;-------------------------------------------------------------------
                          ; MAIN PROGRAM
ORG  0000h
BLINK:  CPL    LED        ; flash (complement) the red LED
        CALL   DELAY      ; call software delay
        JMP    BLINK      ; repeat indefinately
;-------------------------------------------------------------------
                          ; SUBROUTINES

DELAY:                    ; delay 100ms

     MOV    R7,#200       ; 200 * 500us = 100ms
DLY1: MOV   R6,#229       ; 229 * 2.17us = 500us
     DJNZ   R6,$          ; sit here for 500us
     DJNZ   R7,DLY1       ; repeat 200 times (100ms delay)
     RET
END
```

http://10.22.1.39/Labs/auto_nucl/lab2/add/ASSEMBLER_and_DOWNLOADER/ASM51/ASM51.pdf

Операційна система

Операці́йна систе́ма — це базовий комплекс програмного забезпечення, що виконує управління апаратним забезпеченням комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організує взаємодію з користувачем.

Операційна система звичайно складається з ядра операційної системи та базового набору прикладного програмного забезпечення.

Операційна система

Головні функції:

–Виконання на вимогу програм користувача тих елементарних (низькорівневих) дій, які є спільними для більшості програмного забезпечення і часто зустрічаються майже у всіх програмах (ввід і вивід даних, запуск і зупинка інших програм, виділення та вивільнення додаткової пам'яті тощо).

–Стандартизований доступ до периферійних пристроїв (пристрої введення-виведення).

–Завантаження програм у оперативну пам'ять і їх виконання.

–Керування оперативною пам'яттю (розподіл між процесами, організація віртуальної пам'яті).

–Керування доступом до даних енергозалежних носіїв (твердий диск, оптичні диски тощо), організованим у тій чи іншій файловій системі.
–Забезпечення користувацького інтерфейсу.

–Мережеві операції, підтримка стеку мережевих протоколів.

Операційна система

Додаткові функції:

–Паралельне або псевдопаралельні виконання задач (багатозадачність).
–Розподіл ресурсів обчислювальної системи між процесами.

–Організація надійних обчислень (неможливість впливу процесу на перебіг інших), основана на розмежуванні доступу до ресурсів.

–Взаємодія між процесами: обмін даними, синхронізація.

–Захист самої системи, а також користувацьких даних і програм від дій користувача або програм.

–Багатокористувацький режим роботи та розділення прав доступу (автентифікація, авторизація).

Операційна система

Найбільш відомі операційні системи:

MS-DOS	PC DOS
WINDOWS	UNIX
LINUX	Mac OS

Операційна система

До складу операційної системи входять:

Ядро операційної системи, що забезпечує розподіл та управління ресурсами обчислювальної системи;

базовий набір прикладного програмного забезпечення, системні бібліотеки та програми обслуговування.

Ядро системи — це набір функцій, структур даних та окремих програмних модулів, які завантажуються в пам'ять комп'ютера при завантаженні операційної системи та забезпечують три типи системних сервісів:

- ✓управління введенням-виведенням інформації (підсистема вводу-виводу ядра ОС);
- ✓управління оперативною пам'яттю (підсистема управління оперативною пам'яттю ядра ОС);
- ✓управління процесами (підсистема управління процесами ядра ОС).

Операційна система

Багатозадачні операційні системи також включають ще одну обов'язкову складову - механізм підтримки багатозадачності. Ця складова не надається в якості системного сервісу і тому не може бути віднесена до жодної з підсистем.

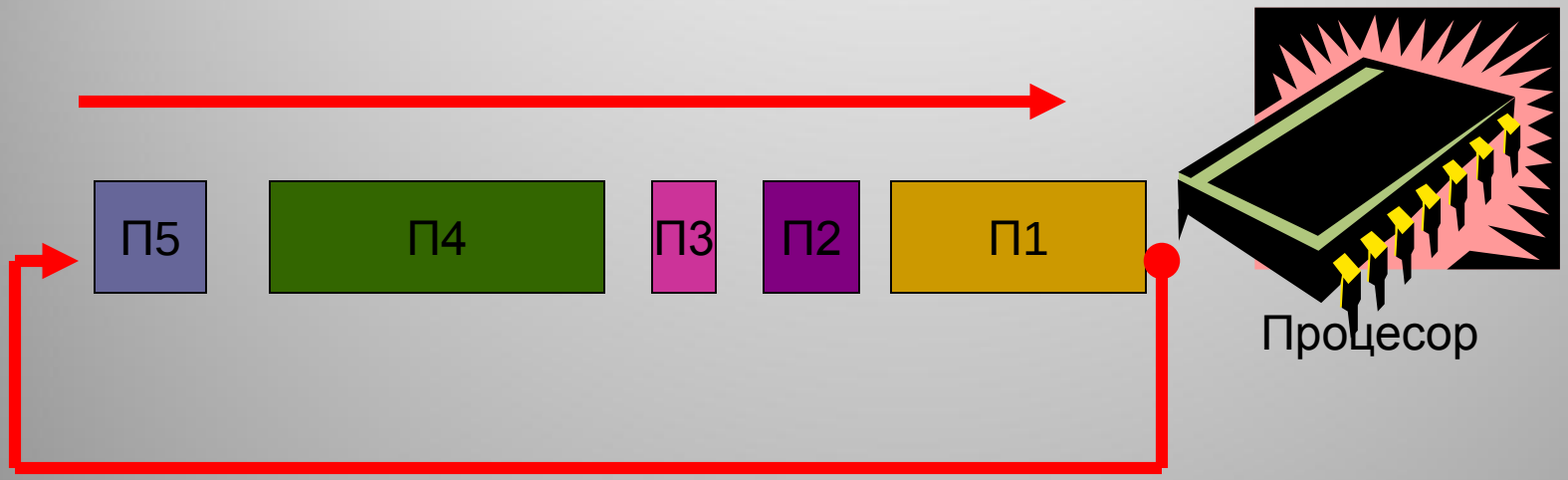
Операційна система

Існує три основних механізми забезпечення багатозадачності (планування задач):

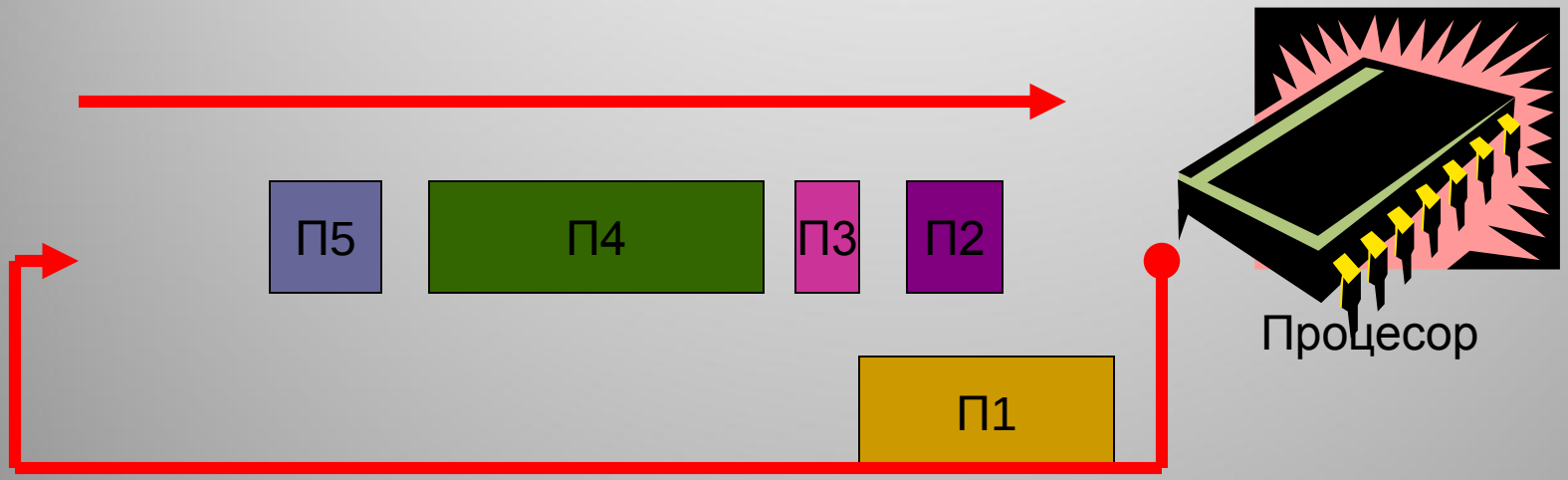
1. шляхом надання процесора окремій задачі на квант часу, який визначається самою задачею (кооперативна Багатозадачність; останнім часом практично не використовується або область використання значно обмежена всередині процесів);
2. шляхом надання процесора окремій задачі на квант часу, який визначається обладнанням обчислювальної системи - інтервальним таймером;
3. виділення під окрему задачу окремого процесора в багатопроцесорних системах.

В сучасних системах, як правило комбінується методи 2 і 3.

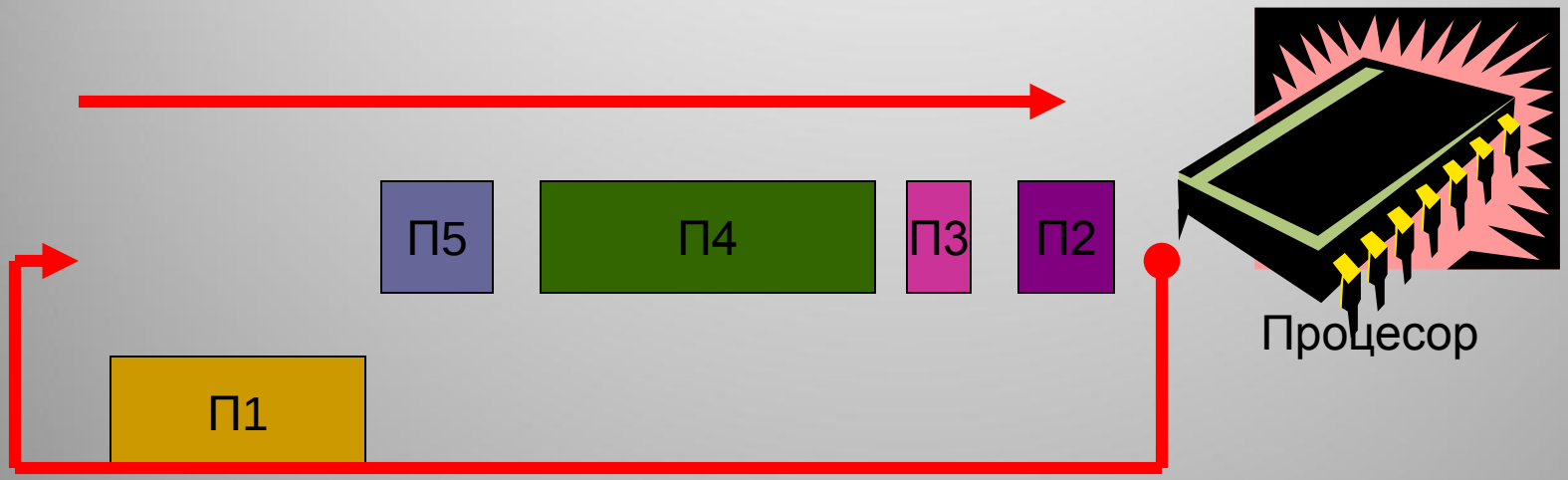
Операційна система



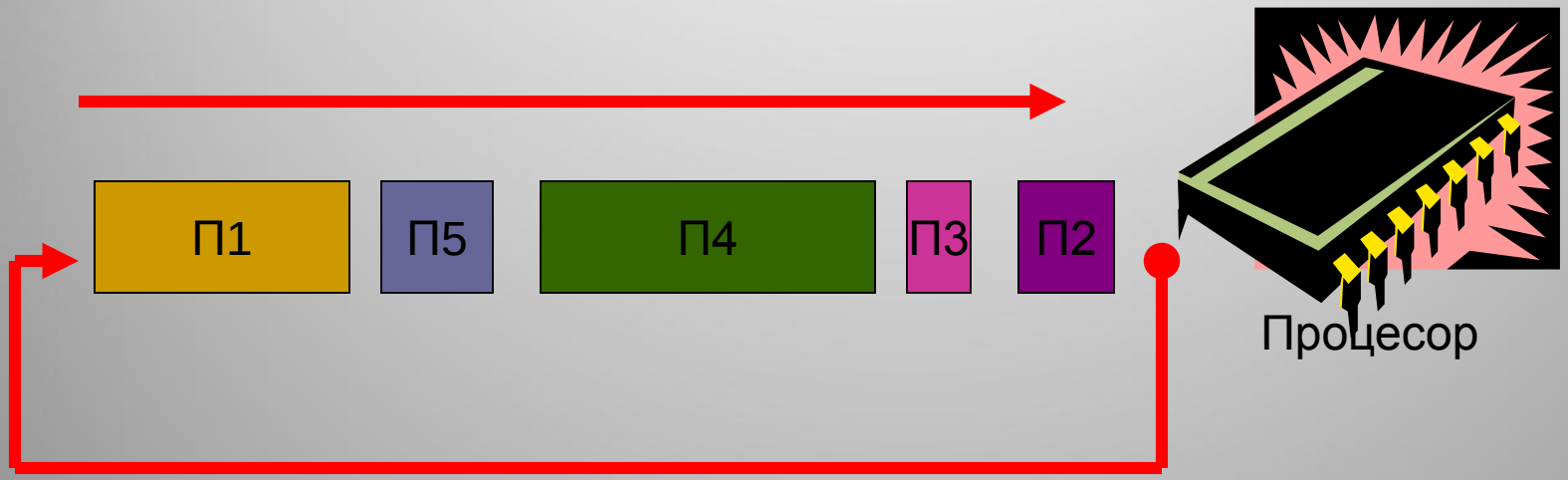
Операційна система



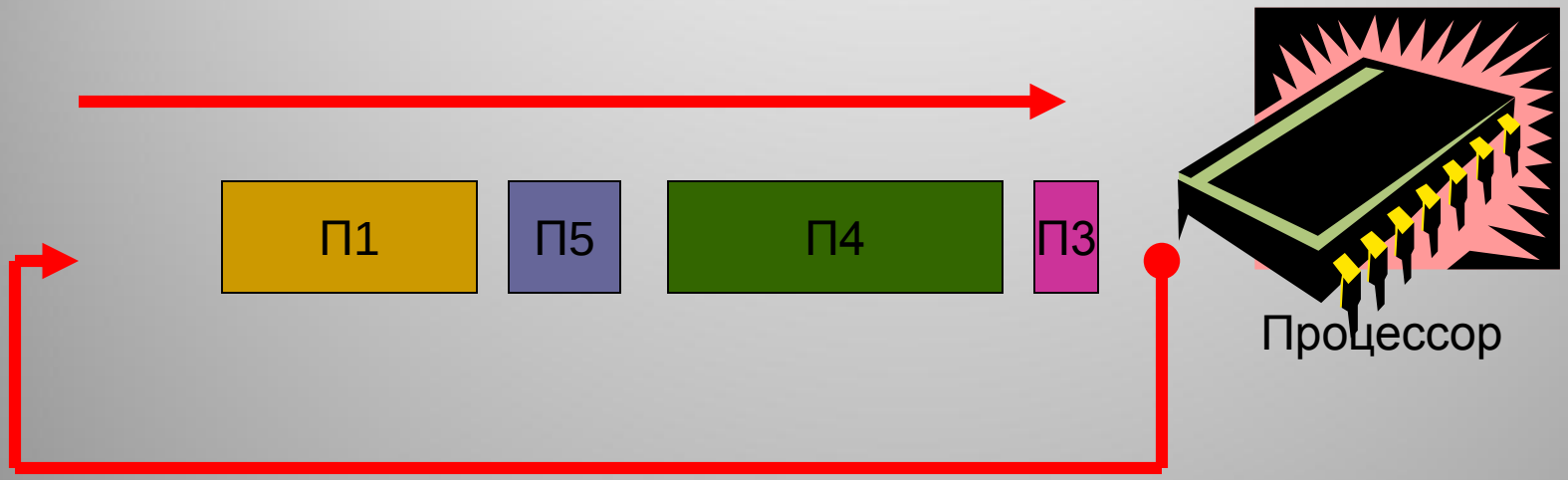
Операційна система



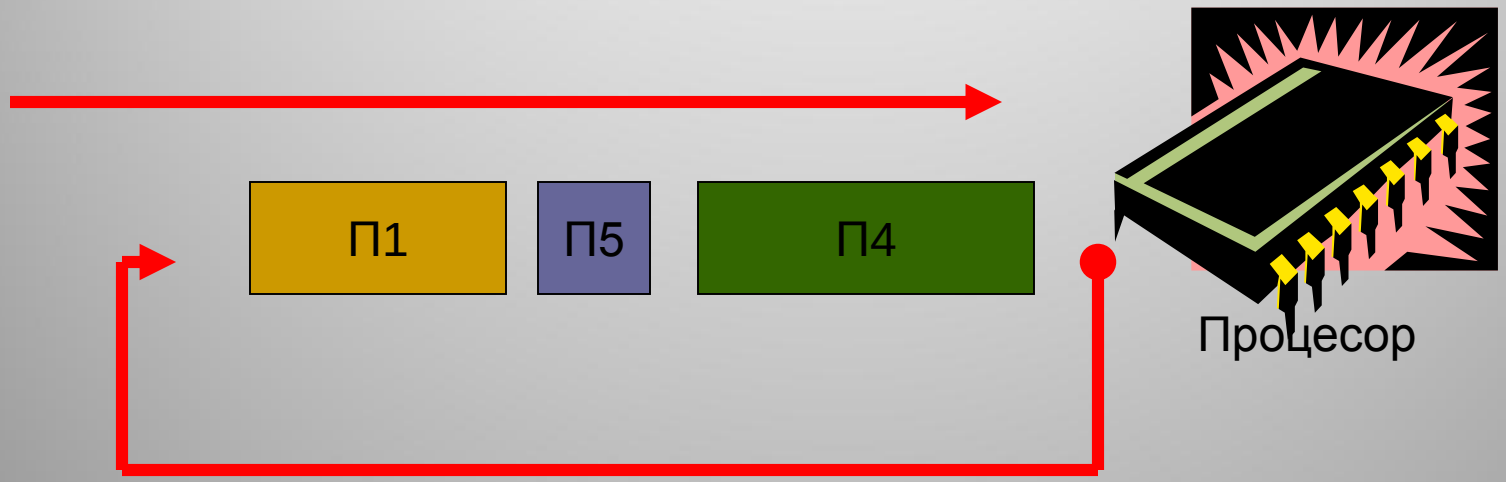
Операційна система



Операційна система



Операційна система



Компілятори

Транслятори поділяються на:

інтерпретатори — перетворюють частину програми в машинний код, виконують і після цього переходять до наступної частини. При цьому щоразу при виконанні програми використовується інтерпретатор.

компілятори — перетворюють текст програми в машинний код, який можна зберегти і після цього використовувати уже без компілятора (прикладом є виконувальні файли з розширенням *.exe).

Компілятори

Компілятор – це програма, що читає програму записану початковою мовою і записує цільовою мовою. Цей процес називають компіляцією (трансляцією, перекладом).

Він складається з двох частин

Аналіз (parsing) – розбиття початкової програми на складові частини та створення проміжного представлення

Синтез – побудова цільової програми з проміжного представлення

Початкова мова визначається її синтаксисом – описом того, з яких конструкцій складається мова, та семантикою – набором правил, що визначають суть цих конструкцій.

Компілятори

Фази компіляції

Концептуально компілятор працює фазово, в процесі кожної фази відбувається перетворення початкової програми з одного представлення до іншого. На практиці фази можуть об'єднуватись і деякі проміжні представлення можуть не будуватись в явному вигляді.

Типове розбиття компілятора на фази:

- Лексичний аналізатор
- Синтаксичний аналізатор
- Семантичний аналізатор
- Генератор проміжного коду
- Оптимізатор
- Генератор цільового коду

Компілятори

Лексичний розбір

Лексичний розбір виділяють для спрощення побудови компілятора.

Це лінійне сканування вхідної програми, при якому символи групуються в *токени* - послідовності символів, що мають певне сукупне значення.
Наступний рядок мовою C++
Len = 3.14 * r;

складається з наступних токенів
Ідентифікатор Len
Символ присвоєння =
Числова стала 3.14
Знак множення *
Ідентифікатор r
Роздільник операторів ;

Компілятори

Лексичний розбір

Лексичний розбір виділяють для спрощення побудови компілятора.

Це лінійне сканування вхідної програми, при якому символи групуються в *токени* - послідовності символів, що мають певне сукупне значення.
Наступний рядок мовою C++
Len = 3.14 * r;

складається з наступних токенів
Ідентифікатор Len
Символ присвоєння =
Числова стала 3.14
Знак множення *
Ідентифікатор r
Роздільник операторів ;

Методика програмування мікроконтролерів

Мікроконтро́лер, або однокристальна мікроЕОМ — виконана у вигляді мікросхеми спеціалізована мікропроцесорна система, що включає мікропроцесор, блоки пам'яті для збереження коду програм і даних, порти вводу-виводу і блоки зі спеціальними функціями (лічильники, компаратори, АЦП та інші).

Використовується для керування електронними пристроями. По суті, це — однокристальний комп'ютер, здатний виконувати прості завдання. Використання однієї мікросхеми значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів.

Мікроконтролер AT89C2051

Методика програмування мікроконтролерів

Типи мікроконтролерів

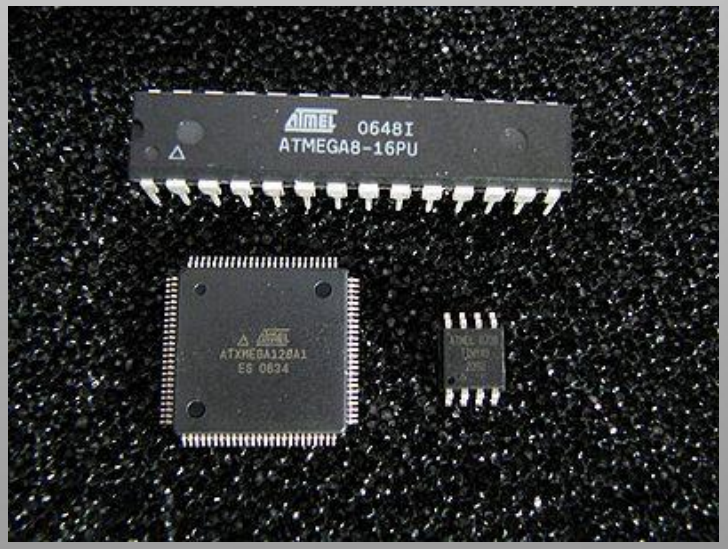
Є декілька архітектур мікропроцесорів:8051

68HC11
eZ8, eZ80
Hitachi H8, SuperH
Rabbit 2000
TLCS-870
MSP430 (16-біт)
CF (32-біти)
ARM
MIPS (32-біти PIC32)
S08
AVR
PIC (8-біт PIC16, PIC18, 16-біт dsPIC33 / PIC24)
V850
PowerPC ISE
PSoC (Programmable System-on-Chip)

Методика програмування мікроконтролерів

AVR — сімейство восьмибітових мікроконтролерів фірми Atmel.

Мікроконтролери сімейства AVR мають гарвардську архітектуру (програма і дані знаходяться в різних адресних просторах) і систему команд, близьку до ідеології RISC. Процесор AVR має 32 8-бітових регістра. Управління мікроконтролером, по суті, є управління цими регістрами. На відміну від «ідеального» RISC, регістри не абсолютно ортогональні: Три «здвоєні» 16-бітові регістри-покажчики X (r26:r27), Y (r28:r29) і Z (r30:r31) Деякі команди працюють тільки з регістрами r16.r31. Результат множення (у тих моделях, в яких є модуль множення) завжди поміщається в r0:r1



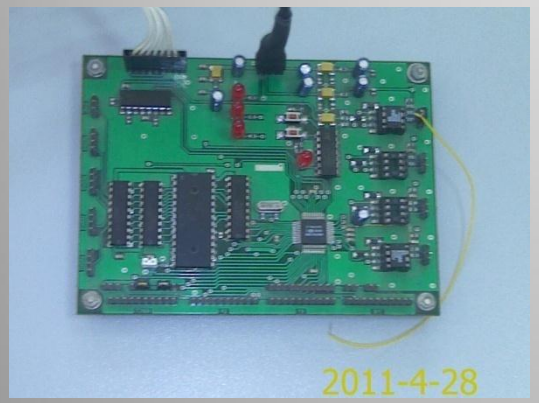
Методика програмування мікроконтролерів

Компілятори для мікроконтролерів компанії Atmel

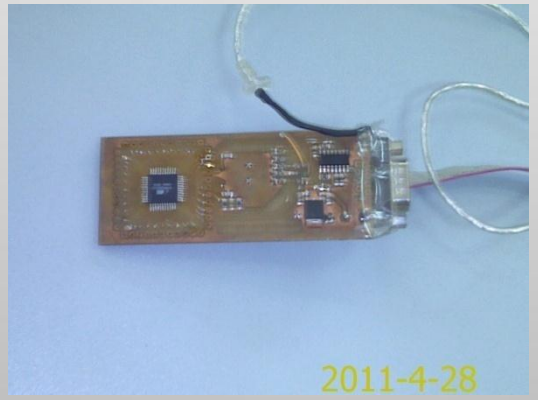
- AVR Studio
- IAR C Compiler
- Image Craft C Compiler
- Code Vision AVR C Compiler
- WinAVR.

Завдання

На запропонованих викладачем платах знайти мікроконтролери/DSP та встановити їх характеристики



Плата №1



Плата №2

Плата №3