УДК 371.315
DOI https://doi.org/10.32782/cusu-pmtp-2025-1-1

ПРОБЛЕМИ ОРГАНІЗАЦІЇ НАВЧАЛЬНОГО СЕРЕДОВИЩА ДЛЯ НИЗЬКОРІВНЕВОГО ПРОГРАМУВАННЯ

Баранюк Олександр Филимонович,
кандидат технічних наук, доцент,
доцент кафедри інформатики, програмування, штучного інтелекту та технологічної освіти
Центральноукраїнського державного університету імені Володимира Винниченка
ORCID ID: 0000-0003-1151-0092

У статті подано результати дослідження способів організації робочого середовища для навчання основ низькорівневого програмування. Наголошено на використанні простих і доступних засобів розробки програм. З огляду на припинення підтримки 16-розрядних програм, пропонується організувати робоче середовище на основі асемблера MASM32 з бібліотекою введення-виведення Irvine32. У курсі архітектури обчислювальних систем більша увага приділяється принципам функціонування обчислювальних систем та перетворення асемблерних програм у виконувані модулі.

Проаналізовано наукові публікації з питань організації навчання програмування початківців, навчання низькорівневого програмування та використання середовищ розробки в навчальному процесі. На основі аналізу сформульовано вимоги до редакторів / середовищ для програмування.

Навчання основ низькорівневого програмування пропонується здійснювати поетапно. На початкових етапах студенти ознайомлюються з організацією навчального середовища для програмування на основі обраного асемблера, засвоюють принцип трансляції програм за допомогою консольних команд та пакетних командних файлів. На наступних етапах пропонується використовувати текстові редактори з набором функцій підтримки програмування, автоматизації процесу компіляції та налагодження.

Згідно з вимогами до середовищ розробки програм, проаналізовано функціональні можливості простого редактора HiEditor та більш розвиненого редактора Notepad++. З'ясовано, що хоча розглянуті редактори не мають усіх бажаних функцій, проте їх можна додати шляхом створення команд користувача для виклику сторонніх програм, зокрема асемблера, компонувальника та налагоджувача. Аналіз можливостей професійного IDE Visual Studio показав, що його також можна використовувати для програмування мовою асемблера, але це потребує численних налаштувань.

Ключові слова: *низькорівневе програмування, асемблер, компілятор, компонувальник, бібліотека, редактор коду, інтегроване середовище розробки.*

Baraniuk Oleksandr. Problems of organizing the learning environment for low-level programming

The article presents the results of a study on methods of organizing a working environment for teaching the basics of low-level programming. Emphasis is placed on the use of simple and accessible software development tools. Due to the discontinuation of support for 16-bit programs, it is proposed to organize a working environment based on the MASM32 assembler with the Irvine32 input/output library. In the computer architecture course, more attention is paid to the principles of computer systems functioning and the assembler programs buildings.

Scientific publications on the organization of programming education for beginners, low-level programming training, and the use of development environments in the educational process were analyzed. Based on the analysis, requirements for editors/programming environments were formulated.

Teaching the fundamentals of low-level programming is proposed to be carried out in stages. At the initial stages, students become familiar with the organization of a programming learning environment based on the chosen assembler, learn the principle of program translation using console commands and batch command files. At the next stages, it is proposed to use text editors with a set of programming support functions, automation of the compilation and debugging process.

According to the requirements for software development environments, the functionality of the simple editor HiEditor and the more advanced editor Notepad++ were analyzed. It was found that although the considered editors do not have all the desired functions, they can be added by creating user commands to call third-party programs, including the assembler, linker, and debugger. An analysis of the capabilities of the professional IDE Visual Studio showed that it can also be used for assembler programming, but this requires numerous settings.

Key words*: low-level programming, assembler, compiler, linker, library, code editor, integrated development environment.*

Вступ. Розробникам програмного забезпечення потрібно добре розуміти архітектуру та принципи організації обчислювальної системи для написання ефективного коду. Сучасні освітні програми з комп'ютерних наук передбачають вивчення курсу «Архітектура обчислювальних систем». Остання версія навчальних програм з комп'ютерних наук (Computer Science Curricula 2023) від Спільної робочої групи з навчальних програм з комп'ютерних наук (The Joint Task Force on Computer Science Curricula) [5] розроблена спільними зусиллями Асоціації обчислювальної техніки (ACM), Комп'ютерного товариства IEEE (IEEE-CS) та Асоціації з розвитку штучного інтелекту (AAAI).

Модель знань навчальної програми CS-2023 містить 17 галузей знань (Knowledge Area), кожна з яких налічує блоки знань (Knowledge Unit). Галузь знань «Архітектура та організація» (AR) навчальної програми CS-2023 спрямована на розуміння загальних принципів організації апаратних платформ, на яких базуються всі комп'ютерні обчислення, та інтерфейсів, що надаються вищим рівням програмного забезпечення.

Блок знань «Організація машини рівня асемблера», серед іншого, передбачає знання системи команд процесора (зокрема, архітектури x86), форматів команд, груп команд маніпулювання даними, керування, введення / виведення, режимів адресації операндів, машинної мови та програмування мовою асемблера, розуміння механізмів виклику підпрограм та повернення з них, операцій введення / виведення та переривань.

Результатами навчання цього блоку знань можуть бути: розуміння представлення команд на рівні машини та в контексті символьного асемблера; порівняння шаблонів мови високого рівня з нотаціями мови асемблера; розуміння форматів команд змінної та фіксованої довжини; здатність до аналізу викликів підпрограм на рівні модуля та збірки; розуміння основних концепцій переривань та операцій введення / виведення; здатність писати прості програми мовою асемблера, зокрема для обробки рядків маніпулювання масивами.

Мову асемблера вивчають уже протягом майже чотирьох десятиліть, але її опанування залишається серйозним викликом для новачків. Студенти стикаються з істотними труднощами під час вивчення мов програмування низького рівня, що вимагає значних зусиль як від них самих, так і від викладачів. Найбільші труднощі виникають на початковому етапі, коли студенти повинні не лише опанувати синтаксис і семантику нової мови, а й розвинути навички розв'язання задач. Крім того, через апаратну залежність мови асемблера важливо володіти знаннями про архітектуру комп'ютера та периферійні пристрої. Ця вимога підкреслює необхідність комплексної підготовки, терпіння, наполегливості та здатності абстрактно мислити.

Аналіз досліджень і публікацій. Проблемам удосконалення процесу навчання мов програмування присвячено досить багато наукових розвідок вітчизняних та зарубіжних учених, як-от Л. В. Гришко, М. І. Жалдак, Н. В. Морзе, К. Ала-Мутка, А. Гомес, А. Робінс, Л. Вінслоу та багато інших [3; 6; 16]. Здебільшого ці дослідження присвячено питанням навчання мов програмування високого рівня, хоча деякі також проблемам навчання програмування мовою асемблера [4; 7; 9]. У роботі [1] як один із напрямів підвищення ефективності вивчення мови асемблера студентами-початківцями відзначається наявність адекватного середовища для розробки програм.

Інтегроване середовище розробки (IDE) – це програмний застосунок, який надає програмістам комплексні можливості для розробки програмного забезпечення [15]. Основні компоненти IDE – зручний редактор програмного коду, компілятор і налагоджувач.

Програмісти використовують IDE, оскільки це дає можливість правильно організовувати робочі процеси, швидко починати нові програмні проєкти, писати синтаксично правильний код, своєчасно виявляти та виправляти помилки в коді, компілювати й налагоджувати програми [15].

У роботі [12] автор розглядає інтегроване середовище як розширений редактор, оснащений убудованим компілятором та утилітами налагодження, що полегшує розробку програм. Додат-

ково IDE надає можливість переглядати файли, отримувати довідку про функції, коди помилок, спостерігати за змінними та об'єктами. До основних функцій IDE належать [12]:

− розпізнавання та виділення ключових слів мови й функцій, пропозиція доступних функцій під час уведення тексту, позначення помилок друкування;

− керування ресурсами (файли, бібліотеки) під час створення програм;

− інструменти налагодження для пошуку та виправлення помилок;

− наявність компілятора й компонувальника для трансляції тексту програми в об'єктний код і завантажувальний модуль цільової платформи.

Ґрунтовне дослідження особливостей IDE та наукових публікацій з теми, подане в роботі [10], показує, що часто автори зосереджуються на особливостях IDE без урахування потреб новачків. З метою виявлення вимог студентів до середовищ розробки програм група австрійських учених з Університету Інсбруку провела опитування, запропонувавши респондентам переважно першого року навчання з кількох університетів оцінити важливість та пріоритетність функцій IDE й зазначити, чи бракує їм деяких функцій. Серед іншого, запропоновано оцінити такі риси: підсвічування синтаксису кольором; наявність налагоджувача; автодоповнення коду; підсвічування кольором помилок; підсвічування парних дужок; підтримка гарячих клавіш; наявність компілятора; виділена консоль.

У результаті опитування учасників експерименту з'ясовано важливість та пріоритетність функцій IDE (табл. 1).

Таблиця 1

Результати аналізу важливості функцій IDE

1	Підсвічування синтаксису	Найбільша кількість оцінок «Дуже важливо», найбільш цінна функція серед опитаних учасників.
2	Підсвічування помилок	Вважається дуже важливою, забезпечує миттєве виявлення помилок без необхідності компіляції коду.
3	Компілятор / інтерпретатор	Функція вважається важливою, користувачі цінують легкий перехід від кодування до виконання.
4	Автодоповнення / пропозиція коду	Функція оцінена як дуже важлива, зменшує кількість помилок, пришвидшує пошук доступних функцій та методів.
5	Налагоджувач	Функція має високу оцінку, але меншу від інших. Очевидно, початківці не усвідомлюють важливості налагодження.
6	Підсвічування парних дужок	Функція важлива, рейтинг дещо нижчий, допомагає розуміти та підтримувати складні структури коду.
7	Комбінації клавіш	Нейтральні оцінки. Дехто вважає їх необхідними для швидкого кодування, інші – не суттєвими, залежить від особистих звичок.
8	Виділена консоль команд	Значна кількість низьких оцінок, хоча для певних завдань виділена консоль є важливою функцією.
Деякі респонденти вважають важливими *засоби навігації файлами* та *нумерацію рядків*		

Матеріали та методи. У процесі дослідження використовували асемблери Turbo Assembler TASM від Borland (TASM, TLINK, TASM32, TLINK32), Macro Assembler MASM від Microsoft (ML, LINK), середовище розробки MASM32 SDK, бібліотека підпрограм введення-виведення Irvine32, текстові редактори HiEditor, Notepad++, TextPad, середовища розробки програм WinAsm Studio, Visual Studio.

У роботі використано загальнонаукові методи дослідження: аналіз, синтез, порівняння та узагальнення. Метод аналізу використаний для огляду літературних джерел та виявлення характеристик редакторів і середовищ розробки для асемблера. Метод порівняння застосований до виявлених характеристик середовищ розробки з метою визначення оптимальних засобів розробки на кожному з етапів навчання. Метод синтезу використаний для визначення параметрів налаштування обраних середовищ програмування. Метод узагальнення застосовувався для виявлення суттєвих вимог до середовищ розробки та формулювання загальних висновків.

Результати. Зважаючи на обмежену кількість годин, які можна виокремити в курсі «Архітектура обчислювальних систем» на вивчення низькорівневого програмування, не ставиться за мету навчити студентів повноцінного програмування мовою асемблера. Натомість основна увага приділяється принципам обробки даних обчислювальними системами (системи числення, формати даних, будова, принцип дії та система команд процесора, способи адресації операндів, робота з масивами, реалізація типових високорівневих керівних структур мовою асемблера), структурі асемблерної програми та принципам побудови виконуваного модуля. Для реалізації поставлених завдань у стислі терміни потрібно надати студенту максимально просте робоче середовище, водночас не забуваючи про процес трансляції програми.

Програма, написана мовою асемблера, не може бути виконана безпосередньо [8]. Її потрібно транслювати (дослівно assemble – зібрати) у виконуваний код. Процес подібний до компіляції високорівневих програм мовами C, C++, хоча там процес побудови виконуваного модуля зазвичай прихований за лаштунками «магічної кнопки» Run більшості сучасних середовищ розробки.

Процес перетворення асемблерної програми містить кілька кроків, представлених схемою на рис. 1 [8].

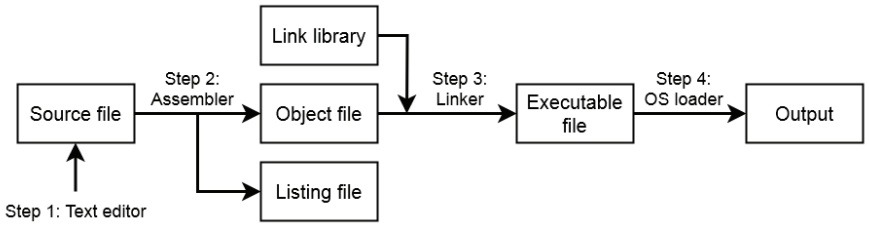

Рис. 1. Процес трансляції асемблерної програми

Спочатку текст програми перетворюється в об’єктний файл, що містить машинні коди процесора, але ще не готовий до запуску на виконання. Цей файл передається компонувальнику (linker), який пов’язує між собою один або кілька об’єктних модулів і бібліотечні файли, створюючи виконуваний файл, готовий до запуску з командного рядка операційної системи [8].

Щоб студенти добре зрозуміли механізм і послідовність компіляції асемблерної програми, пропонується в процесі виконання лабораторних робіт з низькорівневого програмування послідовно використати кілька різних засобів.

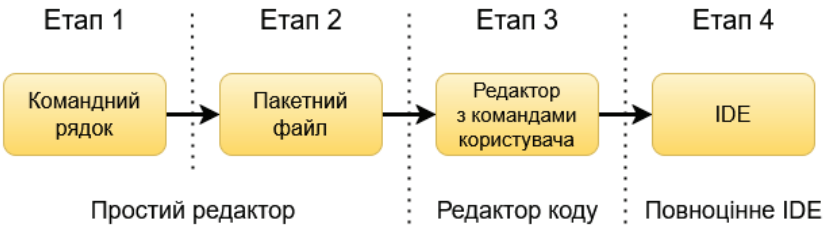

Рис. 2. Етапи виконання циклу лабораторних робіт

На початковому етапі студенти використовують найпростіший редактор коду й командний рядок для компіляції програми, щоб добре засвоїти організацію середовища й команди компіляції. На наступному етапі з метою спрощення побудови програми пропонується використати пакетний командний файл make32.bat. На третьому етапі можна більше зосередитися на про-

грамуванні й перейти до використання «просунутого» редактора коду (advanced editor), який має функції створення користувацьких команд для побудови асемблерної програми. Нарешті, на останньому етапі пропонується продемонструвати, як можна використати повнофункціональне (професійне) IDE з підтримкою проєктів, «магічною кнопкою», вбудованим налагоджувачем та іншими корисними засобами.

Розглянемо детальніше кожний з етапів за схемою: організація робочого середовища – редактор коду – побудова, запуск, налагодження програми.

Досить тривалий час для викладання основ низькорівневого програмування використовували простий і популярний свого часу компілятор TASM від Borland. Достатньо було покласти в каталог з програмою файли tasm.exe, tlink.exe, td.exe – і можна починати програмувати. Причому команди компіляції і компонування були дуже простими: tasm prog.asm і tlink prog.obj. Після цього можна запускати програму безпосередньо (prog) або через налагоджувач (td prog). Це були 16-розрядні програми, які сучасними 64-бітними платформами більше не підтримуються. Можна все ще продовжувати демонструвати 16-розрядне програмування, на віртуальних машинах із застарілими операційними системами або емуляторах DOS-режиму на зразок DOSBox, але це дуже мало схоже на сучасне програмування й не має жодного практичного застосування.

Перехід на 32-бітне програмування мовою асемблера значно ускладнює процес для початківців. З одного боку, є низка більш і менш відомих асемблерів (TASM32, MASM32, NASM та ін.). З іншого – написання навіть простої 32-бітної програми типу «Hello, world!» на асемблері потребує розуміння досить багатьох складних для початківців моментів. Асемблер – це мова машинних команд процесора, який не має засобів взаємодії з користувачем. Для доступу до апаратних ресурсів користувач повинен викликати функції операційної системи через прикладний програмний інтерфейс Win32 API. Для цього до програми варто під'єднати заголовкові файли з визначеннями функцій WinAPI, а компонувальнику вказати, де містяться файли системних бібліотек. Інакше кажучи, потрібно створити й налаштувати робоче середовище (каталоги, файли, шляхи, команди) для програмування мовою асемблера. Це непросте завдання для початківця.

Після багатьох пошуків і спроб організації робочих середовищ вирішено зупинитися на використанні компілятора MASM від Microsoft. Нині є і підтримується спільнотою незалежний проєкт The MASM32 SDK [14], який містить набір інструментів для побудови програм (компілятор, компонувальник, бібліотеки, макроси). MASM32 SDK – це «складна й вимоглива форма програмування, яка потребує високої точності кодування та гарного розуміння як мнемоніки Intel, так й архітектури процесора x86… MASM32 SDK призначений для досвідчених програмістів, які знайомі з написанням програмного забезпечення в 32-розрядних версіях Windows за допомогою інтерфейсу API… Він не дуже підходить для програмістів-початківців через передову технічну природу програмування на асемблері…» [14]. Хоча це досить громіздка система (понад 300 каталогів і 2 500 файлів) з примітивним редактором, проте вона є джерелом надійних і перевірених часом інструментів, як-от компілятор ml.exe та компонувальник link.exe.

Маючи асемблер та компонувальник, уже можна створювати програми, які будуть виконувати команди процесора. Проблема в тому, що процесор не має засобів виведення на екран, тож важко побачити результати роботи програми. Єдиним способом спостереження за роботою програми є запуск налагоджувача в покроковому режимі. Це незручно, адже цінність комп'ютера в тому, що він може вводити дані, обробляти їх і виводити результат, тобто більшість програм працює за шаблоном IPO (Input – Processing – Output).

Стандартної бібліотеки введення-виведення для MASM немає. У 16-бітних програмах часів MS-DOS для введення-виведення символів чи текстових рядків викликали функції DOS, відомі як програмні переривання INT 21h. Для виведення чисел потрібно було писати власні процедури, які перетворювали число в послідовність ASCII-кодів цифр числа. У 32-бітних Windows про-

грамах варто викликати функції операційної системи WinAPI. Але картина подібна – операційна система надає лише функції введення-виведення символів або рядків, до того ж функції WinAPI, досить складні у використанні, оскільки мають багато параметрів і типів даних Windows.

Полегшити діалог з користувачем можна завдяки стороннім бібліотекам введення-виведення, як-от UCR Standard Library від Рендалла Гайда, asmio16 [2], Irvine32 [8] та ін. Подібні бібліотеки містять низку функцій введення-виведення для знакових і беззнакових десяткових чисел, шістнадцяткових чисел та ін. Бібліотека Irvine32, створена автором чудової книги «Мова асемблера для процесорів x86» Кіпом Ірвіном [8], є у вільному доступі й містить десятки корисних функцій для початківців. Рекомендуємо використати її для організації введення-виведення.

Варто організувати робочий простір подібно до структури більшості професійних інструментів: у каталозі BIN – виконувані файли, у каталозі LIB – бібліотеки, у каталозі INCLUDE – файли заголовків. Файли користувача безпосередньо в кореневому каталозі MASM цієї структури. Так, мінімальний набір з бібліотекою Irvine32 містить 8 файлів у трьох каталогах.

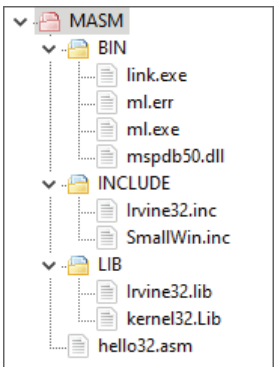

Рис. 3. Структура робочого простору для MASM

Етап 1. На першому етапі студентам важливо зрозуміти принцип побудови виконуваного модуля асемблерної програми. Для цього з консолі Windows потрібно подати команди компіляції та компонування:

```
ML -c -coff hello.asm
LINK hello.obj irvine32.lib kernel32.lib /SUBSYSTEM:CONSOLE
```

Щоб команди компіляції спрацювали, потрібно попередньо налаштувати змінні оточення PATH, INCLUDE, LIB такими командами (або помістити їх до командного файлу setpath.bat):

```
SET PATH=D:\MASM\BIN;%PATH%
SET INCLUDE=D:\MASM\INCLUDE
SET LIB=D:\MASM\LIB
```

Якщо передбачається використання налагоджувача, потрібно внести до виконуваного файлу додаткову інформацію, наприклад, так:

```
ML -Zi -c -Fl -Sg -coff hello.asm
LINK hello.obj irvine32.lib kernel32.lib /SUBSYSTEM:CONSOLE /DEBUG
```

Етап 2. На другому етапі студентам пропонується для компіляції програми використовувати наданий командний файл make32.bat, у якому вже містяться команди налаштування шляхів та побудови виконуваного файлу, тоді команда побудови програми зводиться до make32 hello, що значно спрощує задачу і дає змогу уникати помилок при введенні команд.

Невід'ємною частиною робочого процесу програміста є зручний тестовий редактор. На початкових етапах пропонується використовувати простий текстовий редактор з мінімально

необхідною функціональністю. Наголосимо, що редактор, подібний до Блокнота Windows, не підходить для програмування. Має бути, як мінімум, нумерація рядків і підсвічування синтаксису асемблера.

Вибираючи редактор / IDE для асемблера, використаємо вимоги, наведені вище в огляді літератури: підсвічування синтаксису / помилок, автодоповнення коду, компілятор / компонувальник, гарячі клавіші, виділена консоль, засоби файлової навігації.

Почнемо з простого, але досить функціонального редактора HiEditor. Він не потребує установки, після розпакування архіву готовий до роботи. Містить три файли (сам редактор HiEditor.exe, файл налаштувань HiEditor.ini і файл ключових слів KeyWords.hes) загальним обсягом 133 КБ.

Основні можливості HiEditor: 1) редактор має підсвічування синтаксису для асемблера MASM та деяких інших мов. Ключові слова асемблера (команди, директиви, регістри) та числові значення їх кольору зберігаються у файлі KeyWords.hes; 2) помилки в коді не виділяються, але ключові слова з помилками втрачають колір; 3) HiEditor не має автоматичного завершення чи підказок при введенні; 4) редактор не має вбудованого компілятора, але має команди компіляції для зовнішнього MASM32 або FASM; 5) HiEditor не має вбудованого налагоджувача, але може викликати зовнішній; 6) редактор не має можливості створення гарячих клавіш; 7) редактор не має виділеної консолі.

Тож редактор HiEditor – це простий мінімально достатній редактор для початківців з нумерацією рядків і підсвічуванням синтаксису. У нього бракує деяких функцій, проте Менеджер команд дає змогу створювати власні команди й додавати їх до меню Tools. Так можна створити команди для запуску зовнішнього налагоджувача та запуску програми в консолі Windows. Редактор не підтримує власних гарячих клавіш, але дає змогу присвоїти клавіші доступу до пунктів меню користувача, які викликаються через Alt. Редактор не має засобів навігації файлами, але підтримує роботу з кількома вкладками файлів.

Етап 3. Студенти розуміють механізм отримання виконуваного модуля програми. Можна перейти до третього етапу – використання «просунутого» редактора, який має змогу створювати й запускати на виконання команди користувача через меню, кнопки панелі інструментів або «гарячі» клавіші.

Нашу увагу привернули два редактори, придатні для комфортного програмування різними мовами (Notepad++ або TextPad), насправді їх значно більше. Студенти можуть обрати інший редактор, якщо мають досвід його використання й налаштування.

Notepad++ – це універсальний текстовий редактор, який підтримує деякі функції, необхідні для програмування мовою асемблера, зокрема номери рядків, підсвічування синтаксису, команди користувача, гарячі клавіші. Після установки його 12 каталогів з понад 120 файлів займають менш ніж 20 МБ. Розглянемо можливості редактора Notepad++ створювати асемблерні програми: 1) Notepad++ підтримує підсвічування синтаксису близько 90 мов програмування, зокрема асемблера. Є можливість додати підсвічування синтаксису будь-якої мови; 2) редактор не підтримує підсвічування синтаксичних помилок; 3) Notepad++ забезпечує автодоповнення ключових слів асемблера, однак не пропонує інтелектуального завершення коду або контекстних підказок; 4) редактор не містить вбудованого компілятора, але може викликати будь-які зовнішні компілятори; 5) Notepad++ не має вбудованого налагоджувача, але може викликати зовнішній налагоджувач; 6) редактор підтримує комбінації клавіш для команд користувача; 7) Notepad++ не має окремої консолі, але може викликати консоль Windows або виділену консоль плагіну NppExec.

Notepad++ має певні засоби навігації файловою системою та керування проєктами: панель «Тека як робоча область», Панелі проєктів та вкладки для кількох відкритих файлів. Панель «Тека як робоча область» відображає вміст обраного каталогу на диску у вигляді дерева. Панелі

проєкту дають змогу створити віртуальну папку й додати до неї файли та каталоги незалежно від їх реального розташування.

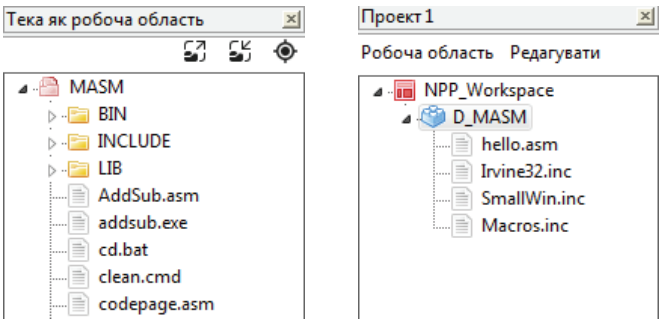

Рис. 4. Панелі навігації каталогу та проєкту Notepad++

Тож Notepad++ – це досить гарний вибір для створення програм будь-якою мовою програмування. У нього відсутні деякі характерні для IDE функції, проте через плагіни можна додати потрібний функціонал, наближаючи можливості до IDE. Так, плагін NPPExec дає змогу виконувати команди, запускати скрипти та інші програми просто з Notepad++ і надає вікно виведення (консоль NppExec). NppExec можна використати для виклику стороннього компілятора, компонувальника, налагоджувача, запуску побудованої програми.

Етап 4. Студенти навчилися створювати програми мовою асемблера й можуть зосередитися на програмуванні наявними засобами або використати повноцінне IDE, наприклад Visual Studio [13]. Виникає питання: чому одразу не почати програмувати в професійному середовищі? Річ у тому, що Visual Studio – це потужне середовище розробки для великих проєктів, а не для початківців, воно не має відповідного проєкту для асемблера, тож не передбачає створення програм на асемблері, хоч і має компілятор MASM. Після установки мінімального пакету для C++ займає на диску близько 3ГБ (2 500 каталогів з майже 20 000 файлів). Visual Studio можна пристосувати для кодування на асемблері шляхом відповідних налаштувань проєкту C++ [8] або навіть створення власного шаблону проєкту, але покрокові інструкції з налаштування співмірні з обсягом цієї статті. При цьому не вдається досягти всього необхідного для комфортної роботи, доводиться шукати сторонні розширення. Як незаперечну перевагу можна відзначити наявність досить пристойного вбудованого налагоджувача.

Результати порівняння розглянутих редакторів/IDE наведено в таблиці.

Таблиця 2

Порівняння функцій засобів розробки асемблерних програм

Показник	HiEditor	Notepad++	Visual Studio
Номери рядків	ПП	ПП	ПП
Підсвічування синтаксису	ПП	ПП	НП+
Підсвічування помилок	НП	НП	НП+
Автодоповнення	НП	ЧП	НП+
Файловий навігатор	НП	ЧП	ПП
Компілятор	НП+	НП+	ПП
Налагоджувач	НП+	НП+	ПП
Комбінації клавіш	ЧП	ПП	ПП
Виділена консоль	НП+	НП+	ПП
Кількість файлів / обсяг	3 / 133 КБ	123 / 19 МБ	19000 / 2,6 ГБ

ПП – повністю підтримує; ЧП – частково підтримує; НП – не підтримує; НП+ – не підтримує, але можна додати.

Висновки. Вивчення основ низькорівневого програмування в курсі «Ахітектура обчислювальних систем» потребує, з одного боку, максимально простого середовища для програмування й розуміння принципів побудови програм, з іншого – використання сучасних інструментів. Потужні інтегровані середовища розробки (IDE) призначені для швидкої та ефективної розробки великих проєктів мовами високого рівня, але не пристосовані для низькорівневого програмування й не розраховані на початківців.

Пропонується в межах виділених навчальних годин організувати поетапне знайомство студентів із засобами побудови асемблерних програм й основними концепціями програмування згідно з принципом системності та послідовності. На початку студенти організовують і налаштовують робоче середовище, кодують у простому текстовому редакторі й будують програму консольними командами, далі використовують командні файли для компіляції, потім переходять до роботи в розвиненому редакторі з інтегрованими засобами компіляції, а на останньому етапі можуть використати професійне середовище розробки. Це дає можливість засвоїти принципи побудови асемблерних програм із використанням сучасних підходів та засобів.

У подальшому дослідженні планується зосередитися на розробці навчальних завдань для кожного з етапів.

Література:

1. Баранюк О. Пошук шляхів підвищення ефективності вивчення мови асемблера. *Наукові записки. Серія : Проблеми методики фізико-математичної і технологічної освіти*. 2011. Вип. 2. С. 18–26.

2. Баранюк О. Розробка навчальної бібліотеки підпрограм для низькорівневого програмування. *Наукові записки. Серія : Проблеми методики фізико-математичної і технологічної освіти*. 2015. Вип. 7(2). С. 9–15.

3. Жалдак М. І. Проблеми інформатизації навчального процесу в середніх і вищих навчальних закладах. *Комп'ютер в школі та сім'ї*. 2013. № 3. С. 8–15.

4. Agarwal K. K., Agarwal A. Do we need a separate assembly language programming course? *Journal of Computing Sciences in Colleges*. 2004, № 19 (4). p. 246–251.

5. Computer Science Curricula 2023 / ACM; IEEE-CS; AAAI. New York: CM, 2023. 459 p. URL: https://dl.acm.org/doi/pdf/10.1145/3664191.

6. Gomes A., Mendes A.J. Learning to Program – Difficulties and Solutions. *International Conference on Engineering Education. ICEE*, 2007. p. 283–287.

7. Hyde R. Why Learning Assembly Language Is Still a Good Idea. 2004. URL: https://bixoft.nl/english/hyde.htm

8. Irvine K. Assembly language for x86 processors. 7th ed. Upper Saddle River : Pearson Education, 2015. 868 p.

9. Khan A. W. Mastering Assembly Language. 2024. URL: https://blog.ahmadwkhan.com/mastering-assembly-language

10. Requirements for an Online Integrated Development Environment for Automated Programming Assessment Systems / Frankford E., Crazzolara D., Sauerwein C. et al. *Proceedings of the 16th International Conference on Computer Supported Education: CSEDU*. 2024. Vol. 1. Angers : SciTePress. p. 305–313. DOI: 10.5220/0012556400003693

11. Robins A., Rountree J., Rountree N. Learning and Teaching Programming : A Review and Discussion. *Computer Science Education*. 2003. Vol. 13. No. 2. p. 137–172.

12. Satav S. K., Satpathy S. K., Satao K. J. A comparative study and critical analysis of various integrated development environments of C, C++, and Java languages for optimum development. *Universal Journal of Applied Computer Science and Technology*. 2011. No. 1. p. 9–15.

13. Sotnik S., Lyashenko V., Schakurova T. Modern Integrated Software Development Environments *International Journal of Academic and Applied Research (IJAAR)*. 2021. Vol. 5(10). p. 157–161. URL: https://openarchive.nure.ua/handle/document/18022

14. The MASM32 SDK Uncompromised capacity for the professional programmer. URL: https://masm32.com

15. Timbó R. Integrated Development Environments: Definition, What Is, and Purpose. 2024. https://www.revelo.com/blog/integrated-development-environments.

16. Winslow L. Programming Pedagogy : A Psychological Overview. *SIGCSE Bulletin*. 1996. № 28 (3). p. 17–22.

References:

1. Baraniuk, O. (2011). Poshuk shliakhiv pidvyshchennia efektyvnosti vyvchennia movy asemblera [Finding ways to improve the efficiency of assembly language learning]. *Naukovi zapysky. Seriia : Problemy metodyky fizyko-matematychnoi i tekhnolohichnoi osvity*. 2, 18–26. [in Ukrainian]

2. Baraniuk, O. (2015). Rozrobka navchalnoi biblioteky pidprohram dlia nyzkorivnevoho prohramuvannia [Development of a training library of subroutines for low-level programming]. *Naukovi zapysky. Seriia : Problemy metodyky fizyko-matematychnoi i tekhnolohichnoi osvity*. 7(2), 9–15. [in Ukrainian]

3. Zhaldak, M.I. (2013). Problemy informatyzatsii navchalnoho protsesu v serednikh i vyshchykh navchalnykh zakladakh [Problems of informatization of the educational process in secondary and higher educational institutions]. *Kompiuter v shkoli ta simi*, (3), 8–15. [in Ukrainian]

4. Agarwal, K., & Agarwal, A. (2004). Do we need a separate assembly language programming course? *Journal of Computing Sciences in Colleges*. 19(4), 246–251 [in English].

5. ACM; IEEE-CS; AAAI (2023). Computer Science Curricula 2023. New York: CM. Retrieved from: https://dl.acm.org/doi/pdf/10.1145/3664191 [in English].

6. Gomes, A., & Mendes, A.J. (2007). Learning to Program – Difficulties and Solutions. *International Conference on Engineering Education. ICEE*, 283–287 [in English].

7. Hyde, R. (2004). Why Learning Assembly Language Is Still a Good Idea. Retrieved from: https://bixoft.nl/english/hyde.htm [in English].

8. Irvine, K. (2015). Assembly language for x86 processors. 7th ed. Pearson Education [in English].

9. Khan, A. W. (2024). Mastering Assembly Language. Retrieved from: https://blog.ahmadwkhan.com/mastering-assembly-language [in English].

10. Frankford, E., Crazzolara, D., & Sauerwein, C. et al. (2024). Requirements for an Online Integrated Development Environment for Automated Programming Assessment Systems. *Proceedings of the 16th International Conference on Computer Supported Education*. SciTePress. 1, 305–313. DOI: 10.5220/0012556400003693 [in English].

11. Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming : A Review and Discussion. *Computer Science Education*. 13(2), 137–172 [in English].

12. Satav, S. K., Satpathy, S. K., & Satao, K. J. (2011). A comparative study and critical analysis of various integrated development environments of C, C++, and Java languages for optimum development. *Universal Journal of Applied Computer Science and Technology*, 1, 9–15 [in English].

13. Sotnik, S., Lyashenko, V., & Schakurova, T. (2021). Modern Integrated Software Development Environments. *International Journal of Academic and Applied Research (IJAAR)*. 5(10), 157–161. Retrieved from: https://openarchive.nure.ua/handle/document/18022 [in English].

14. The MASM32 SDK Uncompromised capacity for the professional programmer. Retrieved from: https://masm32.com [in English].

15. Timbó, R. (2024). Integrated Development Environments: Definition, What Is, and Purpose. Retrieved from: https://www.revelo.com/blog/integrated-development-environments [in English].

16. Winslow, L. (1996). Programming Pedagogy : A Psychological Overview. *SIGCSE Bulletin*. 28(3), 17–22 [in English].