# ВСТУП

З метою підвищення рівня підготовки програмістів в області електроніки, ознайомлення з принципами побудови і функціонування процесорів на апаратному рівні, а також для формування уявлення про класичну архітектуру мікропроцесорів і мовами програмування низького рівня кафедрою електроніки та управляючих систем Харківського національного університету імені В.Н. Каразіна за участю студентів старших курсів була розроблена навчальна модель цифрового процесора. Вона побудована на мікросхемах малої ступеня інтеграції, дозволяє продемонструвати процес виконання програми на рівні окремих логічних елементів і оброблюваних ними сигналів [1].

Трохи пізніше на кафедрі був створений інтерфейсний модуль, який дозволив підключати макет процесора до персонального комп'ютера для моніторингу його стану і управління роботою в реальному часі. Завдяки розробці цього інтерфейсу і супутнього програмного забезпечення з'явилася можливість:

* завантажувати в оперативну пам'ять більш об'ємні програми, отримані шляхом трансляції з мови асемблер;
* відображати на екрані монітора поточний стан моделі процесора, вміст регістрів, пам'яті, стан шин;
* показувати хід виконання окремих команд;
* керувати роботою процесора як на рівні виконання окремих машинних циклів, так і на рівні виконання підпрограм.

Всі ці можливості істотно поліпшують наочність макета і дозволяють більш ефективно використовувати його в навчальному процесі.

Взаємодія моделі процесора з комп'ютером через створений інтерфейс здійснюється через паралельний восьмирозрядний LPT-порт [2]. Передача даних з комп'ютера в макет виконується по паралельній восьмирозрядній шині, а читання стану функціональних вузлів моделі проводиться напівбайтами за допомогою системи мультиплексорів і з використанням службових регістрів LPT-порту. При організації обміну, програма, безпосередньо, управляє роботою як регістрів даних, так і вхідних і вихідних службових регістрів в складі порту. Паралельна передача даних забезпечує високу швидкодію системи в цілому.

Модель цифрового процесора активно використовується в навчальному процесі на двох факультетах – комп’ютерних наук і фізико-технічному, Харківського національного університету імені Каразіна. Вона викликає величезний інтерес у студентів під час проведення лабораторних робіт і при активній участі ведеться її вдосконалення. Зокрема один студент розробив для даної моделі додатковий модуль оперативної пам’яті ємністю 32 кілобайт, яка істотно розширює можливості самої моделі.

Конструкція інтерфейсів регістрів (портів) дозволила на пряму підключати до моделі процесора рідкокристалічний індикатор і клавіатуру, перетворивши модель на найпростішу модель комп’ютера.

Декілька років тому було розроблено програму, яка передбачає можливість читання і запису виконуваного коду, функції зберігання підготовлених програм на диску, яка дає можливість виконання, моніторингу і індикації стану процесора. Але функціональність бажала кращого. В неї було декілька принципових помилок, що час від часу викликали некоректну роботу програми, також вона застаріла для нових операційних систем. Але головною проблемою було те, що вона не давала можливість працювати з пам’яттю 256 байт, що значно звужувало можливості даної програми.

В своїй бакалаврській дипломній роботі я почала розробляти програму, яка буде відповідати всім вимогам, коректно працювати, з ергономічним дизайном, зрозумілим для користувача. В результаті було створено дизайн інтерфейсу і розроблено сам інтерфейс користувача, який протестований на різних операційних системах і комп’ютерах. Результат тестування показав, що створена інтерфейсна програма являється кросбраузерною.

Але не вирішена головна задача. Так як за допомогою мови JavaScript, на якій був розроблений інтерфейс користувача, на сьогоднішній день неможливо управляти апаратними ресурсами комп’ютера, то потрібно розробити ще одну програму, яка матиме таку можливість, а також буде кросс-платформною. Но зміна програмного забезпечення неможлива без розробки нового інтерфейсного модуля.

## Проблематика

Робота з моделлю здійснювалася при використанні операційних систем, що надають можливість доступу до апаратних ресурсів комп'ютера (із серії операційних систем Windows робота була можлива з версіями Windows 98, Windows Vista, Windows XP).

Але на сьогоднішній день всі перераховані вище операційний системи не підтримуються компанією Майкрософт і поступово витісняються з використання. На зміну їм приходять більш нові версії, такі як Windows 7, Windows 8, Windows 10. У всіх цих системах з метою забезпечення безпеки та їх стійкості до шкідливих програм максимально обмежений доступ до апаратних ресурсів комп'ютера. Це робить неможливим використання раніше створеної програми з управління моделлю процесора, так як в існуючій версії передача даних здійснюється через службові регістри зовнішнього порту комп'ютера. Скоріш за все, наступні операційні системи будуть ще більше обмежувати можливості прямого управління портами без використання спеціально розроблених драйверів, які передбачають відповідні правила доступу і обмеження повноважень.

Сучасна тенденція розвитку персональних комп'ютерів визначена в першу чергу комерційними інтересами виробників і орієнтована на задоволення потреб найбільш широкого контингенту користувачів. Це, в першу чергу, - реалізація комунікативних, мультимедійних та ігрових функцій. З цієї причини в сучасних комп'ютерах «зникли» паралельні порти введення-виведення. Передача даних в основному здійснюється через USB-порти, а також через порти, в основу роботи яких покладено радіочастотний принцип передачі інформації (технології Bluetooth і Wi Fi). В результаті створене раніше обладнання неможливо підключити до сучасних ноутбуків і стаціонарних персональних комп'ютерів.

У зв'язку з цим виникла необхідність для подальшого успішного використання моделі процесора розробити новий модуль апаратного інтерфейсу і нове програмне забезпечення, яке дозволило б управляти апаратурою при роботі з сучасними комп'ютерами в операційних системах Windows 7 ... Windows 10.

## Мета роботи

Метою дипломної роботи є дослідження та аналіз методів організації взаємодії в комп’ютерних системах управління та розробка на результатах цього аналізу нового апаратно-програмного комплексу для моніторингу та управління моделлю цифрового процесора.

## Галузь застосування

Галузь застосування даного дослідження є учбовий процес з підготовки фахівців по програмуванню та електроніці на технічних факультетах Харківського національного університету імені Каразіна.

# РОЗДІЛ 1. Опис об’єкту та аналіз можливостей управління

## 1.1 Опис можливостей інтерфейсного модуля

При розробці моделі процесору в її конструкції був передбачений роз’єднувач для підключення комп’ютерного інтерфейсу. На контакти цього роз’єднувача виведено:

* внутрішня шина даних процессора;
* шина адреси;
* сигнали управління оперативно-запам'ятовуючим пристроєм (ОЗП);
* вихідні шини регістрів арифметико-логічного пристрою (АЛП);
* шина командного слова;
* основні сигнали управління процесора;
* сигнали, які передають стан признакових бітів.

Завдяки такій апаратній організації існує можливість повного управління роботою ОЗП: заносити до нього код програми, зчитувати та очищати пам’ять. Окрім того є можливим безпосереднього управління усіма апаратними процесами, що відбуваються в моделі. Комп’ютер має можливість зчитувати поточний стан усіх шин та основних регістрів процесору і за допомогою відповідного програмного забезпечення відображати на моніторі [3].

Контроль стану учбового обладнання та управління ним здійснюється через паралельні шини передачі даних саме тому існуючий інтерфейсний модуль було побудовано на основі паралельного комп’ютерного інтерфейсу CENTRONIX, який нажаль на сьогодні не використовується в сучасних комп’ютерів.

Тому однією з основних задач даного дослідження є пошук схемних та програмних рішень, які могли б замінити існуюче обладнання на таке, яке зараз розповсюджене у сучасних комп’ютерах і яке буде використовуватись у майбутньому (можно описать и нарисовать старий интерфейсній модуль).

## 1.2 Формулювання научно-технічних завдань

По-перше, на основі детального аналізу технічних характеристик та можливостей комунікаційних портів комп’ютера і вибрати порт, що буде задовольняти усім вимогам для організації зв’язку з апаратурою, при використанні якого не буде проблем з взаємодією з будь-якої операційної системи.

По-друге, необхідно розробити протокол взаємодії комп'ютера з керованим пристроєм і систему команд, які б не вимагали безпосереднього доступу до апаратних регістрів порту, а грунтувалися на передачу даних з використанням уніфікованих протоколів обміну, підтримуваних операційними системами і вбудованими системними драйверами.

По-третє, при виборі технології розробки програмного забезпечення слід орієнтуватися на впровадження принципів кроссплатформності, щоб забезпечити працездатність керуючої програми під різними операційними системами.

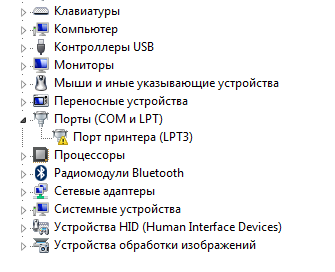
# РОЗДІЛ 2. Обґрунтування і вибір теоретичних та експериментальних методів дослідження поставлених задач

Щоб взаємодіяти з моделлю цифрового процесору було проведено дослідження можливостей управління паралельними та послідовними портами комп’ютера. Вивчено існуючі програмні технології для можливості організації такого програмного доступу. Проведено експериментальне дослідження різноманітних варіантів передачі даних до управляючої системи та аналіз отриманих результатів.

Для організації зв'язку з апаратурою необхідно орієнтуватися на застосування найбільш використовуваних на сьогоднішній день в комп'ютерах і захищених від впливу електромагнітних завад USB-портів. Але у них є недоліки:

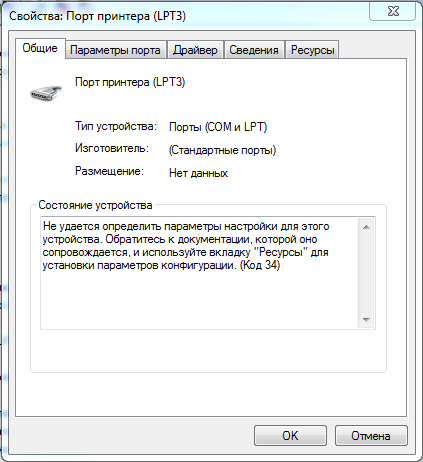
* коротку відстань передачі даних;
* передача через USB-порт здійснюється в послідовному коді, отже, для управління макетом необхідно пристрій декодування, що забезпечує запис даних в паралельні регістри і здійснює зворотне перетворення даних.

Так як сучасні ноутбуки та комп’ютери вже не обладнані LPT-портом, спробувала через адаптер USB-LPT приєднатися до макету. Але спроба робити з цим адаптером не дала позитивного результату, тому що він призначений для роботи за принтером і підтримує відповідний протокол взаємодії. В диспетчері пристроїв він відображався з помилкою (Рисунок 2.1).



*Рисунок 2.1 – Спроба підключення до віртуального LPT-порту невдала*

А в описі стану пристрою було сказано, що не має можливості визначити параметри налаштування(Рисунок 2.2).



*Рисунок 2.2 – Не вдається визначити параметри налаштування*

Це означає, що використовувати даний адаптер без переробки програмного забезпечення внутрішнього контролеру адаптеру неможливо. Тому що відсутній доступ до регістрів створеного віртуального LPT-порта.

Була також спроба використати COM-порт, який організовується при підключенні адаптера COM-USB. Так як інтерфейс для моделі має тільки LPT-порт, для данного эксперименту був використаний технологічний замикач для COM-порта, який передає на вхід порту те інформаційне повідомлення, яке було відправлене через цей порт. За допомогою спеціального програмного забезпечення можна було переконатися, що існує можливість як передачі так і прийому повідомлення.

Отже, в якості найбільш зручної технології вибрано використання віртуального COM-порту [4]. Також це дає нам можливість віддалити пристрій на відстань до декількох десятків, а то і сотень метрів, хоча це призведе до певного зниження швидкості обміну інформацією. А в майбутньому, при відповідному переробці апаратного інтерфейсу, можна перейти на протокол бездротового зв'язку і використовувати один з радіоканалів, наприклад, Bluetooth або Wi-Fi.

Як пристрій для перетворення даних з послідовного формату в паралельний і назад можна застосувати мікроконтролер або ж розробити мікропрограмний автомат на основі програмованої логічної матриці.

# РОЗДІЛ 3. Розробка системних узгоджень, програмних та технічних складових системи

## 3.1 Структура інтерфейсного модуля

Принцип побудови двонаправленого каналу передачі даних, який пов'язує комп'ютер з моделлю цифрового процесора, показаний на Рисунку 3.1.1.



Рисунок 3.1.1 – Структура каналу передачі даних

Комп'ютер виконує обмін інформацією з зовнішнім пристроєм за допомогою одного з вбудованих USB-портів, до якого приєднаний адаптер USB / COM. Послідовна лінія передачі даних, що зв'язує адаптер з інтерфейсним модулем, виконана в стандарті RS232. Сам інтерфейсний модуль безпосередньо приєднується до моделі процесора через передбачений конструкцією моделі роз'єм.

Взаємодія модуля з моделлю процесора здійснюється паралельними шинами і окремими дискретними сигналами. Для управління і контролю стану моделі необхідно забезпечити формування п'яти керуючих сигналів, читання чотирьох однобайтовим довічних слів, а також організувати формування двонапрямлених восьмирозрядних шин для читання / запису адреси і даних при взаємодії з оперативною пам'яттю. Узагальнена структурна схема багатофункціонального модуля показана на Рисунку 3.1.2.



Рисунок. 3.1.2 – Структурна схема інтерфейсного модуля

Інтерфейс дозволяє через мультиплексор зчитувати стан регістрів А і В, стан регістра команд, а також стан окремих сигналів управління, які виробляються генератором сигналів машинного циклу. Мультиплексор також забезпечує доступ по читанню шини адреси і шини даних.

При роботі з модулем пам'яті контроль шини адреси і шини даних здійснюється безпосередньо з портів мікроконтролера. Окремими дискретними сигналами забезпечено формування сигналів читання і запису, а також сигналу Reset і імпульсів тактової частоти. Частина сигналів мікроконтролера використовується для управління мультиплексором.

## 3.2 Протокол взаємодії комп'ютера з моделлю цифрового процесора

Для того щоб створити зручний та функціональний протокол, виділимо основні критерії, яким він повинен відповідати:

* правильне асоціативне сприйняття синтаксису команд оператором;
* інформації повинна бути представлена у вигляді ASCII кодів, але максимально коротких синтаксичних форм, щоб добитися найменшого зниження швидкості обміну;
* можливість майбутнього доопрацювання протоколу з метою розширення його функціональності.

## 3.2.1 Правильне асоціативне сприйняття синтаксису команд

У випадку, коли взаємодія з моделлю цифрового процесору буде виконуватись не через спеціальну програму, а, наприклад, через консоль оператором, щоб він (оператор) довго не вивчав різні довідники по правильному написанню команди, слід синтаксис команди зробити не в машинних кодах, типу 002, 0А5, а зрозумілими для людини словами – read, write і так далі. Це значно покращить сприйняття синтаксису оператором, прискорить швидкість роботи з протоколом, а сам протокол буде більш ергономічний, тим самим з ним буде приємно, легко працювати.

## 3.2.2 Інформації у ASCII кодах

Протокол повинен бути націлений на процес передачі даних, наприклад, як в принтерах висилаються потоки даних, і не були задіяні службові регістри апаратних портів, як спочатку було при роботі з макетом. Зараз же ми можемо використовувати тільки стандартні протоколи обміну.

Якщо подивитися на взаємодію комп'ютера з принтером, то як правило принтери підключають до комп'ютерів через паралельний порт. Паралельна передача даних – це переміщення декількох бітів інформації в одному циклі. Ширше тракт, по якому переміщаються дані на принтер або з нього.

Нові операційні системи підтримують USB PnP. При установці пристрою USB до комп'ютерної системи, що підтримує PnP, його виявлення проводиться автоматично, і починається процес установки драйвера.

Драйвер − це системна програма, яка повністю управляє принтером. Драйвер приймає команди та текст для друку від операційної системи і додатків, переводить команду в формат, який розуміє принтер, а потім передає їх йому. Принтер, отримавши команду виконує її, повертає результат драйверу, що потім переказує цю команду на «мову» операційний системи і видає результати користувачеві.

Як бачимо, з боку програми створюється тільки потік переданих даних, і немає необхідності в доступі до керуючих регістрів. А потік даних забезпечується стандартними драйверами Windows.

Щоб вирішити дану проблему, перш за все, треба відмовитися від службових регістрів апаратних портів, в незалежності, які порти будуть використовуватися і перейти до уніфікованому протоколу передачі даних в зовнішні пристрої.

Для передачі інформації з комп'ютера на модель процесора використовується текстовий формат команд. Це виключає появу в переданих пакетах спеціальних символів, які можуть викликати непередбачену роботу драйверів і системних служб Windows при спробі передачі бінарних значень.

Таким чином, при переході до передачі текстових файлів зникає необхідність в доступі до керуючих регістрів, а передача даних може здійснюватися по послідовному каналу з використанням тільки двох сигнальних проводів TXD і RXD - з використанням Z-модемного з'єднання. Прийом і передачу інформації здійснює мікроконтролер інтерфейсного модуля, він же виконує перетворення команд в паралельний код для безпосередньо взаємодії з регістрами моделі процесора.

Важливим фактором є довжина повідомлення. Так як вирішено використовувати символи ASCII кодів, тим самим ми збільшуємо об’єм передачі даних, тому що кожен символ – це один байт. Тобто, команда, наприклад, WRITE − займає аж 5 байт, це тільки командне слово, а ще можуть бути параметри, що записувати і куди, тим самим команда буде дуже об’ємна і значно знизить швидкість передачі. Тому слід зробити команду лаконічну, щоб не перезавантажувати канал обміну, наприклад WR, по першим літерам можна зрозуміти що це команда для запису і в цьому випадку командне слово буз займати всього лише 2 символа. Але не забувати про перший критерій для протоколу − асоціативне сприйняття команди оператором.

## 3.2.3 Розширення протоколу

Протокол не повинен бути націлений тільки на дану реалізацію моделі цифрового процесору. У майбутньому її можуть вдосконалювати, і щоб не створювати новий протокол, треба продумати основні його положення і синтаксис команд таким чином, щоб передбачити можливі доповнення у майбутньому.

## 3.3.1 Опис протоколу упавління приладами та обладнаннями ICSP (Istrument Control Special Protocol)

Відповідно до прийнятої тенденцією рішення задачі був розроблений детальний протокол взаємодії між комп'ютером і мікропроцесорним пристроєм, на базі якого і спроектований інтерфейсний модуль.

## 3.3.2 Призначення

Протокол IСSР регламентує обмін між управляючим комп’ютером та програмно керованими пристроями, які під’єднані до нього за допомогою послідовних каналів зв’язку (наприклад, через інтерфейси RS485, RS232, USB та їм подібні). Протокол створено для забезпечення можливості мультиплатформенного управління пристроями з використанням вбудованих в операційні системи драйверів доступу до апаратних портів комп’ютера, якщо вони відповідають вимогам специфікації XMODEM(протокол телетайпного обмена) [5].

Ця версія протоколу орієнтована в першу чергу на обмін числовими даними, якими можуть бути адреси, байти даних, багатобайтові слова та масиви чисел. Але протокол може бути легко модифікований та розширений для передачі текстових повідомлень за рахунок введення відповідних ключових слів або формування додаткових правил запису текстових повідомлень.

## 3.3.3 Основні положення

1. Протокол IСSР передбачає організацію взаємодії між комп’ютером та керованою системою за принципом «запит–відповідь», роль ведучої станції, яка завжди є ініціатором обміну, виконує управляючий компьютер. Обмін інформацією здійснюється з обмеженим використанням інтерфейсних засобів, тобто відповідає вимогам Z-модему [6].

2.  До одного апаратного порту комп’ютера одночасно може бути під’єднано не більше одного керованого пристрою, так як протокол не передбачає адресного звернення до абонента.

3.  Обмін інформацією здійснюється повідомленнями, загальний об’єм яких не повинен перевищувати 1024 байта.

## 3.3.4 Синтаксис повідомлення

1.  Повідомлення передається у текстовому форматі та являє собою послідовність символів, які представлені ASCII-кодами в діапазоні значень від 020h до 05Fh. Тобто, до тексту повідомлення можуть входити цифри, великі та малі літери латинського алфавіту, пробіл, знаки пунктуації та деякі спеціальні символи, коди яких потрапляють до вказаного діапазону. Не допускається використання псевдографічних символів і символів кирилиці (ASCII > 05Fh) та управляючих символів, для яких значення ASCII < 020h.

2.  Програми, що формують та виконують обробку повідомлень, мають бути нечутливого до регістру літер латинського алфавіту, тобто, вживання великих та малих літер у повідомленні є рівноцінним.

3.  Повідомлення завжди починається зі слова-ідентифікатора: команди, запиту, ознаки формату чи підтвердження відповіді, тощо. Останнім символом повідомлення (тобто, ознакою його закінчення) завжди має бути символом «;» (ASCII = 03Bh).

4.  Слово-ідентифікатор обов’язково повинно починатися з великої або малої літери латинського алфавіту та може включати в себе всі дозволені  п. 3.1 символи за винятком пробілу (ASCII = 020h) та символу «;» (ASCII = 03Bh).

5.  Окрім слова-ідентифікатора до складу повідомлення можуть входити параметри команди –– додаткові дані для командного слова. Їх може бути від 0 до 256. Параметри відокремлюються від командного слова та один від одного символом пробілу (ASCII = 020h). Після останнього параметру ставиться символ закінчення команди «;» (ASCII = 03Bh).

6. Всі числа, що входять до складу слова ідентифікатора та параметри – передаються символами, що відповідають їх запису в шістнадцятирічній системі числення.

Приклад команди представлений на Рисунку 3.3.4.1.

Командне слово

Параметри команди

Кінець команди

WR-M address data;

*Рисунок 3.3.4.1Синтаксис команди*

Можна зробити висновок, що даний синтаксис повідомлення сильно не обмежує в написанні нових команд. Вважаю, що символів в діапазоні значень від 020h до 05Fh ASCII-таблиці буде достатньо, щоб описати команду, а використання псевдографічних символів і символів кирилиці (ASCII > 05Fh) та управляючих символів, для яких значення ASCII < 020h, забезпечать впевненість в тому, що ми, випадково, не напишемо системну команду операційної системи, яка при конвертації може виконатись.

## 3.3.5 Вимоги до формату передачі даних

1. Швидкість обміну повідомлень становить 9600 бот.

2. COM-порт повинен бути налаштований наступним чином:

* + - посилка починається зі стартового біта;
    - кількість бітів даних становить 8;
    - біт парітету відсутній;
    - завершується посилка 1 стоп-бітом.

3. Час очікування відповіді - 30 мілісекунд.

## 3.4 Процес встановлення зв’язку з зовнішніми пристроями

1. Для початку програма, яка здійснює підключення до COM-порта, повина запросити в операційної системи перелік комунікаційних портів, які входять до складу компьютера.

2. Згідно з протоколом ICSP v1 запросити тип у кожного пристрою;

3. Якщо пристрій відповів коректно:

3.1 Налаштовати швидкість обміну повідомлення, яка вказана в п.1. вимог до формату передачі даних;

3.2 Намагаєтися зєднантися до кожного пристрою, з вказаною швидкістю;

4. Якщо з’єднання пройшло успішно, проходить перевірка версії;

5. При коректній версії, починається обмін даними між пристоями;

6. Якцо на якомусь етапі не будо отримано віповіді за час вказаний в п.3. вимог до формату передачі даних;

6.1, відправляється повторний запит. І якщо повторної відповіді не отримано, з’єднання вважаеться невдалим.

## 3.5 Створення системи команд

Для аналізу можливості моніторингу за станом моделі процесору та побудови системних рішень, було проведено аналіз принципової схеми інтерфейсного модуля та складено таблицю (Таблиця 3.5.1) функціональних процедур для роботи с пристроєм.

Таблиця 3.5.1 - Функціональні процедури інтерфейсного модуля

|  |  |  |
| --- | --- | --- |
| **Команди управління моделлю процесора** | **Команди контролю стану вузлів процесора** | **Команди для роботи з оперативною пам’яттю** |
| * Процедура ініціалізації * Формування перезапуску процесора * Формування тактового імпульсу | * Читання стану шини даних * Читання стану шини адреси * Читання стану регістра А * Читання стану регістра В * Читання командного слова * Читання керуючих сигналів * Читання стану окремих бітів | * Очистка програмної пам'яті * Запис даних в програмну пам'ять * Читання з зазначеної адреси програмної пам'яті |

За допомогою цієї таблиці я маю представлення, які команди можна відправляти на інтерфейсний модуль, і легко зможу написати систему команд.

## Перший варіант системи команд

В Таблиці 3.5.2 представлено фрагмент першого варіанту системи команд.

*Таблиця 3.5.2 Перший варіант системи команд*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ пп** | **Синтаксис команди** | **Опис** | **Приклад** | **Відповідь** | **Опис відповіді** |
| **Команди для роботи з оперативною пам’ятю** | | | | | |
| **1.** | WR-M- ADD BYTE; | WRite in Memory by ADdress follow DATA.  Командна записує інформацію в пам'ять макету. Записати можна або один байт, або блок інформації. Якщо записується блок, то символи розділяються space(пробіл). | WR-M-147 1; - запис в пам'ять по адресі 147 значення 1.  WR-M-147 1 2; - запис в пам’ять з адреси 147 значення 1, 2. | WR-M-AD DATA-OK; | Операція виконана вдало. |
| ERR: AD-address WR-write\_info RD-read\_info | Помилка: в адресі «address» записувалось «write\_info», а при перевірці – отримали «read\_info» |
| **2.** | CLR-M; | CLeaR Memory.  Команда генерує очистку пам’яті макету. | **-** | CLR-ОК; | Очистка пройшла успішно. |
| ERR: CLR-M AD-address DATA; | Помилка: при очистці пам’яті, в адресі «address» знайдена така інформація «DATA». |
| **3.** | RD-M-AD N; | ReaD from Memory from ADdress and Number of information.  Читання інформацію з пам’яті з вказаної адреси(AD) вказану кількість елементів(N). | RD-M-147 10;  Прочитати з пам’яті , поз адреси 147 10 элементів. | RD-M-AD N; | Після завершення операції отримуємо масив даних розміром N. |

В першому стовпці даної таблиці представлено номер по порядку команд; в другому – синтаксис команди, а в третьому опис, що робить дана команда; в четвертому показано приклад, в п’ятому і шостому – відповіді і їх опис, які можливі при виконанні даної команди.

Після додаткового аналізу та спілкуванні с фахівцями в галузі програмування, було прийнято рішення ці команди оптимізувати, прибравши символ «-», адже значної ролі він не відігравав, а це ще додаткові байти при відправленні даної інформації.

## Система команд

В додатку А представлено повна таблиця системи команд. Вона розділена на три основні частини:

* команди для роботи з оперативною пам’яттю;
* команди контролю стану вузлів процесора;
* команди управління моделлю процесора.

## Команди для роботи з оперативною пам’яттю

Даний блок містить команди для роботи з оперативною пам’яттю моделі. Є три основні команди – записати в макет данні, зчитати і очистити пам’ять(Таблиця 3.5.3).

*Таблиця 3.5.3 Команди для роботи з оперативною пам’яттю*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ пп** | **Синтаксис команди** | **Опис** | **Приклад** | **Відповідь** | **Опис відповіді** |
| **Команди для роботи з оперативною пам’ятю** | | | | | |
| **1.** | MWR ADD BYTE; | WRite to Memory by ADDress next Bytes.  Командна записує інформацію в пам'ять макету. Записати можна або один байт, або блок інформації. Якщо записується блок, то символи розділяються space(пробілом). | MWR 147 1; - запис в пам'ять по адресі 147 значення 1.  MWR 147 1 2; - запис в пам’ять з адреси 147 значення 1, 2. | MWR ADD BYTES OK; | Операція виконана вдало. |
| ERR: ADD-address WR-write\_info RD-read\_info | Помилка: в адресі «address» записувалось «write\_info», а при перевірці – отримали «read\_info» |

*Продовження таблиці 3.5.3*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ пп** | **Синтаксис команди** | **Опис** | **Приклад** | **Відповідь** | **Опис відповіді** |
| **Команди для роботи з оперативною пам’ятю** | | | | | |
| **2.** | MCLR; | Memory CLeaR.  Команда генерує очистку пам’яті макету. | **-** | CLR-ОК; | Очистка пройшла успішно. |
| ERR: MCLR AD-address DATA; | Помилка: при очистці пам’яті, в адресі «address» знайдена така інформація «DATA». |
| **3.** | MRD ADD N; | ReaD from Memory from ADDress and Number of information.  Читання інформацію з пам’яті з вказаної адреси(ADD) вказану кількість елементів(N). | MRD 147 10;  Прочитати з пам’яті , поз адреси 147 10 элементів. | MRD ADD N; | Після завершення операції отримуємо масив даних розміром N. |

## Команди контролю стану вузлів процесора

Ці команди призначені для контролю стану вузлів процесору. За допомогою них ми можемо прочитати стан шини адреси, даних, регістрів А і Б, командне слово, керуючі сигнали і інше (Таблиця 3.5.4).

*Таблиця 3.5.4 Команди контролю стану вузлів*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ пп** | **Синтаксис команди** | **Опис** | **Відповідь** | **Опис відповіді** |
| **Команди контролю стану вузлів процесора** | | | | |
| **4.** | BDRD; | Bus Data ReaD - читати з шини даних | DB; | Значення шини даних. |
| **5.** | BARD; | Bus Address ReaD - читати з шини адреси | AB; | Значення шани адреси. |
| **6.** | RARD; | ReaD from Register A - читати з регістру А | RA; | Значення регістру А. |
| **7.** | RBRD; | ReaD from Register B - читати з регістру Б. | RB; | Значення регістру В. |
| *Продовження таблиці 3.5.4* | | | | |
| **№ пп** | **Синтаксис команди** | **Опис** | **Відповідь** | **Опис відповіді** |
| **Команди контролю стану вузлів процесора** | | | | |
| **8.** | CWRD; | ReaD Command Word - читати командне слово. | CW; | Значення командного слова. |
| **9.** | CSRD; | ReaD Command Signal - читати керуючий сигнал. | CS; | Значення керуючого сигналу. |
| **10.** | STAT; | STATus - читати стан макету. Тобто всі вузли процесора(шини даних і адреси, регістрів А та В, командного слова, керуючого сигналу). | DB AB RA RB CW CS; | Масив зі значеннями вузлів процесору. |
| **11.** | TYPE; | TYPE – отримати тип пристрою | TYPE; | Тип пристрою. |
| **12.** | VER; | VERsion – отримати версію пристрою. | VERSION; | Версія пристрою. |

## Команди управління моделлю процесора

За допомогою цих команд можна формувати та очистити тактовий імпульс, генерувати один тактовий імпульс, або перезапустити процесор (Таблиця 3.5.5).

*Таблиця 3.5.5. Команди управління моделлю процесора*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ пп** | **Синтаксис команди** | | | **Опис** |
| **Команди управління моделлю процесора** | | | | |
| **12.** | | SC; | Set high level on Clock Impulse – поставити високий рівень тактового імпульсу. | |
| **13.** | | CC; | CLeaR low level on Clock Impulse – поставити низький рівень тактового імпульсу. | |
| **13.** | | PC; | Pulse Clock line (n pulses) – генерація декількох тактових імпульсів. | |
| **14.** | | CLKD; | Disable internal CLoK – відключити внутрішній тактовий імпульс. | |

|  |  |  |
| --- | --- | --- |
| *Продовження таблиці Таблиці 3.5.5.* | | |
| **№ пп** | **Синтаксис команди** | **Опис** |
| **Команди управління моделлю процесора** | | |
| **15.** | CLKE; | Enable internal CLoK – включити внутрішній тактовий імпульс. |
| **16.** | CKS; | ClocK start – генерація тактової частоти. |
| **17.** | CKE; | ClocK enable – дозвіл роботи тактової генрації макету. |
| **18.** | RES; | Перезапуск процесора. |

## Помилки

В даній таблиці приведені помилки, які виникають при неправильному синтаксисі, параметрах і таке інше (Таблиця 3.5.6).

*Таблиця 3.5.6 Помилки*

|  |  |  |
| --- | --- | --- |
| **№ пп** | **Синтаксис помилки** | **Опис** |
| **1.** | EIC | Invalid command – неправильна команда. |
| **2.** | EIP | Invalid parameter – неправильний параметр. |
| **3.** | ECT | Non-terminated. |
| **4.** | EIF | Invalid format – неправильний формат. |
| **5.** | EPL | Parameter limit exceeded – кількість параметрів перевищує ліміт. |

# РОЗДІЛ 4. Опис практичної реалізації

## **4.1 Вибір типу програми для реалізації. Локальний web-серві**с

Так як згадувалось раніше, вже була створена програма, точніше інтерфейс користувача. Він був написаний за допомогою мов програмування HTML, CSS і JavaScript. Це гарний, відновлений інтерфейс, саме головне, він відповідає основній критерії – кросс-платформенності. Було проведено тестування на різних операційних системах і комп’ютерах – в результаті якого можна зробити висновок, що даний інтерфейс відображає данні коректно, а функції запису і зчитування інформації з файлу виконуються так як і очікується. Тому доцільно продовжувати роботу в даному напрямку, тобто розробляти веб-програму.

Але в попередній дипломній роботі я зіткнулася з проблемою, що за допомогою мови програмування JavaScript дуже важко програмувати для апаратної частини, тому що розробки в цій області тільки починають розвиватися і немає багато матеріалу для навчання, також треба мати спеціальні модулі, які зараз коштують недешево та нешироко розповсюдженні. Тому було вирішено розробити програму апаратного управління.

Дану програму треба розробити так, щоб вона могла спілкуватися і з вже існуючим інтерфейсом користувача і мала доступ до апаратних ресурсів комп’ютера. Із всіх існуючих на сьогоднішній день технологій найкращим варіантом є web-програма для локального застосування.

У результаті пошуку інформації в інтернеті на тему розробки web-програми для локального застосування можна сказати, що із-за того що переважна більшість відвідуваних web-сайтів доступно через інтернет, однак багато компаній прийшли висновку, що і розробка додатків для внутрішніх мереж займає важливе місце. Цю ідею можна розвинути далі і розробити повнофункціональні web-додатки, які ніколи не будуть відправляти пакети даних через мережевий інтерфейс. Також важливим фактом є те, що досвідчені web-розробники іноді витрачають великі зусилля на вивчення якого-небудь GUI-інструментарію, хоча те, що дійсно потрібно в їх ситуації - це простий CGI-скрипт.

У всякому разі, web-додаток, призначений для роботи тільки в локальній мережі, набагато простіше додатків для універсального застосування. Це дає можливість легко задати вимоги для браузера, а продуктивність сервера, швидше за все, не буде великою проблемою. Прості додатки, що використовують стандартні віджети CGI-форм і їм подібні, можна написати за набагато менший час, ніж треба було б на розробку самодостатніх додатків. Додатки, побудовані навколо обробки форм або даних, часто є відмінними кандидатами на реалізацію в якості звичайних web-сервісів.

В моєму випадку такий спеціальний додаток може забезпечити більш витончене і просте рішення поставленої задачі.

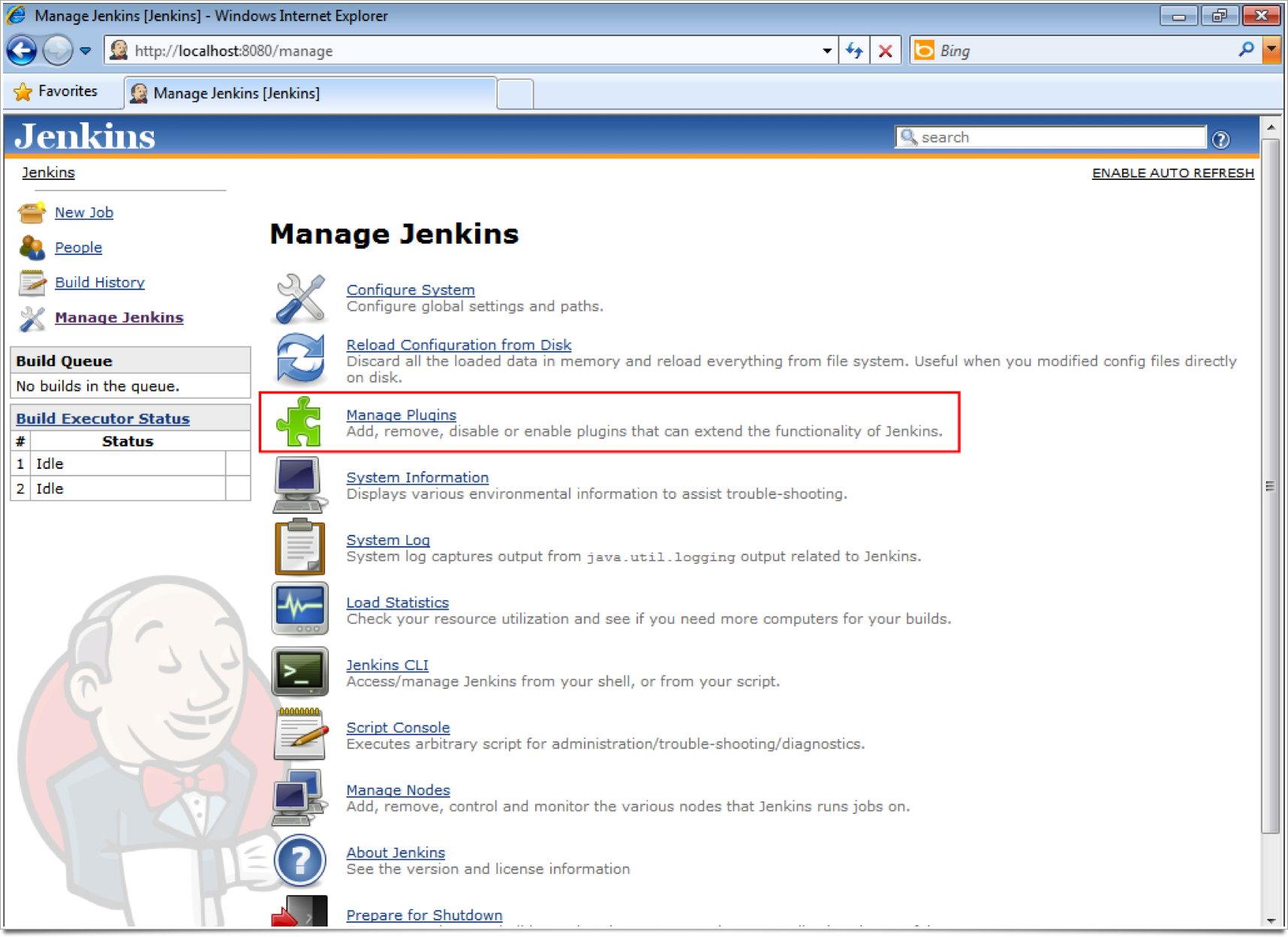
## Переваги використання web-браузер

Якщо проаналізувати, що вміє web-браузер, чого не вміють інші програми? Відповідь очевидна: нічого. Але тоді що ж можна написати на високорівневих мовах, чого не можна виконати в машинних кодах? Знову ж таки, нічого. Перевага при використанні web-браузера в якості інтерфейсу полягає в тому, що все основне кодування вже виконано. Не потрібно відстежувати події зміни розмірів або розгортання вікна або події меню. Все, що потрібно зробити, це прочитати фрагмент даних з запиту і обробити його.

Web-браузер робить ще одну дуже корисну річ: він надає безліч налаштувань і переваг, про які вам не потрібно турбуватися. Користувач може змінювати розмір шрифту під час роботи. Аналогічно, якщо згенеровані вихідні дані представляються в простій і акуратній HTML-формі, їх можна легко і швидко вивести на друк. Багато функцій, які в іншому випадку довелося б реалізовувати самостійно (наприклад, збереження вихідних даних в файл, вивід вихідних даних на друк, зміна розмірів вікон), вже реалізовані [7].

## Приклади існуючих локальних Web-серверів

Одним з найпопулярніших локальних Web-серверів є додаток для Continuous Integration – Jenkins. Після загрузки з офіційного сайту даної програми ви отримуєте файл з розширенням war – формат файлу, що описує, як повний веб-додаток упаковується відповідно до специфікації Java-сервлетів в файл в форматі JAR або ZIP. Клікнувши на цей файл, відбувається установка і після завершення все що потрібно зробити – це набрати в браузері localhost:8080. Окриється стартова сторінка даного додатку і можна працювати (Рисунок 4.1.1).



*Рисунок 4.1.1 – Стартова сторінка Jenkins*

Свою програму хочу зробити по такому принципу, адже є багато плюсів: невеликий файл, установка файлу триває швидко, кроссбраузерність, знайомий інтерфейс браузера, до якого вже всі звикли.

## 4.2 Взаємодія інтерфейсу с програмою-сервером

Перед тим як користувач побачить вміст сайту у себе на екрані, браузер робить запит на сервер щоб отримати вміст.

Перше що потрібно запам'ятати - браузер запитує і отримує дані за допомогою HTTP протоколу, тому його (браузер) ще називають HTTP клієнтом.

Протокол передачі Гіпертексту (HTTP - англ. HyperText Transfer Protocol) - спеціально розроблений протокол як основа World Wide Web і використовується для передачі всіх необхідних даних: HTML коду, зображень, CSS файлів, javascript і т.д. [8].

Відповідно до специфікації OSI, HTTP є протоколом прикладного (верхнього, 7-го) рівня. Актуальна на даний момент версія протоколу, HTTP 1.1, описана в специфікації RFC 2616.

Протокол HTTP припускає використання клієнт-серверної структури передачі даних. Клієнтську програму формує запит і відправляє його на сервер, після чого серверне програмне забезпечення обробляє цей запит, формує відповідь і передає його назад клієнтові. Після цього клієнтську програму може продовжити відправляти інші запити, які будуть оброблені аналогічним чином.

Завдання, яке традиційно вирішується за допомогою протоколу HTTP - обмін даними між призначеним для користувача додатком, що здійснює доступ до веб-ресурсів (зазвичай це веб-браузер) і веб-сервером. На даний момент саме завдяки протоколу HTTP забезпечується робота Всесвітньої павутини.

Також HTTP часто використовується як протокол передачі інформації для інших протоколів прикладного рівня, таких як SOAP, XML-RPC і WebDAV. У такому випадку говорять, що протокол HTTP використовується як «транспорт».

API багатьох програмних продуктів також має на увазі використання HTTP для передачі даних - самі дані при цьому можуть мати будь-який формат, наприклад, XML або JSON.

Як правило, передача даних по протоколу HTTP здійснюється через TCP / IP-з'єднання. Серверне програмне забезпечення при цьому зазвичай використовує TCP-порт 80 (і, якщо порт не вказано явно, то зазвичай клієнтське програмне забезпечення за замовчуванням використовує саме 80-й порт для відкритих HTTP-з'єднань), хоча може використовувати і будь-який інший.

Для того, щоб сформувати HTTP-запит, необхідно скласти стартовий рядок, а також задати принаймні один заголовок - це заголовок Host, який є обов'язковим, і повинен бути присутнім в кожному запиті. Справа в тому, що перетворення доменного імені в IP-адресу здійснюється на стороні клієнта, і, відповідно, коли ви відкриваєте TCP-з'єднання, то віддалений сервер не володіє жодною інформацією про те, яка саме адреса використовувалася для з'єднання: це могла бути, наприклад, адреса exzample.google.com, google.com або m.google.com - і у всіх цих випадках відповідь може відрізнятися. Однак фактично мережеве з'єднання у всіх випадках відкривається з вузлом 212.24.43.44, і навіть якщо спочатку при відкритті з'єднання було поставлено не ця IP-адреса, а будь-яке доменне ім'я, то сервер про це ніяк не дізнається - і саме тому цю адресу необхідно передати в заголовку Host.

Стартовий (початковий) рядок запиту для HTTP 1.1 складається за такою схемою:

Метод URI HTTP / Версія

Наприклад (такий стартовий рядок може вказувати на те, що запитується головна сторінка сайту):

GET / HTTP / 1.1

Метод (в англомовній тематичній літературі використовується слово method, а також іноді слово verb - «дієслово») являє собою послідовність з будь-яких символів, крім керуючих і роздільних, і визначає операцію, яку потрібно здійснити з зазначеним ресурсом. Специфікація HTTP 1.1 не обмежує кількість різних методів, які можуть бути використані, проте в цілях відповідності загальним стандартам і збереження сумісності з максимально широким спектром програмного забезпечення як правило використовуються лише деякі, найбільш стандартні методи, зміст яких однозначно розкритий в специфікації протоколу.

URI (Uniform Resource Identifier, уніфікований ідентифікатор ресурсу) - шлях до конкретного ресурсу (наприклад, документа), над яким необхідно здійснити операцію (наприклад, в разі використання методу GET мається на увазі отримання ресурсу). Деякі запити можуть не ставитися до будь-якого ресурсу, в цьому випадку замість URI в стартовий рядок може бути додана зірочка (астеріск, символ «\*»). Наприклад, це може бути запит, який відноситься до самого веб-сервера, а не якого-небудь конкретного ресурсу. В цьому випадку стартовий рядок може виглядати так:

OPTIONS \* HTTP / 1.1

Версія визначає, відповідно до якої версією стандарту HTTP складений запит. Вказується як два числа, розділених крапкою (наприклад 1.1).

Для того, щоб звернутися до веб-сторінці за певною адресою (в даному випадку шлях до ресурсу - це «/»), слід відправити наступний запит:

GET / HTTP / 1.1

Host: exzample.google.com

При цьому слід враховувати, що для розриву рядків слід використовувати символ повернення каретки (Carriage Return), за яким слідує символ перекладу рядка (Line Feed). Після оголошення останнього заголовка послідовність символів для розриву рядків додається двічі.

Втім, в специфікації HTTP рекомендується програмувати HTTP-сервер таким чином, щоб при обробці запитів в якості міжрядкового роздільник сприймався символ LF, а попередній символ CR, за наявності такого, ігнорувався. Відповідно, на практиці більшість серверів коректно обробить і такий запит, де заголовки відокремлені символом LF, і він же двічі доданий після оголошення останнього заголовка.

А як отримати відповідь? Стартовий рядок відповіді має наступну структуру:

HTTP / Версія Код стану Пояснення

Версія протоколу тут задається так само, як в запиті.

Код стану (Status Code) - три цифри (перша з яких вказує на клас стану), які визначають результат здійснення запиту. Наприклад, в разі, якщо був використаний метод GET, і сервер надає ресурс із зазначеним ідентифікатором, то такий стан задається за допомогою коду 200. Якщо сервер повідомляє про те, що такого ресурсу не існує - 404. Якщо сервер повідомляє про те, що не може надати доступ до цього ресурсу через відсутність необхідних привілеїв у клієнта, то використовується код 403. Специфікація HTTP 1.1 визначає 40 різних кодів HTTP, а також допускається розширення протоколу і використання додаткових кодів станів.

Пояснення до коду стану (Reason Phrase) - текстове (але не включає символи CR і LF) пояснення до коду відповіді, призначене для спрощення читання відповіді людиною. Пояснення може не враховуватися клієнтським програмним забезпеченням, а також може відрізнятися від стандартного в деяких реалізаціях серверного програмного забезпечення.

Після стартового рядка слідують заголовки, а також тіло відповіді. наприклад:

HTTP / 1.1 200 OK

Server: nginx / 1.2.1

Date: Sat, 08 Mar 2014 22:53:46 GMT

Content-Type: application / octet-stream

Content-Length: 7

Last-Modified: Sat, 08 Mar 2014 22:53:30 GMT

Connection: keep-alive

Accept-Ranges: bytes

Wisdom

Тіло відповіді слідує через два розрива рядків після останнього заголовка. Для визначення закінчення тіла відповіді використовується значення заголовка Content-Length (в даному випадку відповідь містить 7 вісімкових байтів: слово «Wisdom» і символ розриву рядків) [9].

## AJAX

AJAX - це абревіатура, яка означає асинхронний Javascript і XML. Насправді, AJAX не нова технологія, так як і Javascript, і XML існують досить довготривалий час, а AJAX - це синтез позначених технологій. AJAX частіше всього асоціюється з терміном Web 2.0 і підноситься як найновіший Web-додаток.

При використанні AJAX немає необхідності оновлювати кожну всю сторінку, так як оновлюється тільки її конкретна частина. Це набагато зручніше, так як не приходиться довго чекати, і економічно, так як не всі володіють безлімітним інтернетом.

Переваги AJAX:

* можливість створення зручного Web-інтерфейсу;
* активна взаємодія з користувачем;
* часткова перезавантаження сторінки, замість повної;
* зручність використання.

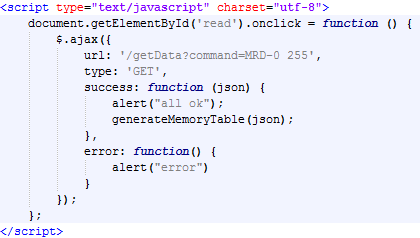
Повертаючись до нашого інтерфейсу користувача, де на першій вкладці є таблиця – візуальне представлення пам’яті моделі цифрового процесору, в більшості випадків робота буде проводитись з нею. А саме:

1. Користувач має можливість редагувати вміст таблиці, клікнув по будь-якому елементу в ній;
2. Також користувач може зберегти поточний стан таблиці в файл натиснувши на кнопку «SAVE», а через деякий час відкрити її (кнопка «OPENЦ), і вміст файлу відобразиться в таблиці;
3. При натисканні на кнопку «WRITE», інформація з таблиці буде відправлятись на модель;
4. При натисканні на кнопку «READ», стан пам’яті моделі буде зчитуватись та відображатися в даній таблиці;
5. Також є можливість очистити пам’ять, це означає, що в кожному елементі таблиці буде дефолтне значення «00», паралельно з цим виконується очистка пам’яті на моделі процесору.

Якщо взаємодія з макетом може відбуватися не дуже часто і інформація, зазвичай, з моделі приходить велика, тобто повна таблиця, то можна зробити перезавантаження сторінки, але при цьому можуть зникнути дані з другої вкладки, це перше.

А друге, випадок, коли іде процес заповнення таблиці вручну, то кожен раз перезавантажувати сторінку немає сенсу. Тому доцільно використовувати технологію AJAX в даному проекті.

Далі приведено фрагмент коду в якому буде відправлятися на сервер команда «getData» з параметром command=MRD-0 255. (Рисунок 4.2.1)



*Рисунок 4.2.1 – Функція ajax для отримання даних з макету*

По перше, треба поставити обробник подій – при натисканні на кнопку «READ» (в HTML розмітці цей елемент має id=”read”), буде визиватись обробник подій .onclick. В цьому методі і напишемо наш ajax. Для цього треба вказати URL сервісу, який обробить даний запит (url: ‘getData?command=MRD 0 255’), тип запиту (type: “Get”), дії при успішній/невдалій відповіді сервісу. При успішній відповіді повідомляємо користувача, що запит виконано і за допомогою методу generateMemoryTable() передаємо json, який прийшов, з нього візьмуться дані для таблиці і відображаться в ній. При невдачі на сервері, повино прийти повідомлення в якому описана причина проблеми, і це, також, повідомляється користувачеві за допомогою спливаючого вікна.

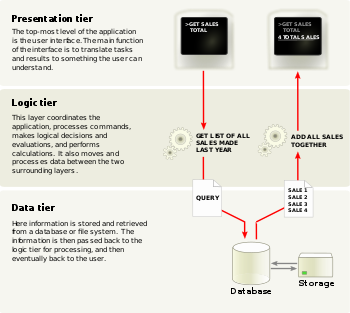
Аналогічно, з відповідними параметрами пишуться методи для інших кнопок інтерфейсу.

**4.3 Програма-сервер**

Дана програма буде призначатися для прийому запитів від браузеру, конвертація даних, якщо потрібна, відправка до моделі цифрового процесору, отримання відповіді від неї та відправка даної відповіді браузеру. Це так звана триярусна архітектура, але в моєму випадку буде трохи змінена.

**Триярусна архітектура**

У комп'ютерних технологіях **трирівнева архітектура**, синонім **триланкова архітектура** (англ. three-tier або Multitier architecture) передбачає наявність наступних компонент програми: клієнтський застосунок (зазвичай говорять «тонкий клієнт» або термінал), підключений до сервера застосунків, який в свою чергу підключений до серверу бази даних (Рисунок 4.3.1).

[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Overview_of_a_three-tier_application_vectorVersion.svg)

*Рисунок 4.3.1 – Visual overview of a Three-tiered application*

*Клієнт* — це [інтерфейсний](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0) (зазвичай [графічний](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)) компонент, який представляє перший рівень, власне [застосунок](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) для кінцевого користувача. Перший рівень не повинен мати прямих зв'язків з базою даних (за вимогами безпеки), не повинен бути навантаженим основною бізнес-логікою (за вимогами масштабованості) і зберігати стан програми (за вимогами надійності). На перший рівень може бути винесена і зазвичай виноситься найпростіша бізнес-логіка: інтерфейс авторизації, алгоритми шифрування, перевірка значень, що вводяться, на допустимість і відповідність формату, нескладні операції (сортування, групування, підрахунок значень) з даними, вже завантаженими на термінал.

*Сервер застосунків* розташовується на другому рівні. На другому рівні зосереджена більша частина бізнес-логіки. Поза ним залишаються фрагменти, що експортуються на термінали (див. вище), а також розміщені в третьому рівні збережені процедури і тригери.

*Сервер бази даних* забезпечує зберігання даних і виноситься на третій рівень. Зазвичай це стандартна реляційна або об'єктно-орієнтована СУБД. Якщо третій рівень являє собою базу даних разом з збереженими процедурами, тригерами і схемою, яка описує застосунок в термінах реляційної моделі, то другий рівень будується як програмний інтерфейс, що зв'язує клієнтські компоненти з прикладною логікою бази даних.

Але в моєму випадку сервера бази даних не буде. Замість нього буде модель цифрового процесору, на яку ми будемо відправляти команди та отримувати відповіді.

У порівнянні з [клієнт-серверною](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0) або [файл-серверною](https://uk.wikipedia.org/w/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80&action=edit&redlink=1) архітектурою можна виділити такі переваги трирівневої архітектури:

* [масштабованість](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D1%96%D1%81%D1%82%D1%8C)
* конфігурованість — ізольованість рівнів один від одного дозволяє (при правильному розгортанні архітектури) швидко і простими засобами переконфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів
* високий рівень безпеки
* висока [надійність](https://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B4%D1%96%D0%B9%D0%BD%D1%96%D1%81%D1%82%D1%8C)
* низькі вимоги до швидкості каналу (мережі) між терміналами і сервером застосунків
* низькі вимоги до продуктивності і технічних характеристик терміналів, як наслідок зниження їхньої вартості. Терміналом може виступати не тільки комп'ютер, але і, наприклад, [мобільний телефон](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D1%96%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D1%82%D0%B5%D0%BB%D0%B5%D1%84%D0%BE%D0%BD) [10]. <https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D1%8F%D1%80%D1%83%D1%81%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0>

**Servlets**

Сервлет є інтерфейсом Java, реалізація якого розширює функціональні можливості сервера. Сервлет взаємодіє з клієнтами за допомогою принципу запит-відповідь.

Хоча сервлети можуть обслуговувати будь-які запити, вони зазвичай використовуються для розширення веб-серверів. Для таких додатків технологія Java Servlet визначає HTTP-специфічні сервлет класи.

Пакети javax.servlet і javax.servlet.http забезпечують інтерфейси і класи для створення сервлетів.

Перша специфікація сервлетів була створена в Sun Microsystems (версія 1.0 була закінчена в червні 1997). Починаючи з версії 2.3, специфікація сервлетів розроблялася під керівництвом Java Community Process. Стандарт JSR 53 визначав як Servlet 2.3, так і специфікацію JavaServer Page 1.2. JSR 154 включає в себе специфікації Servlet 2.4 і 2.5. Поточна специфікація на 13 червня 2013 року - Servlet 3.1 (описана в JSR-340).

**Стандарті та специфікації**

Сервлети, [інтерфейси](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) та базові класи, протоколи роботи з ними, робоче оточення, описуються у відповідних специфікаціях компанії [Sun Microsystems](https://uk.wikipedia.org/wiki/Sun_Microsystems).

Для полегшення розробки HTTP сервлетів, у специфікації описано абстрактний клас HttpServlet, від якого розробникам пропонується [успадковувати](https://uk.wikipedia.org/wiki/%D0%A3%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) свої сервлети.

**Схема роботи та застосування**

1. Клієнт (наприклад, [Веб-оглядач](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BE%D0%B3%D0%BB%D1%8F%D0%B4%D0%B0%D1%87)), відвідує веб-сторінку та надсилає HTTP запит на сервер.
2. Web-сервер отримує запит та передає його контейнеру сервлетів. Контейнер сервлетів може виконуватись в тому ж самому процесі, що і веб-сервер, в окремому процесі на тій же системі, що і веб-сервер, або взагалі в окремому процесі на іншій системі.
3. Контейнер сервлетів з'ясовує який сервлет слід викликати, виходячи з інформації про конфігурацію утримуваних сервлетів, та викликає його, передаючи як параметри об'єктні представлення запиту та відповіді.
4. Сервлет використовує об'єкт запиту для отримання інформації про віддаленого користувача, параметри HTTP запиту тощо. Сервлет виконує запрограмовані в ньому дії та надсилає результати роботи через об'єкт відповіді.
5. Після того, як сервлет припиняє обробку запиту, контейнер сервлетів перевіряє коректність відправки відповіді, та повертає управління до головного веб-сервера.

Сервлети, також, використовуються в технології [JSP](https://uk.wikipedia.org/wiki/JSP). Шаблони сторінок [транслюються](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) у вихідні тексти Java-класів успадкованих від стандартних класів сервлетів. Java-компілятор [компілює](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80) ці вихідні тексти в Java-байт коди. Отримані скомпільовані класи можуть використовуватись в сервлет-контейнері. Як правило, сервлет-контейнери виконують усі ці допоміжні дії автоматично [11]. <https://uk.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%BB%D0%B5%D1%82>

[12] <https://www.oracle.com/technetwork/java/index.html>

На сервер приходить запит від клієнта, запит містить в собі URL і параметри. Сервер має спеціальний конфігураційний файл, який йому повідомляє про те, який сервлет треба виконати в разі приходу певного URL. Сервлет виконується (там ви можете використовувати параметри) і створює HTML-сторінку, яка відсилається клієнту.

Сервер по суті є контейнером (тепер вже не візуальних компонентів), який завантажує сервлети, виконує їх, викликаючи певні методи і отримавши від них результат, відправляє його клієнту.

Таким чином сервлет - це Java-клас, який успадковується зазвичай від класу HttpServlet і перевизначає частина методів:

• doGet - якщо ми хочемо, щоб сервлет реагував на GET запит.

• doPost - якщо ми хочемо, щоб сервлет реагував на POST запит.

• doPut, doDelete - якщо ми хочемо, щоб сервлет реагував на PUT і DELETE запит (є і такі в HTTP). Ці методи реалізуються вкрай рідко, тому що самі команди теж дуже рідко зустрічаються.

• init, destroy - для управління ресурсами в момент створення сервлету і в момент його знищення [13]. http://java-course.ru/student/book1/servlet/

**Apache Tomcat**

Як з'ясовується, просто написати сервлет мало. Його треба скомпілювати і після цього встановити на сервер. І треба встановити так, щоб сервер знав, як ним користуватися.

Сервер, який допоможе − це Apache Tomcat. Tomcat (в старих версіях - Catalina) - контейнер сервлетів з відкритим вихідним кодом, що розробляється Apache Software Foundation. Реалізує специфікацію сервлетів, специфікацію JavaServer Pages (JSP) і JavaServer Faces (JSF). Написаний на мові Java.

Tomcat дозволяє запускати веб-додатки, містить ряд програм для самоконфігурірованія.

Tomcat використовується в якості самостійного веб-сервера, як сервер контенту в поєднанні з веб-сервером Apache HTTP Server, а також в якості контейнера сервлетів в серверах додатків JBoss і GlassFish [14] https://ru.wikipedia.org/wiki/Apache\_Tomcat.

**Системи збирання проектів**

Автоматизація збирання - етап процесу розробки програмного забезпечення, що полягає в автоматизації широкого спектра завдань, що вирішуються програмістами в їх повсякденній діяльності.

Включає в себе такі дії, як:

* компіляція вихідного коду в Об'єктний модуль;
* зборка бінарного коду в виконуваний файл;
* виконання тестів;
* розгортання програми на виробничій платформі;
* написання супровідної документації або опис змін нової версії.

Автоматизація збирання має такі переваги:

* Покращення якості продукту;
* Прискорення процесу компіляції і компонування;
* Позбавлення від зайвих дій;
* Мінімізація «поганих (некоректних) збірок»;
* Позбавлення від прив'язки до конкретної людини;
* Ведення історії збірок і релізів для розбору випусків;
* Економія часу і грошей завдяки причин, зазначеним вище. [15] <https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%81%D0%B1%D0%BE%D1%80%D0%BA%D0%B8>

**Apache Maven**

Мавен - це інструмент для збірки Java проекту: компіляції, створення jar, створення дистрибутива програми, генерації документації. Прості проекти можна зібрати в командному рядку. Якщо збирати великі проекти з командного рядка, то команда для збірки буде дуже довгою, тому її іноді записують в bat / sh скрипт. Але такі скрипти залежать від платформи. Для того щоб позбутися від цієї залежності і спростити написання скрипта використовують інструменти для збірки проекту.

Для платформи Java існують два основні інструменти для збірки: Ant і Maven.

Основні переваги Maven

* незалежність від OS. Збірка проекту відбувається в будь-якій операційній системі. Файл проекту один і той же;
* управління залежностями. Рідко які проекти пишуться без використання сторонніх бібліотек (залежностей). Ці сторонні бібліотеки часто теж в свою чергу використовують бібліотеки різних версій. Мавен дозволяє управляти такими складними залежностями. Що дозволяє вирішувати конфлікти версій і в разі потреби легко переходити на нові версії бібліотек;
* можлива збірка з командного рядка. Таке часто необхідно для автоматичного складання проекту на сервері (Continuous Integration);
* гарна інтеграція з середовищами розробки. Основні середовища розробки на java легко відкривають проекти які збираються c допомогою maven. При цьому найчастіше проект налаштовувати не потрібно - він відразу готовий до подальшої розробки;
* як наслідок - якщо з проектом працюють в різних середовищах розробки, то maven зручний спосіб зберігання налаштувань. Її конфігураційний файл середовища розробки і для збірки один і той же - менше дублювання даних і відповідно помилок;
* декларативне опис проекту [16] http://www.apache-maven.ru/.

**Створення проекту**

Для розробки даного проекту буду використовувати IDE **Intellij IDEA. Для початку створюю проект Maven і щоб була можливість використовувати Servlet треба додати в pom.xml – файл конфігурації нашого проекту наступні залежності:**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>ua.hnu</groupId>

<artifactId>microproc</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

</dependencies>

</project>

Ці залежності автоматично завантажують потрібні нам бібліотеки для роботи з сервлетами, також ми будемо її доповнювати іншими бібліотеками, які нам знадобляться при написанні нашої програми. В цьому файлі ми також вказуємо, що вихідний файл програми матиме розширення war, це означає, щоб запустити нашу програму, не потрібно буде її додатково завантажувати, а просто клікаючи на файлі цього розширення він буде автоматично запускатися в браузері.

Також нам потрібно додати плагін для роботи з сервером Tomcat. Для цього вставимо в наш pom.xml слідуюче:

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

Працює по аналогії з залежностями, але завантажується не бібліотека, а програма.

Далі напишемо наш сервлет:

package ua.hnu.servlet;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

@WebServlet(“/example”)

public class MainServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

PrintWriter pw = resp.getWriter();

pw.println(“<H1>Hello, world!”);

}

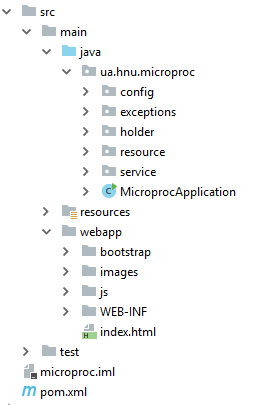
}

Ми написали клас, який реалізує абстрактний клас **HttpServlet і переопреділили його метод doGet. Після запустку нашого web-додатку, коли браузер надішле запит методом GET «localhost:8080/example» (вказано в анотації @WebServlet перед класом), то запит прийде на цей сервлет і у відповідь ми відправляємо фразу «**Hello, world!».

## Структура проекту

Можна було б на кожну команду моделі процесору написати свій сервлет, але це було би багато класів, у яких повторювався код. Також при майбутньому вдосконаленні моделі і при доповненням нових команд довелося б знову створювати нові класи, а сам проект був би достатньо об’ємним. По принципам об’єктно-орієнтованому програмування SOLID [17] <https://web-creator.ru/articles/solid>, першим пунктом сказано − принцип єдиного обов'язку / відповідальності (single responsibility principle) означає, що кожен об'єкт повинен мати один обов'язок і цей обов'язок повинен бути повністю інкарпсульован в класі. Всі його сервіси повинні бути спрямовані виключно на забезпечення цього обов'язку. Поки що, в даній реалізації у нас є тільки один клас, який приймає запити від клієнта, писати сюди конвертацію та взаємодію з моделлю неправильно, виходячи з того, що написано вище. Тому треба зробити декілька рівнів нашої програми, щоб забезпечити інкарпсульованість кожного обов’язку у відповідному класі.

Приблизна схема проекту представлена на Рисунку 4.3.2.



*Рисунок 4.3.2 – Структура проекту*

В папці config будуть конфігураційні класи, в яких при старті програми будуть ініціалізуватися всі класи і їх зв’язки між собою.

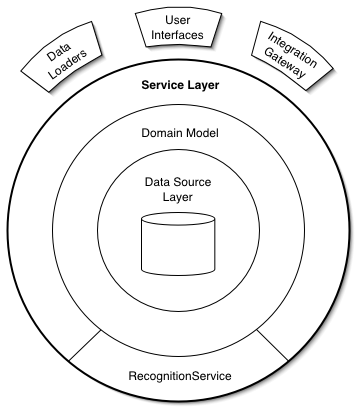
Data Transfer Object (DTO) - один з шаблонів проектування, використовується для передачі даних між підсистемами додатки. В папці dto будуть знаходитися такі об’єкти.

При виконанні різних операцій, як за моделлю так і при взаємодії з клієнтом, або просто при виконанні програми можуть виникнути помилки, не всі існуючі помилки можуть підійти для описання проблеми, тому тут будемо створювати свої власні exceptions.

Також можуть бути змінні, які будуть використовуватися на різних рівнях нашої програми, тому створимо клас в папці holders PropertyHolder, який буде зберігати ці дані.

В папці resource будуть класи, за допомогою яких будемо приймати запити від клієнту та відправляти відповіді.

І папка service буде слугувати для зберігання класів, які конвертують дані та взаємодіють з моделлю процесору. Це так званий Service Layer Pattern (Рисунок 4.3.3). Він визначає межу між додатком і шаром сервісів, який утворює набір доступних операцій і управляє відповіддю додатки в кожній операції.



*Рисунок 4.3.3 –* *Service Layer Pattern*

Бізнес-додатки зазвичай потребують різних інтерфейсах до даних, які вони зберігають і логіці, яку реалізують: завантажувачі даних, призначені для користувача інтерфейси, інтеграційні шлюзи і інше. Всупереч різним цілям, ці інтерфейси часто потребують у взаємодії з додатком для доступу і управління його даними і виконання логіки. Ці взаємодії можуть бути складними, що використовують транзакції на декількох ресурсах і управління декількома відповідями на дію. Програмування логіки взаємодії для кожного інтерфейсу викличе більше кількість дублювання.

Патерн Service Layer визначає для додатка кордон і набір допустимих операцій з точки зору взаємодіючих з ним клієнтських. Він інкапсулює бізнес-логіку додатка, керуючи транзакціями і керуючи відповідями в реалізації цих операцій [18].

Програма інтерфейсу користувача буде знаходитись в папці webapp.

## Spring Framework

Spring Framework, або просто Spring - один з найпопулярніших фреймворків для створення веб-додатків на Java. Фреймворк - це щось схоже на бібліотеку, але є один момент. Грубо кажучи, використовуючи бібліотеку, ви просто створюєте об'єкти класів, які в ній є, викликаєте потрібні вам методи, і таким чином отримуєте потрібний вам результат. Тобто, тут більш імперативний підхід: ви чітко вказуєте в своїй програмі в який конкретний момент треба створити якийсь об'єкт, в який момент викликати конкретний метод, і так далі. З фреймворками справи йдуть трохи інакше. Ви просто пишете якісь свої класи, прописуєте там якусь частину логіки, а створює об'єкти ваших класів і викликає методи за вас вже сам фреймворк. Найчастіше, ваші класи імплементують якісь інтерфейси з фреймворка або успадковують якісь класи з нього, таким чином отримуючи частину вже написаної за вас функціональності. Але не обов'язково саме так. В Spring наприклад намагаються по максимуму відійти від такої жорсткого зв’язку (коли ваші класи безпосередньо залежать від якихось класів / інтерфейсів з цього фреймворка), і використовують для цієї мети анотації.

Але важливо зрозуміти, що Spring - це просто набір якихось класів і інтерфейсів, які вже написані.

Ще хочу зазначити, що Spring можна використовувати не тільки для веб-додатків, але і для так знайомих усім нам самих звичайних консольних програмок.

**Структура**

Але Spring - це не один якийсь конкретний фреймворк. Це скоріше загальне назви для цілого ряду невеликих фреймворків, кожен з яких виконує якусь свою роботу (Рисунок 4.3.4).

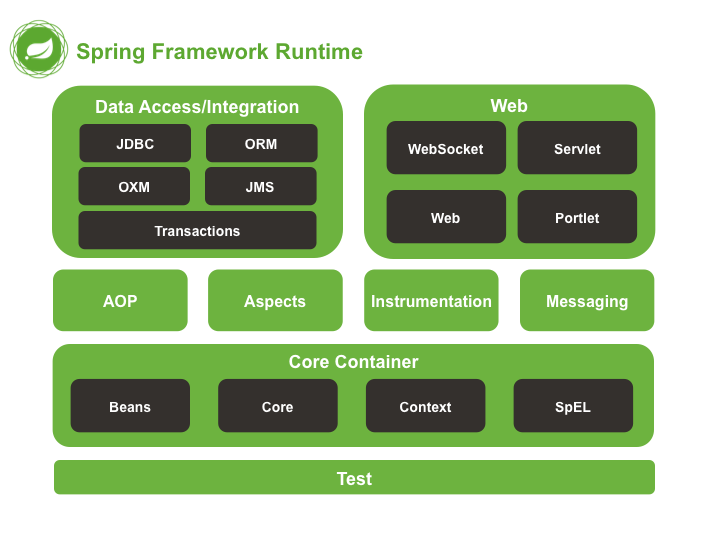


Рисунок 4.3.4 – Структура Spring framework

Як видно, Spring має модульну структуру. Це дозволяє підключати тільки ті модулі, що нам потрібні для нашогї програми, але не підключати ті, якими ми свідомо не будемо користуватися. Наскільки мені відомо, то саме цей підхід і допоміг Spring обійти свого конкурента в той час (EJB) і захопити лідерство. Тому що функції, які має EJB тягнули дуже багато залежностей за собою, та й взагалі виходили повільні і неповороткі.

На зображенні видно, що Spring фреймворк складається ніби з кількох модулів:

* data access;
* web;
* core;
* та інших.

Як можна було здогадатися, модуль data access містить в собі засоби для роботи з даними (в основному, з базами даних), web - для роботи в мережі.

Крім того, є ще так-звана ціла Spring-інфраструктура: безліч інших проектів, які не входять до сам фреймворк офіційно, але при цьому бесшовно інтегруються в ваш проект на Spring.

## Переваги Spring

Можна відразу сказати, що як тільки їм хоч трохи опанувати – одразу ж розумієш скільки всякої різної роботи тепер не доводиться робити, і скільки всього бере на себе Spring. Можна написати пару десятків рядків конфігов, написати парочку класів - і вийде працюючий проект. Але як тільки починаєш замислюватися скільки там всього знаходиться "під капотом", скільки роботи виконується, і скільки довелося б писати коду, якщо робити такий же проект на голих сервлетах або на сокетах і чистої Java.

## Dependency Inversion та Inversion of Control

Перше з чим стикаєшся, коли починаєш вивчати Spring – це Dependency Inversion (DI) та Inversion of Control (IoC).

IoC – інверсія управління. Про це вже згадувалося, коли описувалось, що при використанні бібліотеки ви самі прописуєте в своєму коді який метод якого об'єкта викликати, а у випадку з фреймворками – найчастіше фреймворк буде викликати в потрібний йому момент код, який ми написали. Тобто, тут вже не ви керуєте процесом виконання коду/програми, а фреймворк це робить за вас. Ви передали йому управління (інверсія управління).

Під DI розуміють то Dependency Inversion (інверсію залежностей, тобто спроби не робити жорстких зв'язків між вашими модулями/класами, де один клас безпосередньо зав'язаний на інший), то Dependency Injection (впровадження залежностей, це коли будь-які об'єкти створюєте не ви в main методі і потім передаєте їх в свої методи, а за вас їх робить Spring [19]. <https://javarush.ru/groups/posts/476-spring-dlja-lenivihkh-osnovih-bazovihe-koncepcii-i-primerih-s-kodom-chastjh-1>

ApplicationContext - це головний інтерфейс в Spring-додатку, який надає інформацію про конфігурацію програми. Він доступний тільки для читання під час виконання, але може бути перезавантажений при необхідності і підтримки додатком. Число класів, що реалізують ApplicationContext інтерфейс, доступні для різних параметрів конфігурації і типів додатків.

ApplicationContext надає:

* Фабричні методи біна для доступу до компонентів додатка;
* Можливість завантажувати файлові ресурси в загальному вигляді;
* Можливість публікувати події та реєструвати обробники на них
* Можливість працювати з повідомленнями з підтримкою інтернаціоналізації;
* Спадкування від батьківського контексту;
* Доступність контексту програми.

Нижче представлено кілька способів отримання посилання на контекст програми. Ви можете реалізувати ApplicationContextAware як показано на прикладі нижче:

package ua.hnu.spring;

import org.springframework.beans.BeansException;

import org.springframework.context.ApplicationContext;

import org.springframework.context.ApplicationContextAware;

public class A implements ApplicationContextAware {

    private ApplicationContext applicationContext;

public void setApplicationContext (ApplicationContext

applicationContext) {

        this.applicationContext = applicationContext;

    }

}

Також можна використовувати анотацію @Autowired для введення посилання безпосередньо в наш клас:

package ua.hnu.spring;

import org.springframework.beans.factory.annotation.Autowired;

public class B {

    @Autowired

private ApplicationContext applicationContext;

}

## Бібліотека для взаємодії з інтерфейсним модулем

Для організації обміну даних між програмою апаратного управління і комунікаційним портом була обрана бібліотека jSSC - Java-бібліотека для простої роботи з COM-портом (Java Simple Serial Connector) [20]. https://habr.com/post/133766/ Своє офіційне публічне початок вона бере в 2010 році. Саме тоді було вирішено поділитися нею з розробниками на Java (поширюється під ліцензією LGPL). До написання бібліотеки привів факт відсутності адекватних інструментів для роботи з послідовними портами. На той час уже існували javax.comm і rxtx, але в цих бібліотеках є ряд своїх недоліків. Бібліотека javax.comm 3.0 не призначена для роботи в операційних системах сімейства Windows, крім того, складно знайти її якісну ліцензійну версію. Бібліотека rxtx не підійшла для вирішення завдання через свою нестабільність.

У самій бібліотеці jSSC можна виділити кілька основних частин:

- SerialNativeInterface - клас, який надає доступ до всіх «нативним» методам бібліотеки jSSC;

- SerialPort - клас, за допомогою якого буде безпосередньо проводитися робота з обраним портом;

- SerialPortEventListener - інтерфейс, який необхідно реалізувати, для того щоб отримувати дані з порту.

Нижче наведено фрагмент коду для команди очищення програмної пам'яті моделі процесора. За допомогою даної підпрограми виконується відкриття і настройка порту, передача команди і очікування прийому відповіді з його подальшою обробкою.

 package ua.hnu.connector;

public class Test {

   private static SerialPort serialPort;

   public static void main (String [] args) {

     // Передаємо в конструктор ім'я порту

     serialPort = new SerialPort ( "COM1");

     try {

       // Відкриваємо порт

       serialPort.openPort ();

       // Виставляємо параметри

       serialPort.setParams (

         SerialPort.BAUDRATE\_9600,

         SerialPort.DATABITS\_8, SerialPort.STOPBITS\_1,

         SerialPort.PARITY\_NONE

);

       // Включаємо апаратне керування потоком

       serialPort.setFlowControlMode (

         SerialPort.FLOWCONTROL\_RTSCTS\_IN |

         SerialPort.FLOWCONTROL\_RTSCTS\_OUT

);

       // Встановлюємо слухач і маску

       serialPort.addEventListener (

         new PortReader (), SerialPort.MASK\_RXCHAR);

       // Відправляємо запит пристрою

       serialPort.writeString ("MCLR;");

     } catch (SerialPortException ex) {

       System.out.println (ex);

     }

   }

   private static class PortReader implements

       SerialPortEventListener {

     public void serialEvent (SerialPortEvent event) {

       if (event.isRXCHAR () && event.getEventValue ()> 0) {

      try {

         // Отримуємо відповідь від пристрою, обробляємо дані

         String data = serialPort.readString (

          event.getEventValue ()

);

       } catch (SerialPortException ex) {

        System.out.println (ex);

       }

     }

   }

}

## Реалізація

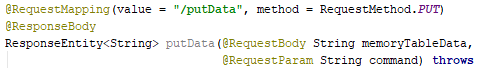
Вже була описана структуру проекта, зараз хочу описати конкретну реалізацію.

В папці config знаходяться класи конфігурації. ApplicationConfiguration – це клас позначений Spring анотацією @Configuration, це означає, що в ньому можна створювати біни. Також тут буде створюватися обгортка для SerialPortEngine, за допомогою якої буде здійснюватися підключення до віртуального COM порта.

В даній папці знаходиться і сама обгортка SerialPortEngineWrapper. Він знаходиться в цьому пакеті, адже починаючи роботу з програмою і модель, треба її підключити до комп’ютера, якщо підключення невдале, то користувачеві буде показане повідомлення про невдале з’єднання.

В SerialPortEngineWrapper описано алгоритм з’єднання, який описано в протоколі. Якщо з’єднання вдале, щоб кожен раз при підключенні до моделі не виконувати цей алгоритм, бажано зберегти назву порту до якого підключена модель процесора. Для цього в пакеті holders є клас PropertyHolder, яких зберігає глобальні змінні на протязі роботи програми.

Коли ми запустили нашу Web-програму, і клікнули на кнопку, наприклад WRITE, за допомогою Ajax, який було описано вище, на програму апаратного управління, через HTTP протокол передається запит. Команда і параметри. Як розповідалося вище, ці запити отримує сервлет. Але щоб не створювати багато класів, за нас це робить Spring, ми створимо лише один interface MicroProcWebApi, який матиме 3 методи: для GET, PUT і DELETE. Чому саме ці методи? Інші нам не потрібні. Для того щоб дістати дані з моделі, ми використовуємо метод GET, вказуючи, що саме хочемо взяти (стан пам’яті, стан шини адреси, регістрів і т.д.). Для зміни – метод PUT, а для очищення – DELETE, аналогічно методу GET у запиті передаватимемо дані, куди записати і що очистити.



В анотації @RequestMapping в value вказуємо доповнення до URL і method який буде використовуватися для передачі даних.

В @RequestBody вказуємо параметри які записати, а в @RequestParam – назву команди для моделі цифрового процесору.

У даного інтерфейсу є реалізація MicroProcWebApiImpl. В ньому ми передаємо дані на наступний рівень сервісу і результат відправляємо клієнту приводячи його ResponseEntity. Якщо при виконанні виникла помилка, відправку клієнту перехвачує CastomAdvisor, який формує зрозумілу для клієнта відповідь. Даний клас знаходиться в пакеті exceptions.

На сервіс частині є інтерфейс ProcService, в якому аналогічні методи як і в MicroProcWebApi. Також є реалізація ProcServiceImpl в якому визиваємо InstractionCallerApi, який безпосередньо працює COM портом. А сам ProcServiceImpl клас робить конвертацію даних.

І класс MicroprocApplication за допомогою якого ми запускаємо нашу прогрму.

# РОЗДІЛ 5. Результати тестування

Створений програмний пакет пройшов кілька етапів тестування з використанням апаратного емулятора, який замість моделі процесора виконував функції обміну відповідно до розробленого протоколом і системою команд. Емулятор реалізований на мікроконтролері ADuC831 і дозволяє підтримувати як режим штатного виконання протоколу, так і режим моделювання помилок в каналі передачі даних.

Багатогодинне тестування програмного комплексу на різних комп'ютерах з операційними системами Windows XP, Windows 7 і Windows 10 підтвердило Кросплатформені якості створеного програмного забезпечення, а також показало високу стійкість програмного пакета до виникнення нештатних ситуацій і порушень обміну.

# Висновок

## На прикладі комп’ютеризації учбової моделі цифрового процесору були дослідженні та описані проблеми організації обміну в системах контролю і управління, побудованих з використанням сучасних операційних систем сімейства Windows.

## При виконанні даної дипломної роботи була вивчена проблема взаємодії нових операційних систем з апаратними ресурсами комп’ютера. Запропоновано варіант побудови каналу обміну між комп’ютером та моделлю цифрового процесору, замінивши LPT-порт на COM-порт.

## Розроблено протокол взаємодії комп'ютера з керованим пристроєм і систему команд, який не вимагає безпосереднього доступу до апаратних регістрів порту, а ґрунтується на передачу даних з використанням уніфікованих протоколів обміну, підтримуваних операційними системами і вбудованими системними драйверами.

## Розроблена програма по принципу Web-серверу, з використанням різних бібліотек, таких як jQuery, Ajax, Spring, jSSC.

## Підводячи підсумок, можна сказати, що розроблений метод організації обміну між керуючим комп'ютером і веденим апаратним обладнанням дозволив вирішити проблему взаємодії моделі цифрового процесора з сучасними комп'ютерами широкого застосування і новими операційними системами. На думку розробників метод може успішно застосовуватися для побудови промислових комп'ютеризованих систем автоматичного контролю і управління, які не пред'являють високих вимог до швидкості обміну даними.

## Запропонована схема обміну передбачає можливість використання бездротових каналів передачі інформації для управління обладнанням в разі заміни апаратного драйвера послідовного зв'язку і при відповідній адаптації програмного забезпечення.

## Підтверджена тестуванням мультиплатформеність створеного програмного пакета дає підставу припускати збереження працездатності апаратно-програмного комплексу в разі переходу на нові (створювані) версії операційних систем.

Розроблена по всім експериментальним перевіркам система повністю відповідає розробленій моделі і передбачає подальший розвиток, так як потребує створення програмного забезпечення для самого інтерфейсного модуля. Це являється окремою роботою. І на сьогоднішній день дана тема, по якій я розпочала дослідження і проектну роботу, буде продовжена студенткою факультету комп’ютерних наук Харківського національного університету імені Каразіна.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ю. А. Журавель Модель цифрового процессора [Текст] / Ю. А. Журавель, С. Н. Рева // Вісник Харківського національного університету Серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління» №987 , 2011, с. 5-18.
2. А. А. Белуха Деякі нюанси роз'ємів принтерів типу „Centronics« [Текст] / А. А. Белуха РадіоАматор, № 10 2002, с. 22.
3. Ю. Л. Грабовська Демонстраційна програма упроавління моделлю цифрового процесора [Текст] / Ю. Л. Грабовська // кваліфікаційна робота на здобуття ступеня бакалавра Харківського національного університету імені Каразіна 2017 р.
4. Послідовний інтерфейс RS-232: [Електронний ресурс] // ТОВ Ринок мікроелектроніки 1998-2016 – URL: http://www.gaw.ru/html.cgi/txt/interface/rs232/ (Дата звернення 20.09.2018).
5. XMODEM protocol : [Електронний ресурс] //Wikipedia – URL: <https://en.wikipedia.org/wiki/XMODEM> (Дата звернення: 12.05.2018)
6. ZMODEM protocol : [Електронний ресурс] //Wikipedia – URL: <https://en.wikipedia.org/wiki/ZMODEM> (Дата звернення: 12.05.2018)
7. Web-додатки для локального застосування : [Електронний ресурс] // Пітер Сібах IBM developerWorks – URL: <https://www.ibm.com/developerworks/ru/library/wa-localwebsrv/index.html> (Дата звернення: 17.07.2018)
8. HTTP протокол : [Електронний ресурс] // Євген Рижков 2009-2018 xiper.net – URL: <http://xiper.net/learn/also-need-to-know/how-does-a-browser-communicate-with-the-server>
9. Простою мовою про HTTP : [Електронний ресурс] // Арсеній Хофман «[TM](https://tmtm.ru/)», 2006 – 2018. URL: https://habr.com/post/215117/#uri