**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**

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 (ссилка на бакалавр диплом) відображати на моніторі.

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).

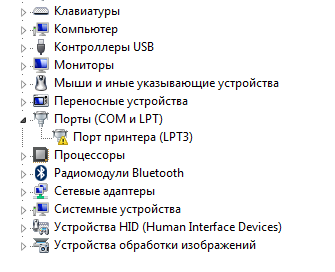
**РОЗДІЛ 2. Обґрунтування і вибір теоретичних та експериментальних методів дослідження поставлених задач**

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:

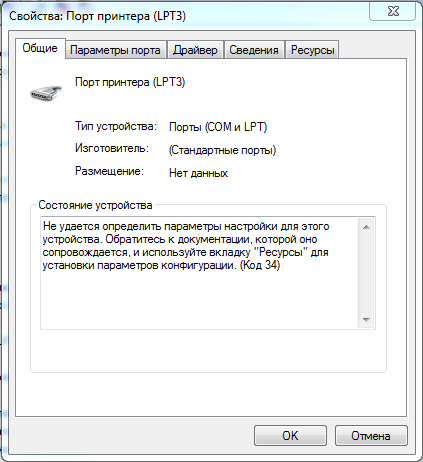
* коротку відстань передачі даних;
* передача через USB-порт здійснюється в послідовному коді, отже, для управління макетом необхідно пристрій декодування, що забезпечує запис даних в паралельні регістри і здійснює зворотне перетворення даних.

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. Також це дає нам можливість видалити 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 технічних систем і процесів, що досліджуються в роботі

## 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 хороший протокол, виділимо основні критерії, яким він повинен відповідати:

* правильне асоціативне сприйняття синтаксису команд оператором;
* інформації повинна бути представлена у вигляді ASCII кодів, щоб вона не була дуже важка, а сам процес передачі був швидкий;
* можливість у майбутньому розширювати протокол.

**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. Якщо передавати дані в шістнадцятирічній системі числення, то на кожен символ припадає 2 байт. Тобто, команда, наприклад, WRITE − займає аж 10 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, 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) [x].

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 [x].

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 підключення до макету**

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.

Але, мені здалось, що можна 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. |
| **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. | |
| **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-сервіс**

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.

**Чим допоможе 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. [https://www.ibm.com/developerworks/ru/library/wa-localwebsrv/index.html]

**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. [http://xiper.net/learn/also-need-to-know/how-does-a-browser-communicate-with-the-server]

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 (1.1)

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 (1.2)

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). [https://habr.com/post/215117/#uri]