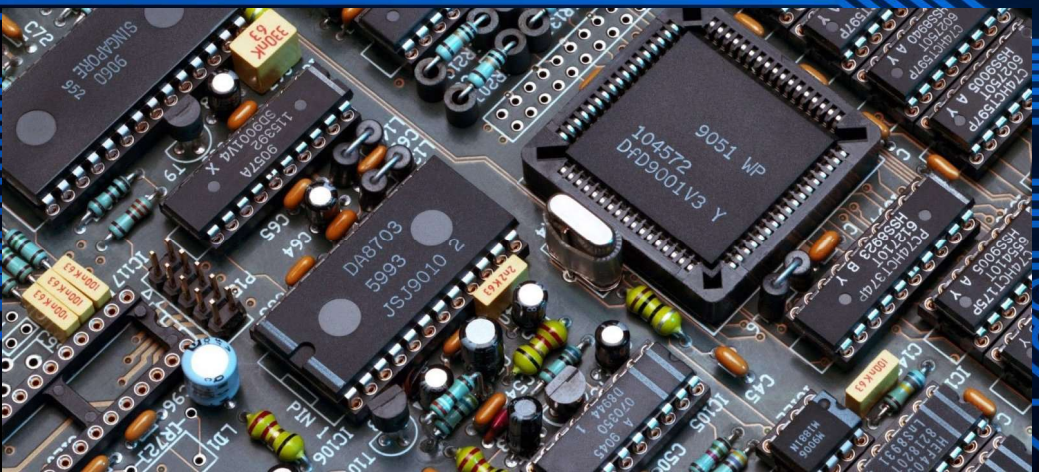
Безпека інтернет речей

Лекція №2



Лекцію проводить:

доц. Лимаренко Вячеслав Володимирович

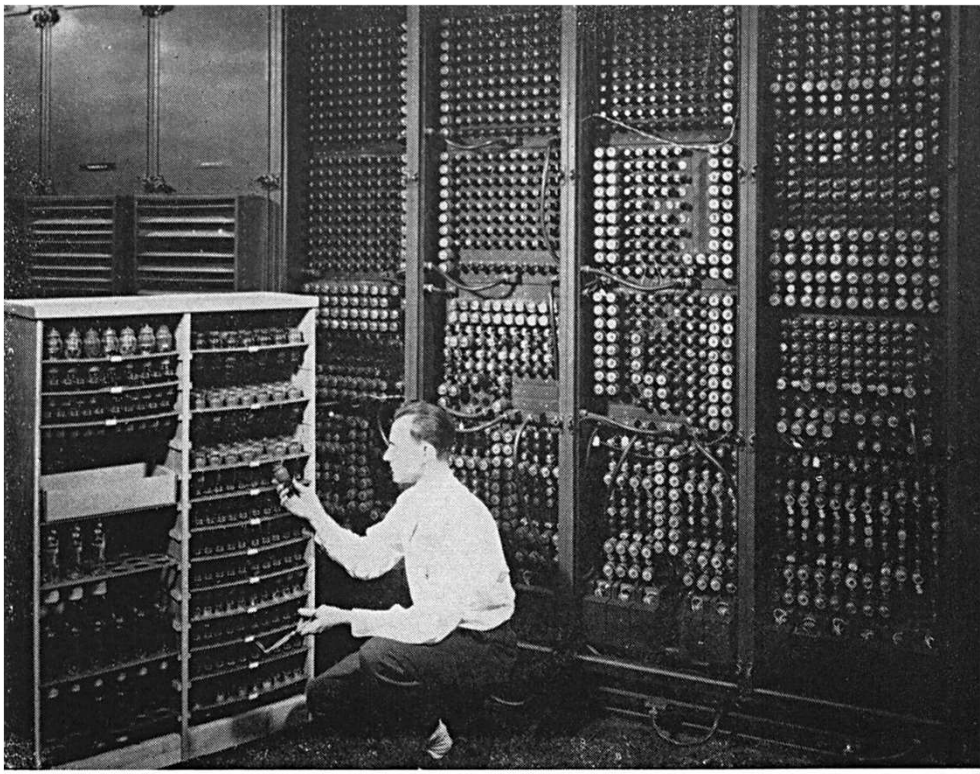
к.т. 066-0708586

Мікроконтролери і мікропроцесори. Історія появи

Характеристики та особливості перших комп'ютерів:
– ENIAC – перший комп'ютер;
– з’явився – 1946 рік;
– вага – 30т, займав ціле приміщення або 85 кубічних метрів об'єму в просторі;
– велике тепловиділення, енергоспоживання;
– постійні неполадки через роз'єми електронних ламп. Окисли призводили до зникнення контактів і лампи втрачали зв'язок із платою;
– вимагав постійного обслуговування.

До кінця 60-х у світі було близько 30 тисяч, як універсальних, так і «міні-комп’ютерів».

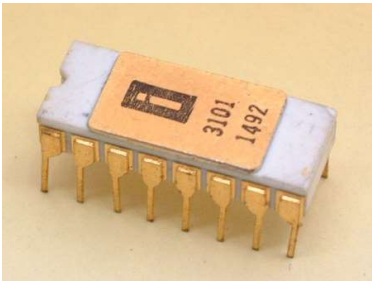
«Міні-комп’ютер» 60-х



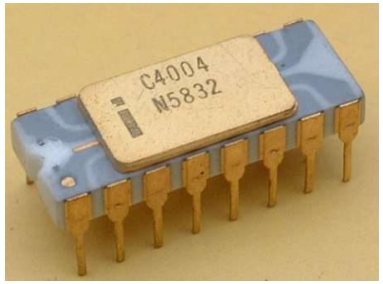
Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Мікроконтролери і мікропроцесори. Історія появи

– 1969 рік було винайдено прообраз інтернету – ARPANET (Advanced Research Projects Agency Network);
– 1907 рік роботи з детекторів та електролюмінесценції напівпровідників;
– 1940-ві діоди та транзистори;
– Роберт Нойс в 1959 винайшов інтегральну мікросхему.



Фірма Intel – внесла величезний внесок у розвиток мікроконтролерів. Засновники: Роберт Нойс, Гордон Мур та Ендрю Гроув. Заснована у 1968 році. На початку виробляла напівпроводникові запам'ятовуючі пристрої. Першим була МС «3101» – 64 розряди, діоди Шотки – біполярна статична ОЗП.



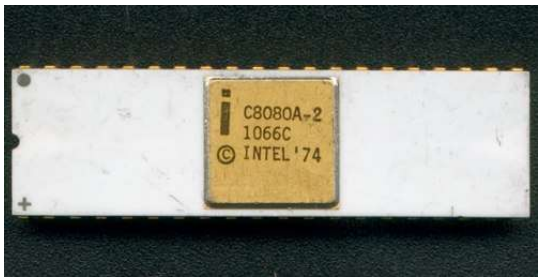
Наступним був винахід «4004» – мікропроцесор з 2300 н/п транзисторів у своєму складі, за продуктивністю не гірше ніж ENIAC, а розміром менше долоні. 4004 – 4-розрядна, р-МОП мікросхема.

Розробником архітектури першого мікропроцесора став Тед Хофф, системи команд Стен Мейзор. Федеріко Фейгін – спроектував кристал. Але спочатку компанія Intel не мала всіх прав на цей чіп, і, заплативши 60 000 доларів компанії Busicom, отримала повні права.

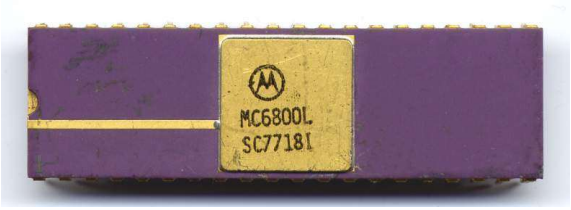
Наступним етапом став випуск у 1972 році процесора «8008». На відміну від попередньої моделі, він вже більше схожий на сучасні моделі. 8008 – 8 розрядний, має акумулятор, 6 регістрів загального призначення, покажчик стеку, 8 регістрів адреси, команди введення-виводу.

Мікроконтролери і мікропроцесори. Історія появи

– 1973 рік – було винайдено найбільш вдалу конфігурацію мікропроцесора, який досі є класичним – це 8-розрядний «8080».



– 1973 рік – поява Motorola з процесором «6800», n-МОП технологія, тришина структура з 16-розрядною шиною адреси. Більш потужна система переривань, для його живлення досить однієї напруги, а не трьох, як у «8080». Крім того, команди були простішими і коротшими.



Подальші кроки

Запровадження 16-розрядних мікропроцесорів. Першим був розроблений «8086» від Intel. Саме його використовували у компанії IBM для створення перших персональних комп'ютерів.

Процесор «68000» – 16-розрядна відповідь від «Motorola», використовувався в комп'ютерах ATARI та Apple.

Для широкої аудиторії у ролі ПК стали популярні ZX Spectrum. В них встановлювалися процесори «Z80», від Sinclair Research Ltd..

Мікроконтролери і мікропроцесори. Історія появи

Мікро-ЕОМ – головний крок масового застосування комп'ютерної автоматизації у сфері управління. Так як в автоматизації основне завдання – контроль та регулювання параметрів, то термін «контролер» закріпився і в цьому середовищі.

Після початку активного імпорту обчислювальної техніки в 1990-х назва «одокристальна мікро-ЕОМ» була витіснена словом «Мікроконтролер».

Перший патент в СРСР на однокристальні мікро-ЕОМ був виданий в 1971 М. Кочрену і Г. Буну, з Texas Instruments. З цих пір на кристалі кремнію крім процесора розміщували ще пам'ять і додаткові пристрої.

Кінець сімдесятих – це нова хвиля конкуренції між Intel та Motorola. Причиною цього стали дві презентації, а саме в 1976 році Intel випустила i8048, а Motorola в 1978 – MC6801, сумісний з раним мікропроцесором MC6800.

1980 рік, Intel випускає популярні й досі МК i8051. Це було зародження величезної родини, яка живе і досі. Для свого часу він мав немислимі 128 000 транзисторів. Це вчетверо перевищувало кількість в процесорі i8086.

Мікроконтролери і мікропроцесори

Мікропроцесор (microprocessor) — інтегральна схема, яка виконує функції центрального процесора (ЦП) або спеціалізованого процесора. Мікропроцесор є практично повним синонімом слова процесор, оскільки функціональні блоки, що на ранніх стадіях розвитку обчислювальної техніки займали цілу плату чи навіть шафу, тепер вміщаються в одну невеличку інтегральну схему із сотнями мільйонів транзисторів всередині. Гарним прикладом сьогодення є SoC — «система на чипі/кристалі».

Мікроконтролер (microcontroller), або однокристальний мікрокомп'ютер — виконаний у вигляді мікросхеми спеціалізований комп'ютер, що включає мікропроцесор, оперативну та постійну пам'ять для збереження виконуваного коду програм і даних, порти вводу-виводу і блоки зі спеціальними функціями (лічильники, компаратори, АЦП та інші). Використовується для керування електронними пристроями. По суті, це — однокристальний комп'ютер, здатний виконувати прості завдання. Використання однієї мікросхеми значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів. Більшість процесорів, що випускаються у світі — мікроконтролери.

У чому відмінність мікроконтролера від мікропроцесора?

Весь комп'ютерний функціонал мікропроцесора (Micro Processor Unit, MPU) міститься одному напівпровідниковому кристалі. За характеристиками відповідає центральному процесору комп'ютера ЦП (Central Processing Unit, CPU). Область його застосування – зберігання даних, виконання арифметико-логічних операцій, управління системами. Мікропроцесор отримує дані з вхідних периферійних пристроїв, обробляє їх та передає вихідним периферійним пристроям.

Мікроконтролер поєднує у собі мікропроцесор та необхідні опорні пристрої, об'єднані в одному чіпі. Якщо потрібно створити пристрій, що комунікує із зовнішньою пам'яттю або блоком ЦАП/АЦП, потрібно лише підключити джерело живлення з постійною напругою, мережу скидання та джерело тактової частоти.

Відомі сімейства сучасних мікроконтролерів

- MSC-51 виробництва Intel. Монокристальний МК з урахуванням Гарвардської архітектури. Основний представник цього сімейства 80C51, створений за технологією CMOS. І хоча ці контролери розроблені ще у 80-х роках минулого століття, але й досі широко використовуються. І сьогодні багато компаній, таких як Siemens, Philips та ін. випускають свої контролери з подібною архітектурою;
- PIC (Microchip). МК Гарвардської архітектури. В його основі лежить архітектура зі скороченим набором команд, вбудована пам'ять команд та даних, низьке енергоспоживання. У це сімейство входять понад 500 різних МК (8-ми, 16-ти, 32-бітні) з різними наборами периферії, пам'яті та іншими характеристиками;
- AVR (Atmel). Високошвидкісні контролери розроблені на власній архітектурі. Основою контролера є Гарвардський RISC-процесор із самостійним доступом до пам'яті програм та баз даних (Flash ПЗП). Кожен із 32 регістрів загального призначення може працювати як регістр-акумулятор та сукупність 16-бітних команд. Висока продуктивність в 1 MIPS на кожен МГц тактової частоти забезпечується за рахунок порядку виконання команд, що передбачає виконання однієї команди та одночасну підготовку до наступної. Для підтримки своєї продукції компанія Atmel випускає безкоштовне та якісне середовище розробки AtmelARM (ARM Limited), що розроблено на власній архітектурі. У сімейство входять 32-х та 64-бітові МК;
- ARM Limited займається лише розробкою ядер та їх інструментів, а ліцензії на виробництво продає іншим компаніям. Ці процесори споживають мало енергії, тому знаходять широке застосування у виробництві мобільних телефонів, ігрових консолей, маршрутизаторів тощо. До компаній, що викупили ліцензії, відносяться: STMicroelectronics, Samsung, Sony Ericsson та ін.
- STM (STMicroelectronics). 8-розрядні контролери (STM8) відносяться до категорії високонадійних виробів із низьким енергоспоживанням. У це сімейство входять контролери STM32, їх основу становить-32 бітний Cortex. Такі контролери мають добре збалансовану архітектуру і мають великі перспективи розвитку.

Відомі сімейства мікроконтролерів

MCS 51 (Intel)
•ESP8266 и ESP32 (Espressif)
•MSP430 (TI)
•ARM (ARM Limited)
- ST Microelectronics STM32 ARM-based MCUs
- ARM Cortex, ARM7 и ARM9-based MCUs
- Texas Instruments Stellaris MCUs
- NXP ARM-based LPC MCUs
- Toshiba ARM-based MCUs
- Analog Devices ARM7-based MCUs
- Cirrus Logic ARM7-based MCUs
- Freescale Semiconductor ARM9-based MCUs
- Silicon Labs EFM32 ARM-based MCUs

•AVR (Atmel)
- ATmega
- ATtiny
- XMega

•PIC (Microchip)
•STM8 (STMicroelectronics)
•C8051F34x
•RL78 (Renesas Electronics)

MCS 51 (Intel)

62E40

ESP32-WROOM-32U-8

STM32

MSP430

ARM Cortex

Microchip

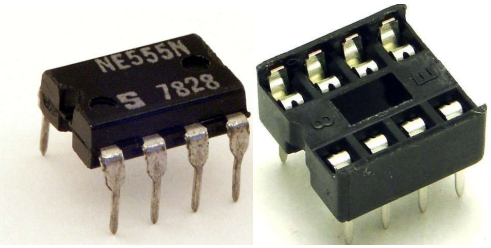
C8051F3

ATmega

RL78

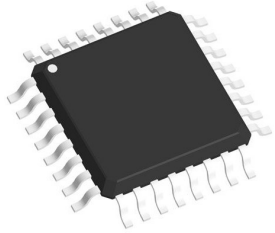
Типи корпусів мікроконтролерів

Зовнішніх відмінностей МК від інших мікросхем немає. Кристали розміщені в корпусах із певною кількістю виходів. Всі МК випускаються тільки в трьох типах корпусів:



- Корпус DIP має два ряди виходів. Відстань між ними 2,54 мм. Виходи вставляються усередину отворів на контактних майданчиках;

- Корпус SOIC. Він підходить для монтажу, який передбачає поверхневе припаювання виходів до контактного майданчика. Відстань між виходами 1,27 мм;



- Корпуси QFP (TQFP). Виходи розташовані з усіх боків. Відстань між ними в 3 рази менша, ніж у DIP. Корпус має квадратну форму. Призначаються тільки для поверхневого паяння;

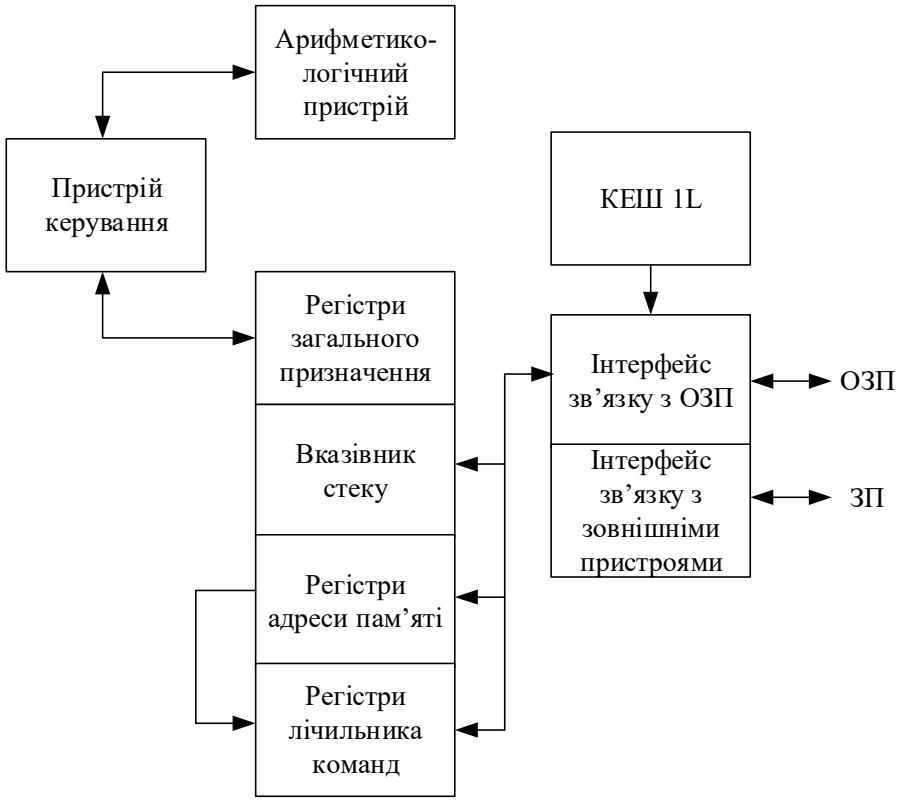
- Корпус QFN. Найменший у порівнянні з попередніми. Контакти розташовані у 6 разів частіше, ніж у DIP. Мають велике поширення в промисловому виробництві, тому що дозволяють значно зменшити габарити приладів, що випускаються.

Поділ за системами команд

RISC-архітектура або скорочена система команд. Орієнтована на швидке виконання базових команд за 1, рідше 2 машинні цикли, а також має велику кількість універсальних регістрів, і більш довгий спосіб доступу до постійної пам'яті. Архітектурна характерна для систем під управлінням UNIX.

В даний час багато архітектур процесорів є RISC-подібними, наприклад, ARM, DEC Alpha, SPARC, AVR, MIPS, POWER і PowerPC. Процесори архітектури x86 раніше були CISC-процесорами, проте, починаючи з Intel Pentium Pro (1995), є CISC-процесорами з RISC-ядром. Вони безпосередньо перед виконанням перетворюють CISC-інструкції x86-процесорів на більш простий набір внутрішніх інструкцій RISC.

Після того, як процесори архітектури x86 були переведені на суперскалярну RISC-архітектуру, можна сказати, що більшість існуючих процесорів засновано на архітектурі RISC.



Поділ за системами команд

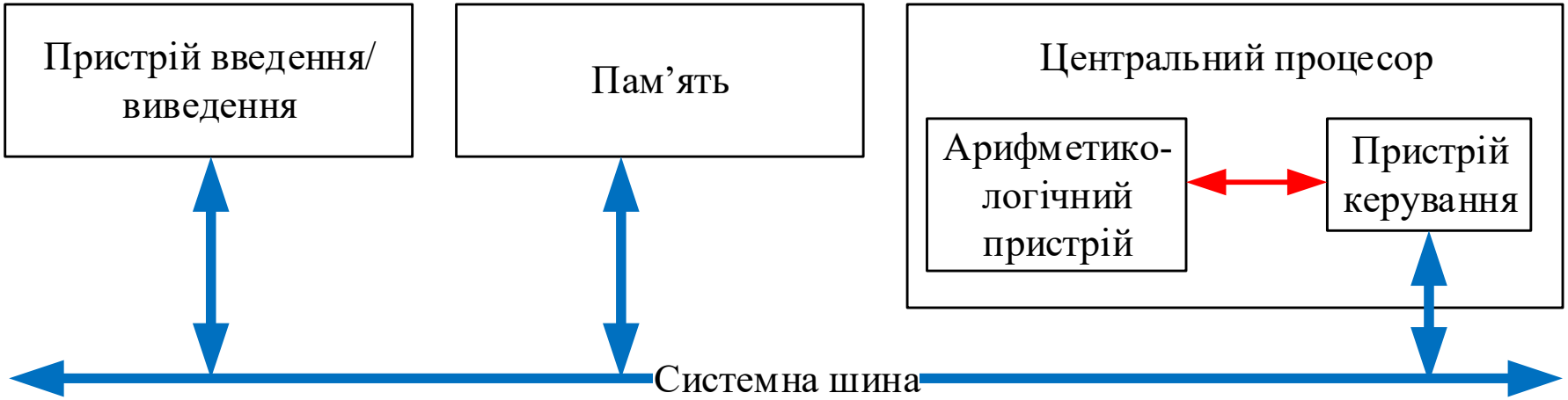
CISC-архітектура, або повна система команд, характерна пряма робота з пам'яттю, більша кількість команд, мала кількість регістрів (орієнтована на роботу з пам'яттю), тривалість команд від 1 до 4 машинних циклів. Приклад – процесори Intel застарілих поколінь.

Зараз в «чистому вигляді» майже не використовується.

Історично була першою.

Поділ за типом пам'яті

Прінстонська архітектура (архітектура Фон-Неймана). Основна її риса – загальна область пам'яті для команд і даних, при роботі з такою архітектурою в результаті помилки програміста дані можуть записатися в область пам'яті програм і подальше виконання програми стане неможливим. Пересилання даних та вибірка команди не може здійснюватися одночасно з тих самих причин. Розроблена у 1945 році.

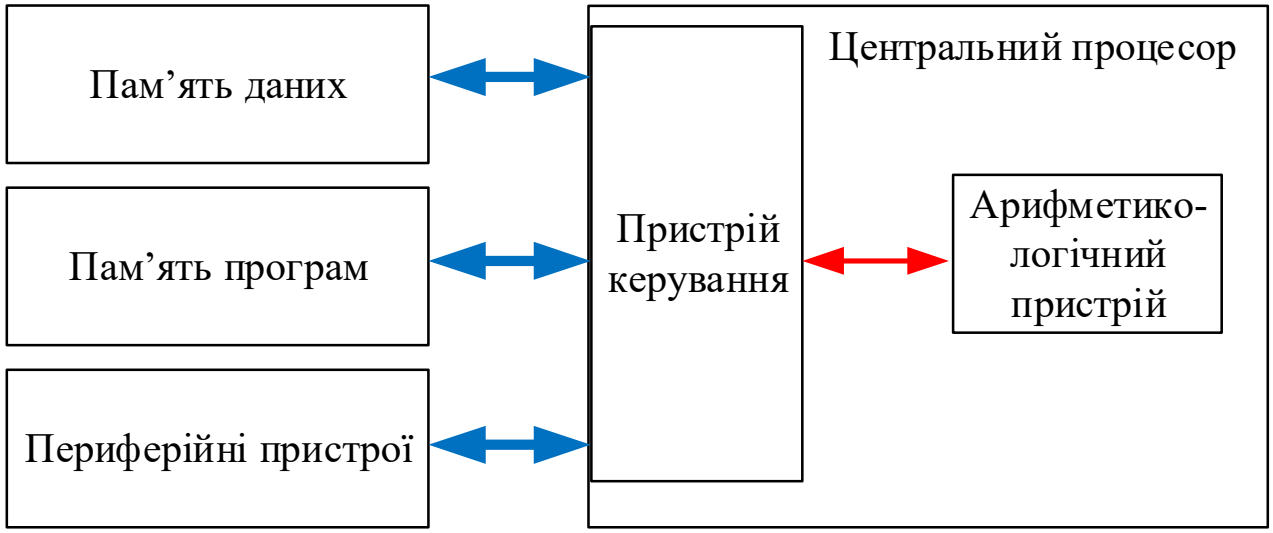


Архітектура фон Неймана має низку важливих переваг.:

- наявність загальної пам'яті дозволяє оперативно перерозподіляти її обсяг для зберігання окремих масивів команд, даних та реалізації стека залежно від розв'язуваних завдань. Таким чином, забезпечується можливість ефективнішого використання наявного обсягу оперативної пам'яті в кожному конкретному випадку застосування;
- використання загальної шини передачі команд і даних значно спрощує налагодження, тестування і поточний контроль функціонування системи, підвищує її надійність.

Поділ за типом пам'яті

Гарвардська архітектура – роздільна пам'ять даних та пам'ять програм, використовувалася в перше на комп'ютерах сімейства Mark. Розроблена у 1944 році.



Гарвардська архітектура широко застосовується у внутрішній структурі сучасних високопродуктивних мікропроцесорів, де використовується окрема кеш-пам'ять для зберігання команд та даних.

У зовнішній структурі більшості мікропроцесорних систем реалізуються принципи Прінстонської архітектури.

Основні типи мікроконтролерів та їх архітектура

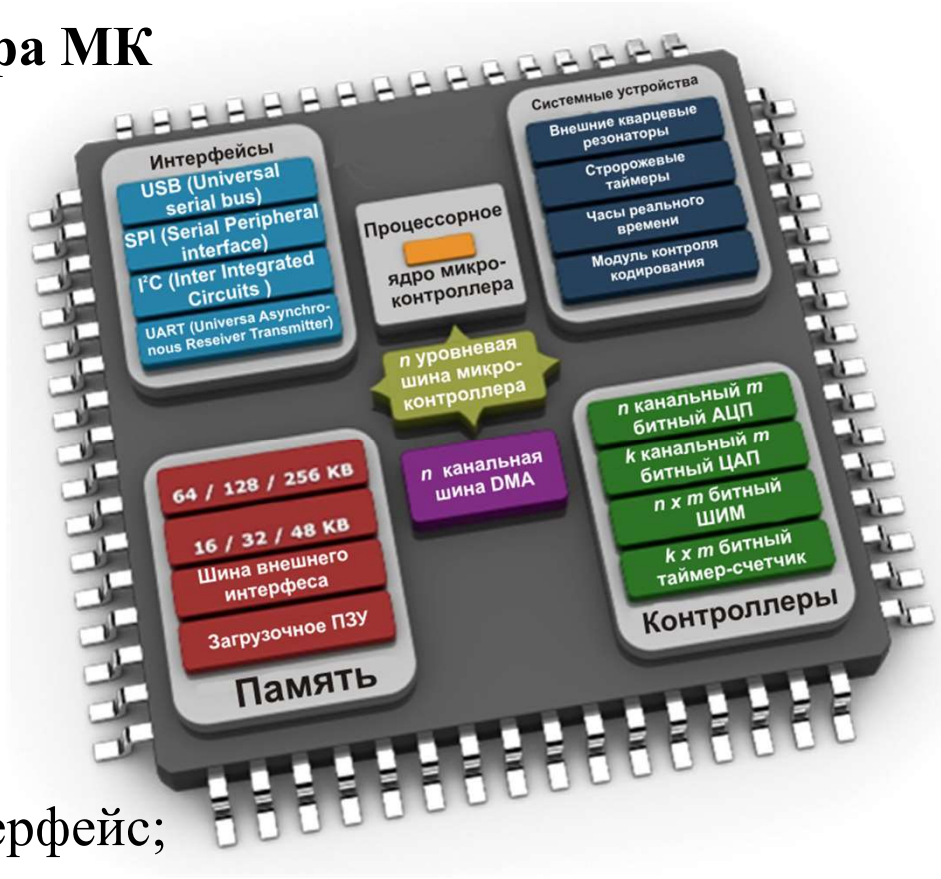
Основні типи:

- вбудовувані (embedded) 8-розрядні МК;
- 16- та 32-розрядні МК;
- цифрові сигнальні процесори.

До складу цих МК входять:

- схема початкового запуску процесора (Reset);
- генератор тактових імпульсів;
- центральний процесор;
- пам'ять програм (E(E)PROM) та програмний інтерфейс;
- засоби введення/виведення даних;
- таймери, що фіксують кількість командних циклів.

Загальна структура МК



Основні типи:

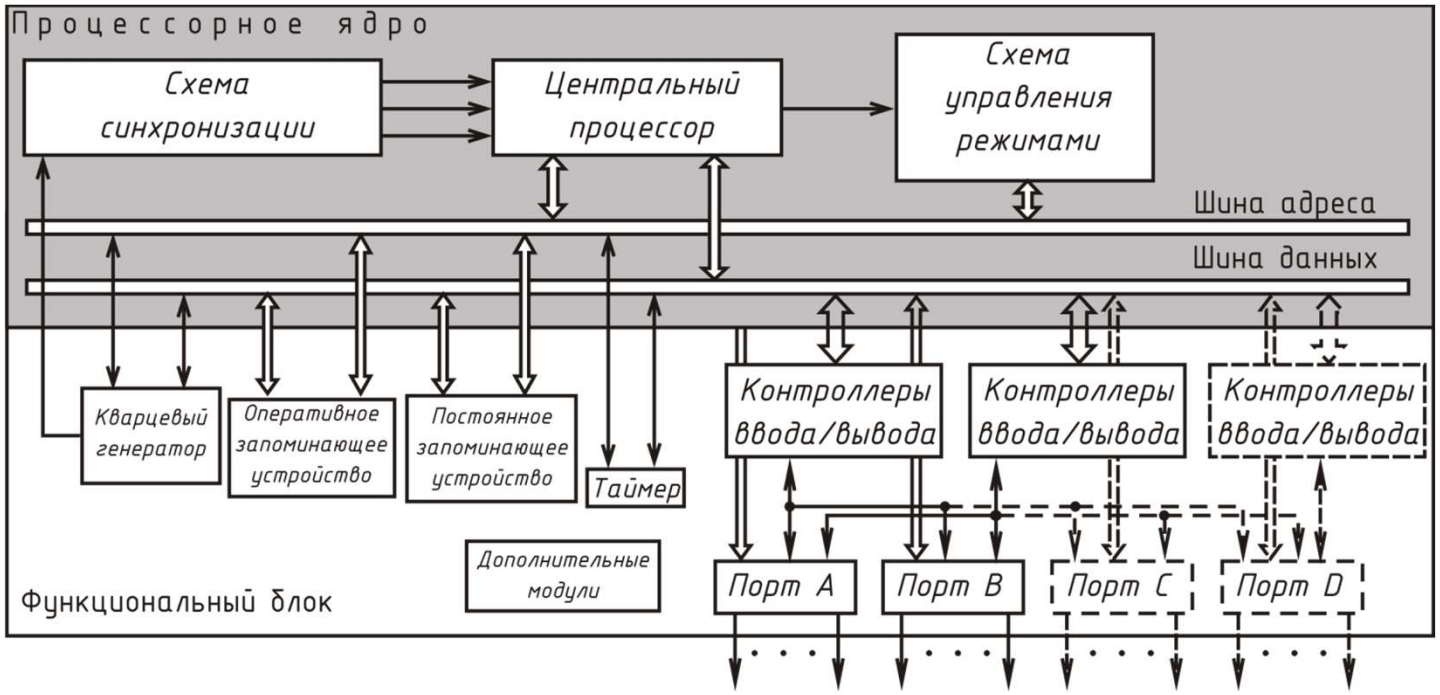
- вбудовувані (embedded) 8-розрядні МК;
- 16- та 32-розрядні МК;
- цифрові сигнальні процесори.

До складу цих МК входять:

- схема початкового запуску процесора (Reset);
- генератор тактових імпульсів;
- центральний процесор;
- пам'ять програм (E(E)PROM) та програмний інтерфейс;
- засоби введення/виведення даних;
- таймери, що фіксують кількість командних циклів.

Модульний принцип побудови мікроконтролерів

Широке розмаїття моделей мікроконтролерів, можливість розробки та виробництва нових моделей за короткий термін забезпечує модульний принцип побудови мікроконтролерів. При модульному принципі побудови всі мікроконтролери одного сімейства повинні містити в собі базовий функціональний блок, який однаковий для всіх мікроконтролерів сімейства, і функціональний блок, що змінюється, який відрізняє моделі мікроконтролерів в межах одного сімейства.



Режими роботи МК

Активний режим. У цьому режимі мікроконтролер виконує основну програму. Усі модулі мікроконтролера перебувають у робочому стані, спостерігається максимальне споживання енергії від джерела живлення;

Режим зниженого енергоспоживання. В даний час однією з основних вимог, що висуваються до пристроїв є знижене енергоспоживання. У сучасних мікроконтролерах можливі два напрямки реалізації режиму зниженого енергоспоживання. Перший полягає у зменшенні споживаного струму в робочому режимі. Досягається це відключенням всіх модулів, що не використовуються, в робочому режимі. Суть другого напряму полягає у повному чи частковому зупиненні процесора на період простою;

Режим початкового запуску (скидання) та переривання. Режим початкового запуску необхідний, щоб забезпечити запуск основної програми тільки після того, як буде встановлена потрібна напруга живлення, а також кварцовий резонатор вийде на потрібну частоту роботи. Як правило для початкового запуску мікроконтролера використовують комбіновані схеми, що забезпечують автоматичне формування сигналу скидання відразу ж після включення живлення і ручне скидання у разі зациклювання основної програми. Як правило, в режимі початкового запуску відбувається примусове обнулення всіх регістрів і прапорів.

Бібліотеки периферійних модулів

Бібліотека кожного сучасного сімейства мікроконтролерів включає модулі п'яти функціональних груп:
– модулі пам'яті;
– модулі периферійних пристроїв;
– модулі вбудованих генераторів синхронізації;
– модулі контролю за напругою живлення та ходом виконання програми;
– модулі внутрішньосхемного налагодження та програмування.

Основні типи групи модулів периферійних пристроїв

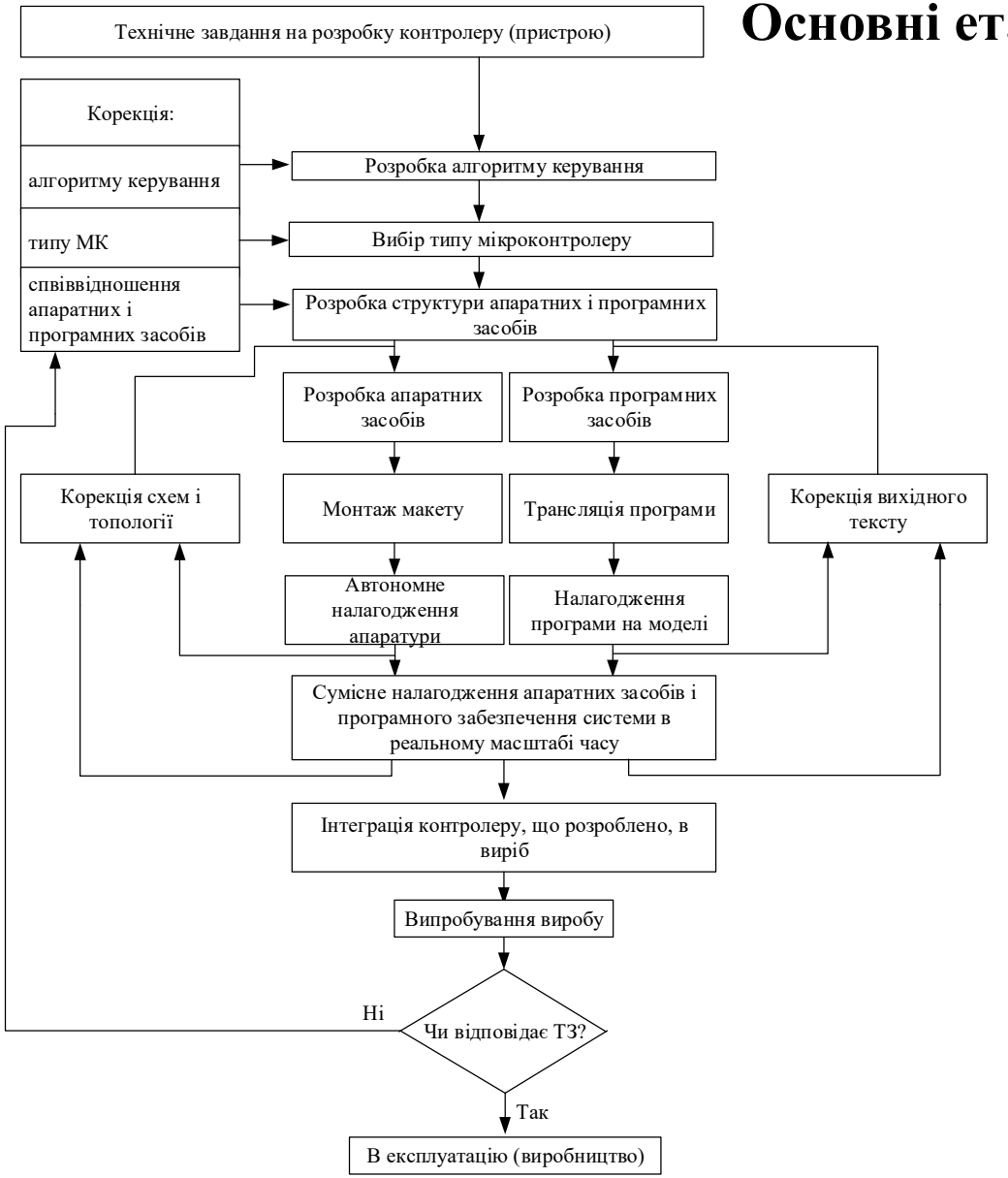
– паралельні порти вводу/виводу. З апаратної точки зору портом вода/виводу є кілька виходів (пінів або іншими словами ніжок мікросхеми) загальна кількість яких визначається розрадністю переданих або одержуваних даних. З точки зору програми для мікроконтролера кожен порт – це кілька спеціальних регістрів (змінних), здійснюючи читання або запис даних, в які можна змінювати стан або режим роботи виходів мікроконтролера. Залежно від програмного налаштування портів мікроконтролера вони можуть бути тільки портами введення, тільки портами вводу або одночасно портами введення і виведення;
– таймери-лічильники, таймери періодичних переривань, процесори подій;
– контролери послідовного інтерфейсу зв'язку декількох типів (UART, SCI, SPI, I2C, CAN, USB);
– UART, Univercal Asynchronous Receiver/Transmitter – універсальний асихронний приймач/передавач. З допомогою цього контролера здійснюється управління перетворенням даних прийнятого мікроконтролером паралельного формату передачі в послідовний і навпаки;
– аналого-цифрові перетворювачі;
– цифрово-аналогові перетворювачі;
– контролери рідкокристалічних індикаторів та світлодіодні матриці.

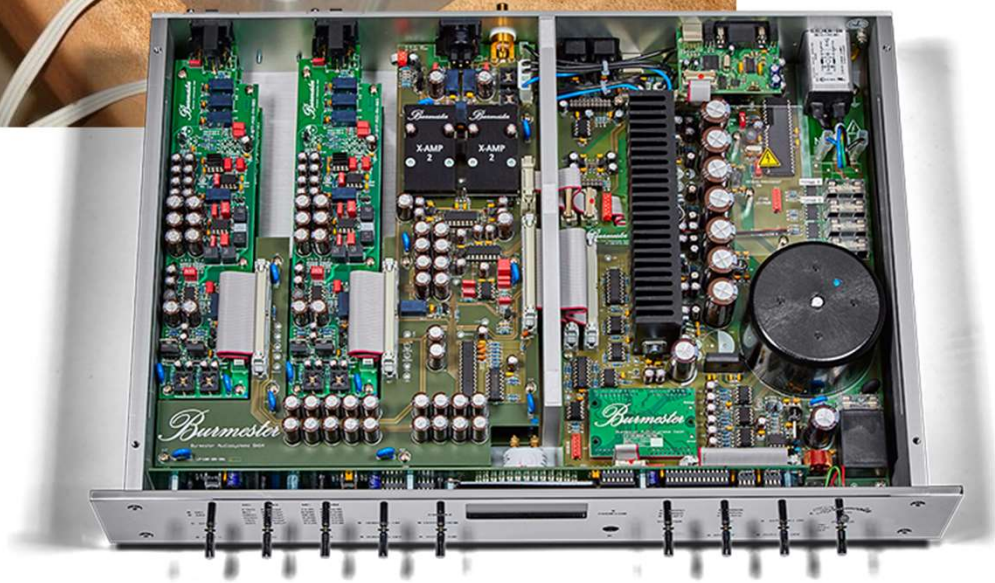
Процесорне ядро мікроконтролера

Процесорне ядро є нерозривною єдністю трьох складових його технічного рішення:

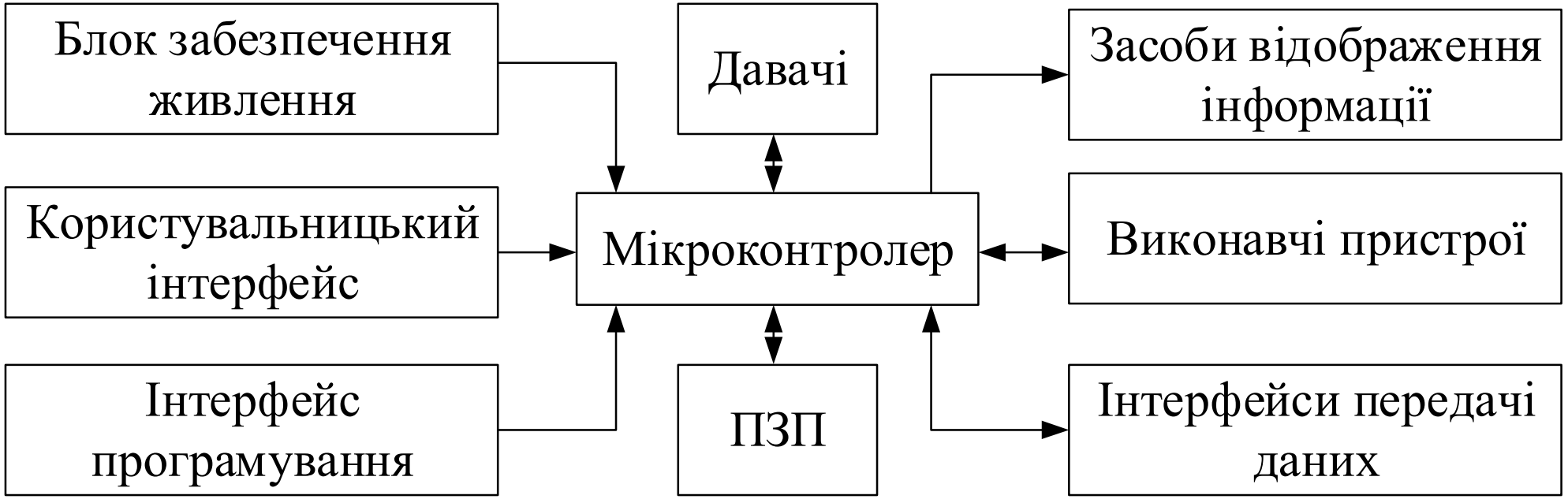
1) архітектури центрального процесора з властивим їй набором регістрів для зберігання проміжних даних, організацією пам'яті та способами адресації операндів у просторі пам'яті, системою команд, що визначає набір можливих дій над операндами, організацією процесу вибірки та виконання команд;
2) схемотехнічного втілення архітектури, яке визначає послідовність переміщення даних по внутрішніх магістралях мікроконтролера між регістрами, арифметично-логічними пристроями та осередками пам'яті в процесі виконання кожної команди;
3) технології виготовлення напівпровідникового кристала мікроконтролера, яка визначає складність розміщеної схеми, визначає допустиму тактову частоту та енергію споживання.

Основні етапи розробки мікроконтролерного пристрою





Уніфікована структурна схема мікроконтролерних пристроїв



Примітки:
– блок забезпечення живлення – бажано з широким спектром вхідної напруги (3,3-12В), захистом від короткого замикання та переполюсовки;
– користувальницький інтерфейс – кнопки, перемикачі, регулятори і т.п.;
– давачі – цифрові або аналогові (за потреби і наявності);
– ПЗП – карти пам’яті, мікросхеми ПЗП, жорсткі диски і т.п.;
– інтерфейси передачі даних, інтерфейс програмування, засоби відображення інформації, виконавчі пристрої – будь які, за потреби.

Що потрібно для програмування мікроконтролера

Щоб мікроконтролер міг виконувати необхідні функції та вирішувати певні завдання, його необхідно запрограмувати. Шлях програмування проходить кілька етапів:

1. Перед тим як розпочати написання коду програми, треба визначитися з кінцевою метою;
2. Складається алгоритм роботи програми;
3. Безпосереднє написання програмного коду. Коди пишуться, наприклад, мовою С або Асемблер;
4. Компіляція програми, тобто, переведення її в двійкову або шістнадцяткову систему 1 і 0. Тільки так її зможе зрозуміти МК;
5. Відкомпільований код записують у пам'ять контролера;
6. Прошивають МК за допомогою програматора. Вони бувають різних типів підключення, наприклад, через COM чи USB порт. Найпростіший і найдешевший програматор USBASP.
7. Тестування та налагодження МК на реальному пристрої.

Синтаксис мови асемблера для мікроконтролерів

Мова Ассемблер – це машинно-залежна мова, тобто набір інструкцій (команд) мови залежить від архітектури МК: кількості регістрів, видів та обсягу пам'яті, набору периферійних пристроїв.

Щоб задовольнити запити споживачів (розробників апаратури), випускають мікроконтролери з різним складом периферійних пристроїв та обсягом пам'яті. Якщо у них загальне обчислювальне ядро, то вони відносяться до одного сімейства МК, який має повний набір команд для найскладнішого МК (з повним набором периферійних пристроїв і максимальним обсягом пам'яті). Чим менше периферійних пристроїв входять до складу конкретного МК, тим менше його набір інструкцій, тобто, із загального набору інструкцій виключаються ті, що належать до відсутніх периферійних пристроїв.

Для більш точної інформації щодо команд конкретного МК звертайтеся до опису команд інструкцій у документації. Для конкретного МК повний набір команд буде усічено.

Крім команд МК для зручності програмування, стиснення вихідного тексту, поліпшення наочності та читання програм у мову Ассемблер введено директиви, оператори, мітки, коментарі, тощо. Цей набір доповнюється та змінюється у міру вдосконалення мови. Він обробляється препроцесором. Препроцесор обробляє текст вихідного коду програми до його компіляції, перетворюючи всі додаткові інструкції команди МК. Він викликається автоматично під час запуску компілятора.

Синтаксис асемблера для МК AVR

Вихідні файли мови Асемблер – це текстові файли, що складаються з рядків, що містять інструкції (команди), мітки та директиви. Інструкції та директиви, як правило, мають один або кілька операндів. Вхідний рядок може мати одну з чотирьох форм:

[мітка:] директива [операнди] [Коментар]
[мітка:] інструкція [операнди] [Коментар]
Коментар
Пустий рядок.

Будь-який рядок може починатися з мітки, яка є набором символів, що закінчується двокрапкою (:). Мітки використовуються для вказівки місця, в яке передається керування під час переходів, а також для надання імен змінних.

Коментар має таку форму:

; [Текст]

Позиції у квадратних дужках необов'язкові. Текст після крапки з комою (;) і до кінця рядка ігнорується компілятором.

Приклад коду:

```
label:  .EQU var1=100       ; Встановлює var1 рівним 100 (Це директива)
        .EQU var2=200       ; Встановлює var2 рівним 200
test:   rjmp test     ; Нескінченний цикл (Це інструкція); Рядок з одним тільки коментарем.
```

Синтаксис асемблера для МК AVR

Ключові слова. Вбудовані ідентифікатори (ключові слова) зарезервовані і не можуть бути перевизначені. Ключові слова включають усі інструкції (команди), регістри R0-R31 та X, Y, Z та всі функції. Ключові слова Асемблер розпізнає незалежно від регістру, в якому вони набрані, за винятком чутливих до регістру опцій, у яких ключові слова набираються в нижньому регістрі (тобто «add» зарезервовано, а «ADD» – ні).

Директиви препроцесора. Асемблер вважає директивами препроцесора всі рядки, що починаються з «#» (або перший непустий символ у рядку, оскільки пробіли та символи табуляції ігноруються).

Коментарі. Додатково до класичних коментарів Асемблера, що починаються з «;», Асемблер визнає коментарі в стилі мови С:

```
. . . . . ; Решта рядка є коментарем.

// Подібно до «;» решта рядка є коментарем.

/* Блок коментарів може розташовуватися в декількох рядках. Цей стиль коментарів не може бути вкладеним */
```

Синтаксис асемблера для МК AVR

Продовження рядка. Подібно до С, рядки вихідних кодів можуть бути продовжені за допомогою «\» – зворотної косої риси в кінці рядка. Це особливо корисно для великих макровизначень препроцесора і для довгих директив .db.

Приклад:

```
.db 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 11,12, 21, 214,235,634,  \n', 0, 2, \
12, 3,"'Це продовження верхнього рядка ", '\n', 0, 3, 0
```

Рядки та символьні константи. Рядок, укладений у подвійні лапки ("), може бути використаний тільки разом з директивою .db. Рядок передається буквально, ніякі службові символи і NUL-закінчення не розпізнаються. Символьні константи, укладені в одиночні лапки ('), можуть використовуватися скрізь, де допустиме ціле вираження. Службові символи в С-стилі розпізнаються з тим самим значенням, як у С.

Асемблером також розпізнаються \ooo (ooo = вісімкове число) і \xhh (hh = шістнадцяткове число).

Приклади:

```
.db "Hello\n"  // - еквівалент:
.db 'H', 'e', 'l', 'l', 'o', '\\', 'n,
```

Для того щоб створити еквівалент С-рядку "Привіт, світ \n", роблять наступним чином:

```
.db " Hello, world", '\n', 0
```

Синтаксис асемблера для МК AVR

Система команд. Система команд мікроконтролерів ATMEL сімейства AVR дуже велика і водночас ефективна. Однією з відмінних рис мікроконтролерів AVR є те, що майже всі команди виконуються за 1 тактовий цикл. Виняток становлять команди переходу. Це суттєво збільшує продуктивність мікроконтролера навіть за відносно невисокій тактовій частоті.

Усі команди можна поділити на 5 типів:

1) арифметичні команди;
2) логічні команди;
3) команди переходу;
4) команди передачі;
5) побітові команди та команди тестування бітів.

Директиви асемблера. Асемблер підтримує безліч директив. Директиви не транслюються безпосередньо у коди операції. Навпаки, вони використовуються, щоб коригувати розташування програми у пам'яті, визначати макрокоманди, ініціалізувати пам'ять тощо. Тобто це вказівки самому асемблеру, а не команди мікроконтролера.

Синтаксис асемблера для МК AVR.
Таблиця 1. Директиви асемблера

Директива	Опис
BYTE	Зарезервувати байт під змінну
CSEG	Сегмент кодів
DB	Задати постійні байти у пам'яті
DEF	Задати символічне ім'я регістру
DEVICE	Задати для якого типу мікроконтролера компілювати
DSEG	Сегмент даних
DW	Задати постійне слово в пам'яті
EQU	Встановіть символ, що дорівнює виразу
ESEG	Сегмент EEPROM
EXIT	Вихід із файлу
INCLUDE	Увімкнути вихідний код з іншого файлу
LIST	Включити генерацію .lst-файлу
NOLIST	Вимкнути генерацію .lst-файлу
ORG	Початкова адреса програми
SET	Встановіть символ, що дорівнює виразу

Синтаксис усіх директив наступний: .[директива]

Тобто перед директивою має стояти крапка. Інакше асемблер сприймає це як мітку.

Синтаксис асемблера для МК AVR.

CSEG – Code segment. Директива CSEG вказує на початок сегмента кодів. Файл, що асемблюється може мати кілька кодових сегментів, які будуть об'єднані в один при асемблюванні.

Синтаксис:

```
.CSEG
```

Приклад:

```
.DSEG                   ; Початок сегменту даних
vartab: .BYTE 4         ; Резервується 4 байти в СОЗП
.CSEG                   ; Початок сегменту кодів
const:  .DW 2           ; Записати 0x0002 в програмній пам'яті
        mov r1,r0       ;  Щось робити
```

DSEG – Data Segment. Директива DSEG вказує на початок сегмента даних. Файл, що асемблюється може містити кілька сегментів даних, які потім будуть зібрані один при асемблюванні. Зазвичай сегмент даних складається лише з директив BYTE та міток.

Синтаксис:

```
.DSEG
```

Приклад:

```
.DSEG                   ; Початок сегменту даних
var1:   .BYTE 1         ; Резервувати 1 байт під змінну var1
table:  .BYTE tab_size   ; Резервувати tab_size байтів.
        .CSEG
        ldi r30,low(var1)
        ldi r31,high(var1)
        ld r1,Z
```

Синтаксис асемблера для МК AVR.

ESEG – EEPROM Segment. Директива ESEG вказує на початок сегмента пам'яті EEPROM. Файл, що асемблюється, може містити кілька сегментів EEPROM, які будуть зібрані в один сегмент при асемблюванні. Зазвичай сегмент EEPROM складається з DB та DW директив (і міток). Сегмент EEPROM пам'яті має власний лічильник. Директива ORG може використовуватися для розміщення змінних у потрібній області EEPROM.

Синтаксис:

```
.ESEG
```

Приклад:

```
.DSEG                    ; Початок сегменту даних
var1:    .BYTE 1         ; Резервувати 1 байт під змінну var1
table:   .BYTE tab_size  ; Зарезервировать  tab_size байт.
         .ESEG
eevar1: .DW 0xffff       ; Записати 1 слово в EEPROM
```

ORG – Встановити адресу початку програми. Директива ORG надає значення локальним лічильникам. Використовується лише спільно з директивами .CSEG, .DSEG, .ESEG.

Синтаксис:

```
.ORG  адреса
```

Приклад:

```
         .DSEG           ; Початок сегменту даних
         .ORG 0x37       ; Установити адресу СОЗП  на 37h
variable: .BYTE 1        ; Зарезервувати байт СОЗП по адресі 37h
         .CSEG
         .ORG 0x10       ; Встановити лічильник команд на адресу 10h
         mov r0,r1       ; Щось робити
```

Синтаксис асемблера для МК AVR.

DB – визначити байт(и) у програмній пам'яті або EEPROM. Директива DB резервує ресурси пам'яті в програмній пам'яті або EEPROM. Директиві має передувати мітка. DB задає список виразів, і має містити принаймні один вираз. Розміщувати директиву слід у сегменті кодів або EEPROM сегменті. Список виразів є послідовністю виразів, розділених комами. Кожен вираз має бути величиною між – 128 та 255. Якщо директива вказується в сегменті кодів і список виразів містить більше двох величин, то вирази будуть записані так, що 2 байти розміщуватимуться в кожному слові Flash-пам'яті.

Синтаксис: LABEL: .DB список варазів

Приклад:

```
        .CSEG
consts: .DB 0, 255, 0b01010101, -128, 0xaa
        .ESEG
const2: .DB 1,2,3
```

DW – визначити слово(а) у програмній пам'яті або EEPROM. Директива DW резервує ресурси пам'яті в програмній пам'яті або EEPROM. Директиві має передувати мітка. DW задає список виразів, і має містити принаймні один вираз. Розміщувати директиву слід у сегменті кодів або EEPROM сегменті. Список виразів є послідовністю виразів, розділених комами. Кожне вираз має бути величиною між -32768 та 65535.

Синтаксис: LABEL: .DW список виразів

Приклад:

```
        .CSEG
varlist: .DW 0, 0xffff, 0b1001110001010101, -32768, 65535
        .ESEG
eevarlst: .DW 0,0xffff,10
```

Синтаксис асемблера для МК AVR.

DEF – присвоїти ім'я регістру. Директива DEF дозволяє присвоїти символічне ім'я регістру. Регістр може мати кілька символічних імен.

Синтаксис: `.DEF Ім’я=Регістр`

Приклад:

```
.DEF temp=R16
.DEF ior=R0
.CSEG
ldi temp,0xf0          ; Завантажити 0xf0 в регістр temp
in ior,0x3f            ; Прочитати SREG в регістр ior
eor temp,
```

EQU – присвоїти ім’я. Директива EQU надає значення мітці. Ця мітка може бути використана в інших виразах. Значення цієї позначки не можна змінити або перевизначити.

Синтаксис: `.EQU мітка=вираз`

Приклад:

```
.EQU io_offset = 0x23
.EQU porta     = io_offset + 2
.CSEG                  ; Початок сегменту кодів
clr r2                 ; Очистити регістр r2
out porta,r2           ; Записати до порту A
```

Синтаксис асемблера для МК AVR.

INCLUDE – вставити інший файл. Директива INCLUDE каже Асемблеру почати читати з іншого файлу. Асемблер буде асемблювати цей файл до кінця файлу або до директиви EXIT. Файл може сам включати директиви INCLUDE.

Синтаксис: .INCLUDE "ім'я файлу"

Приклад:

```
; iodefs.asm: .EQU sreg   = 0x3f      ; Регістр статусу
             .EQU sphigh = 0x3e      ; Старший байт покажчика стека
             .EQU splow  = 0x3d;     ; Молодший байт покажчика стека
; incdemo.asm
             .INCLUDE iodefs.asm     ; Увімкнути файл "iodefs.asm"
             in r0,sreg              ; Прочитати регістр статусу
```

EXIT – вийти із файлу. Директива EXIT дозволяє асемблеру зупинити асемблювання поточного файлу. Зазвичай асемблер працює до кінця файлу. Якщо він зустріне директиву EXIT, то продовжить асемблювати з рядка, що йде за директивою INCLUDE.

Синтаксис: .EXIT

Приклад:

```
.EXIT  ; вийти з цього файлу
```

DEVICE – вказати для якого мікроконтролера асемблювати. Директива дозволяє користувачеві повідомити асемблеру, для якого типу пристрою пишеться програма. Якщо асемблер зустріне команду, яка не підтримується зазначеним типом мікроконтролера, буде видано повідомлення. Також повідомлення з'явиться у випадку, якщо розмір програми перевищить обсяг наявної у пристрою пам'яті.

Синтаксис:

Синтаксис асемблера для МК AVR.

DEVICE – вказати для якого мікроконтролера асемблювати. Директива дозволяє користувачеві повідомити асемблеру, для якого типу пристрою пишеться програма. Якщо асемблер зустріне команду, яка не підтримується зазначеним типом мікроконтролера, буде видано повідомлення. Також повідомлення з'явиться у випадку, якщо розмір програми перевищить обсяг наявної у пристрою пам'яті.

Синтаксис: .DEVICE AT90S1200 |AT90S2313 | AT90S2323 | AT90S2333 | AT90S2343 | AT90S4414 | AT90S4433 | AT90S4434 | AT90S8515 | AT90S8534 | AT90S8535 | ATtiny11 | ATtiny12 | ATtiny22 | ATmega603 | ATmega103

Приклад:

```
.DEVICE AT90S8535  ; використовувати AT90S8535
.CSEG
.ORG 0000
jmp label1; При асемблюванні з'явиться повідомлення, що ;AT90S8535 не підтримує команду jmp
```

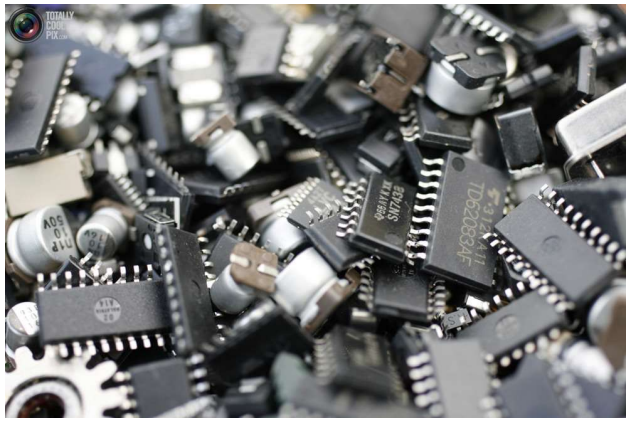
Деякі особливості програмування. Пам'ять даних майже повністю доступна програмі користувача і більшість команд асемблера призначено, щоб обмінюватись даними з нею. Команди пересилання даних надають можливість безпосередньої та непрямої адресації осередків СОЗП, безпосередньої адресації регістрів вводу/виводу та регістрів загального призначення. Оскільки кожному регістру зіставлено осередок пам'яті, то звертатися до них можна як командами адресації регістрів, так і командами адресації осередків СОЗП.

Наприклад, команда:

```
MOV R10,R15 ; скопіювати регістр R15 в регістр R10
```

робить абсолютно те саме, що й команда:

```
LDS R10,0015 ; завантажити в регістр R10 вміст осередку з адресою 0015
```



```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2


C000                    ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START    LDS    #STACK

                ****************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

0013            RESETA  EQU    %00010011
0011            CTLREG  EQU    %00010001

C003 86 13      INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04           STA A  ACIA
C008 86 11              LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A  ACIA

C00D 7E C0 F1           JMP    SIGNON    GO TO START OF MONITO

                ****************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04   INCH    LDA A  ACIA      GET STATUS
C013 47                 ASR A            SHIFT RDRF FLAG INTO
C014 24 FA              BCC    INCH      RECIEVE NOT READY
C016 B6 80 05           LDA A  ACIA+1    GET CHAR
C019 84 7F              AND A  #$7F      MASK PARITY
C01B 7E C0 79           JMP    OUTCH     ECHO & RTS

                ****************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

C01E 8D F0      INHEX   BSR    INCH      GET A CHAR
C020 81 30              CMP A  #'0       ZERO
C022 2B 11              BMI    HEXERR    NOT HEX
C024 81 39              CMP A  #'9       NINE
C026 2F 0A              BLE    HEXRTS    GOOD HEX
C028 81 41              CMP A  #'A
C02A 2B 09              BMI    HEXERR    NOT HEX
C02C 81 46              CMP A  #'F
C02E 2E 05              BGT    HEXERR
C030 80 07              SUB A  #7        FIX A-F
C032 84 0F      HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
C034 39                 RTS

C035 7E C0 AF   HEXERR  JMP    CTRL      RETURN TO CONTROL LOOP
```

Лекцію закінчено
Дякую за увагу