2. Mikroprocesor

01 Co je to mikroprocesor a jeho obecný popis

02 Rozdíl mezi Mikroprocesorem a Mikrořadičem

03 Charakteristika procesoru ARM STM32F4

- **a)** Architektura
- **b)** Velikost a typ paměti
- **c)** Registry

04 Popis:

- **a)** Portů
- **b)** Čítačů/časovačů, včetně realizace čas. smyček
- **c)** Přerušovacího systému

05 Charakteristika školního kitu + periferie

06 Konfigurace projektu a práce v prostředí Keil v.5

2. Mikroprocesor

Mikroprocesor a jeho popis

Mikroprocesor lze definovat jako **sekvenční automat**, což znamená, že pracuje podle **předem daného sledu operací na základě vstupních instrukcí**. Je vyroben technologií **VLSI** *(Very Large Scale Integration)*, což znamená, že **obsahuje miliony až miliardy tranzistorů** na jednom integrovaném obvodu *(čipu)*. Díky tomu je schopný vykonávat **složité výpočetní operace.**

Struktura mikroprocesoru

Řadič (Control Unit - CU)

- **Řídí chod mikroprocesoru**, synchronizuje jednotlivé operace.
- Dekóduje instrukce a určuje, jaké operace mají být **provedeny**.
- Ovládá činnost **aritmeticko-logické jednotky** *(ALU)* na základě zadaných instrukcí.
- **Koordinuje tok dat** mezi jednotlivými částmi mikroprocesoru.

Aritmeticko-logická jednotka (ALU - Arithmetic Logic Unit)

- Provádí základní **matematické** *(sčítání, odčítání, násobení, dělení)* a **logické operace** *(AND, OR, XOR, NOT)*.
- **Výpočetní jádro mikroprocesoru** – bez ní by procesor nebyl schopen vykonávat žádné výpočty.
- **Výsledky operací ukládá do registrů nebo paměti**.

Princip fungování mikroprocesoru

- Mikroprocesor **zpracovává instrukce sekvenčně** – provádí je **jednu po druhé**, podle **pořadí v paměti** programu.
- Každá instrukce se skládá ze tří základních fází:
  - **Načtení instrukce** *(Fetch)* – Instrukce je načtena z operační paměti do registru instrukcí.
  - **Dekódování instrukce** *(Decode)* – Řadič analyzuje instrukci a rozhodne, jak ji vykonat.
  - **Vykonání instrukce** *(Execute)* – ALU provede výpočet nebo jinou operaci podle instrukce.

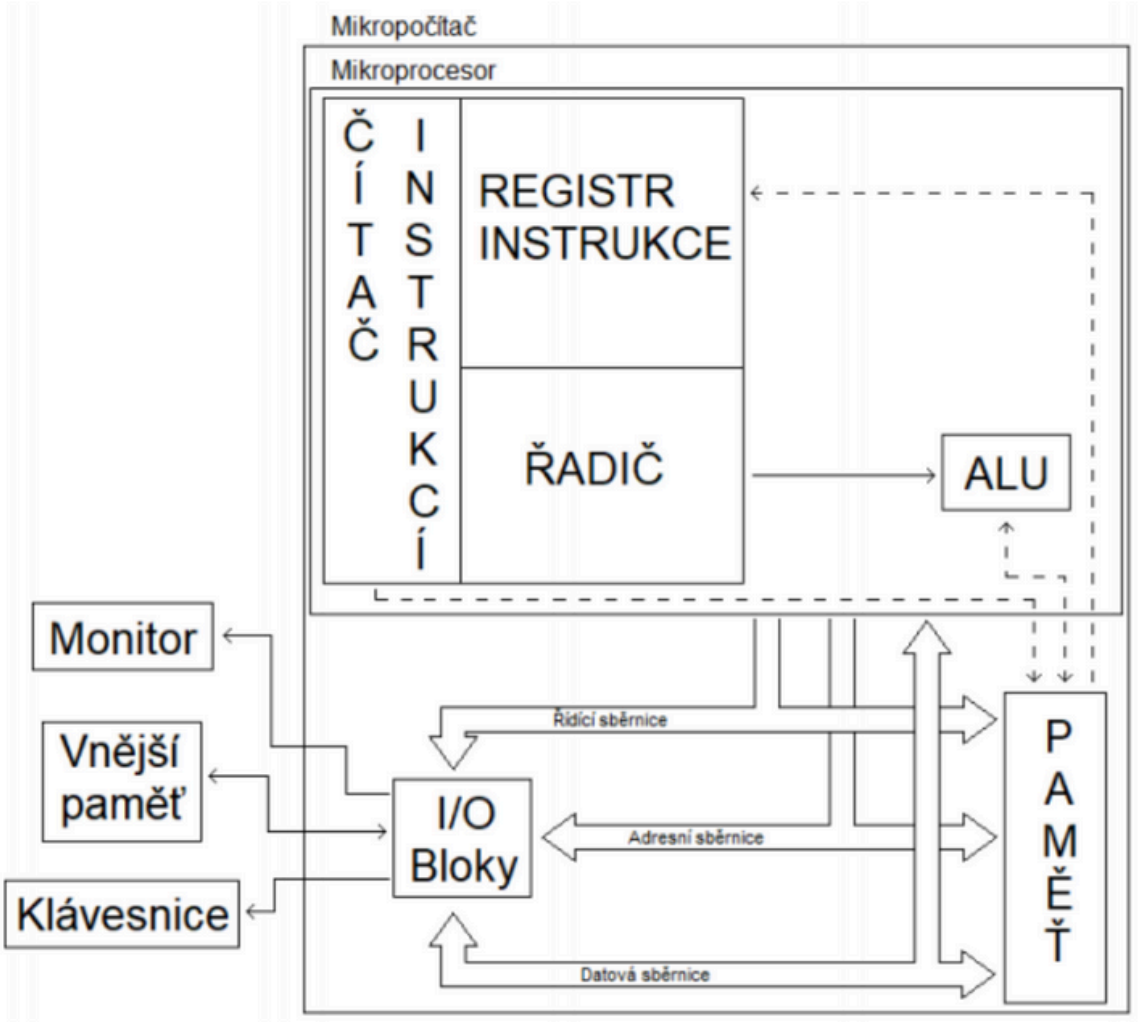
Doplňkové obvody a funkčnost

- Mikroprocesor **není schopen provozu bez podpůrných obvodů**, které zajišťují **komunikaci, vstup, výstup a uložení dat**.
- Je **integrován do pouzdra jediného integrovaného obvodu** *(IC - Integrated Circuit)*.
- Aby mohl pracovat efektivně, **potřebuje přístup k dalším součástem**:
  - **Paměť** *(RAM, ROM)* – slouží k **ukládání instrukcí** a dat.
  - **I/O bloky** *(Input/Output)* – zajišťují **komunikaci s periferiemi** *(klávesnice, monitor, tiskárna, atd.)*.
  - **Řadiče sběrnic** – propojují mikroprocesor s pamětí a vstupně-výstupními zařízeními přes **adresní, datovou a řídicí sběrnici.**

Rozšíření mikroprocesoru na mikropočítač

- **Mikropočítač** = *mikroprocesor + paměť + vstupně-výstupní obvody*
- Mikroprocesor je základní součástí mikropočítače, ale **sám o sobě nemůže fungovat**.
  - **Paměť** umožňuje **ukládání a čtení dat**, **I/O bloky** umožňují **komunikaci s uživatelem** a jinými zařízeními, **řídicí obvody** umožňují **synchronizaci a správu datových toků.**
- Mikropočítačový systém je **rozšířen o periferie** *(monitor, klávesnice, pevný disk, síťová karta)*, což umožňuje **interakci s uživatelem.**

2. Mikroprocesor



Řadič (Control Unit - CU)

Řadič je elektronická řídicí jednotka, která **řídí činnost všech částí procesoru i celého počítače**. Je zodpovědný za **správné vykonávání instrukcí a synchronizaci jednotlivých komponent** systému.

Vlastnosti řadiče

- **Realizace sekvenčním obvodem** – pracuje na principu sekvenční logiky, což znamená, že jeho výstupní stav závisí na předchozím stavu a aktuálních vstupech.
- **Řídicí signály** – generuje signály, které ovládají ostatní části PC *(paměť, ALU, vstupní/výstupní zařízení atd.)*.
- **Reakce na stavová hlášení** – moduly v PC reagují na signály řadiče a posílají zpět stavová hlášení, která obsahují informace o aktuálním stavu operace.
- **Hierarchické řízení** – hlavní řadič procesoru *(součást CPU)* může řídit další řadiče v systému, např. řadič operační paměti nebo řadič vstupně/výstupních operací.

Příklad činnosti řadiče:

1. **Načte instrukci** z paměti *(fetch)*.
2. **Dekóduje** instrukci *(decode)*.
3. **Vygeneruje** odpovídající řídicí **signály**.
4. **Pošle signály** do ALU, paměti nebo I/O zařízení.
5. Po provedení operace **přijme stavové hlášení**.

2. Mikroprocesor

Aritmeticko-logická jednotka (ALU - Arithmetic Logic Unit)

ALU je **základní komponenta procesoru**, která provádí **všechny aritmetické a logické operace**. Bez ní by procesor nemohl vykonávat žádné výpočty.

Typické operace ALU

- **Aritmetické operace:** sčítání, odčítání, násobení, dělení.
- **Logické operace:** AND, OR, XOR, NOT.
- **Porovnávání hodnot:** větší než, menší než, rovnost.
- **Bitové operace:** posuny bitů *(shift, rotate)*.

Struktura a paralelní výpočty

- Moderní procesory mají **více ALU jednotek na jednom čipu.**
- **Rozdělení ALU podle typu operací:**
  - ALU pro **celočíselné operace** *(Integer ALU)*.
  - ALU pro operace s **plovoucí desetinnou čárkou** *(Floating-Point ALU - FPU)*.
- **Paralelní výpočty:** více ALU umožňuje vykonávat **více instrukcí současně** *(tzv. superskalární architektura)*.
- ALU pracuje s **fixním rozsahem operandů** podle **šířky registrů procesoru** *(např. 32bit, 64bit)*.
- Výpočty s vyšší přesností se provádějí **softwarově pomocí knihoven** *(např. BigInt v programování)*.

Druhy mikroprocesorů

1. CPU (Central Processing Unit)

- Hlavní procesor počítače, který vykonává **obecné výpočty a řídí běh operačního systému**.
- Obsahuje **řadič, ALU, registry a sběrnice**.
- Vykonává **instrukce zadané programem**.

2. GPU (Graphics Processing Unit)

- **Grafický procesor**, specializovaný na **zpracování grafiky**.
- Má **velké množství paralelních ALU jednotek**, což umožňuje rychlé vykonávání výpočtů potřebných pro vykreslování obrazu.
- Používá se také pro **výpočty v AI, kryptografii a vědeckých simulacích** *(GPGPU - General-Purpose computing on GPU)*.

3. APU (Accelerated Processing Unit)

- Kombinace **CPU a GPU v jednom čipu**.
- Umožňuje **sdílení paměti mezi CPU a GPU,** čímž **zvyšuje efektivitu**.
- Používá se v noteboocích a konzolích *(např. AMD Ryzen s integrovanou grafikou)*.

4. FPU (Floating-Point Unit)

- Matematický koprocesor, který **urychluje výpočty s desetinnými čísly.**
- Dříve byl oddělený od hlavního CPU, **dnes je běžně integrovaný přímo v procesoru**.

5. Zvukový procesor

- Specializovaný procesor **pro zpracování zvuku.**
- Používá se v **zvukových kartách**, **konzolích** nebo profesionálních **audio zařízeních**.
- Může **urychlovat efekty, mixování a dekódování zvukových formátů** *(např. Dolby Atmos, DTS)*.

2. Mikroprocesor

Von Neumannova architektura

Von Neumannova architektura je **základním konceptem počítačové architektury**, na které jsou **postaveny moderní počítače.**

- Vynalezl ji **John von Neumann v roce 1945**, aby **překonal omezení starších mechanických** a **elektromechanických počítačů**.
- **Hlavní myšlenka:** program a data jsou **uloženy společně** v jedné paměti a jsou **zpracovávány sekvenčně** podle posloupnosti instrukcí.
- Dnes se používá s **určitými úpravami** *(např. multitasking, víceprocesorové systémy)*.

Princip fungování

Von Neumannova architektura pracuje na principu **sekvenčního zpracování instrukcí** *(na rozdíl od Harvardské architektury, kde jsou data a instrukce odděleny)*.

Tento princip znamená, že:

1. Počítač **načte instrukci** z paměti.
2. **Dekóduje** ji a provede ji podle řídicích signálů.
3. **Pokračuje na další instrukci** uloženou v paměti.

Díky tomuto přístupu **není nutné ručně zadávat celý program do operační paměti** *(OP)* před spuštěním, což umožňuje **snadnější změnu programového kódu a vývoj softwaru**.

Nezávislost na problému

- Struktura počítače **není pevně vázána na konkrétní problém**.
- Stejný počítač může **vykonávat různé úlohy** pouhou změnou programu.

Paměť a instrukční formát

- **Instrukce i data mají stejný formát** a jsou **uloženy ve stejné paměti**.
- Data jsou **rozdělena do slov nebo slabik pevné velikosti**.
- Paměť je **adresovatelná** – přístup k datům a instrukcím je **prováděn pomocí adres.**
  - **Nevýhoda tohoto přístupu:**
    - Společné uložení programu a dat může **způsobit přepsání programového kódu**, pokud dojde k chybě *(např. chybné adresování nebo přetečení paměti)*.

Struktura paměti

- Paměť je rozdělena **na buňky stejné velikosti**, přičemž **každá buňka má vlastní adresu**.
- Procesor **přistupuje k paměti pomocí adresní sběrnice**.

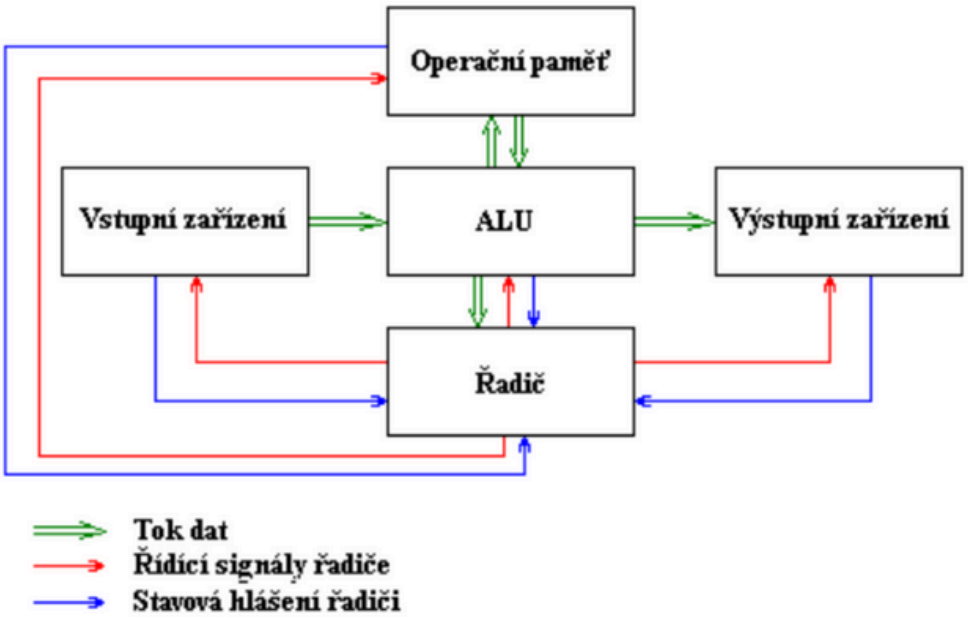
Systém je rozdělen do pěti základních částí:

1. **Řadič** *(Control Unit - CU)*
2. **ALU** *(Arithmetic Logic Unit)*
3. **Paměť** *(Memory)*
4. **Vstupní zařízení** *(Input Devices)*
5. **Výstupní zařízení** *(Output Devices)*

V systému existují tři hlavní typy signálů, jak je znázorněno na obrázku:

1. **Tok dat** *(zelená barva)*
   - Přenos dat mezi pamětí, ALU a vstupně/výstupními zařízeními.
2. **Řídicí signály řadiče** *(červená barva)*
   - Řadič posílá signály k ovládání jednotlivých komponent.
3. **Stavová hlášení řadiči** *(modrá barva)*
   - Jednotlivé části systému posílají zpětnou vazbu řadiči, aby mohl koordinovat další operace

2. Mikroprocesor



Harvardská architektura

Harvardská architektura je počítačová architektura, **která fyzicky odděluje paměť programu a paměť dat.** To znamená, že:

- Instrukce a data jsou **uloženy ve dvou oddělených paměťových blocích**.
- Každá paměť má **svou vlastní sběrnici** *(adresní a datovou)*, což umožňuje **paralelní přístup k instrukcím i datům.**

Výhody oproti Von Neumannově architektuře

- Harvardská architektura vznikla jako **řešení problémů Von Neumannovy architektury**, především:
  - **Žádné riziko přepsání programu** – oddělená paměť eliminuje možnost, že by se program přepsal sám kvůli chybě v kódu.
  - **Rychlejší zpracování instrukcí** – díky dvěma sběrnicím může procesor načítat instrukce a data současně, což zvyšuje výkon *(v Von Neumannově architektuře probíhá přístup sekvenčně).*
  - **Optimalizace paměti** – není nutné, aby paměť programu a paměť dat měly stejnou velikost a parametry, což umožňuje efektivnější využití zdrojů.

Nevýhody a omezení

- **Vyšší složitost řídicí jednotky** – nutnost řídit dvě oddělené paměťové sběrnice klade vyšší nároky na návrh procesoru.
- **Nelze sdílet paměť mezi daty a instrukcemi** – nevyužitá část paměti pro instrukce nelze využít pro data a naopak, což může vést k méně efektivnímu využití paměti.
- **Omezené využití ve všeobecných výpočetních systémech** – Harvardská architektura se používá především v speciálních výpočetních systémech *(např. vestavěné systémy, digitální signálové procesory - DSP).*

Struktura a součásti Harvardské architektury

- **Řadič** *(Control Unit - CU)*
- **ALU** *(Arithmetic Logic Unit)*
- **Paměť programu** *(Instruction Memory)*
- **Paměť dat** *(Data Memory)*
- **Vstupní a výstupní zařízení** *(I/O)*

2. Mikroprocesor

Paralelní zpracování a rychlost

- Von Neumannova architektura zpracovává instrukce sekvenčně, což způsobuje tzv. **Von Neumannovu bariéru** *(Neumann Bottleneck)* – procesor musí čekat, než se instrukce načte.
- Harvardská architektura **umožňuje současné načítání instrukcí a dat**, což výrazně **zvyšuje výkon a propustnost systému.**

Příklad rozdílu v rychlosti:

- **Von Neumannova architektura:** Procesor **načte instrukci → dekóduje → provede → načte data → zpracuje → opakuje.**
- **Harvardská architektura:** Procesor **současně načítá instrukci a data**, což umožňuje **vykonání více operací** během jednoho cyklu.

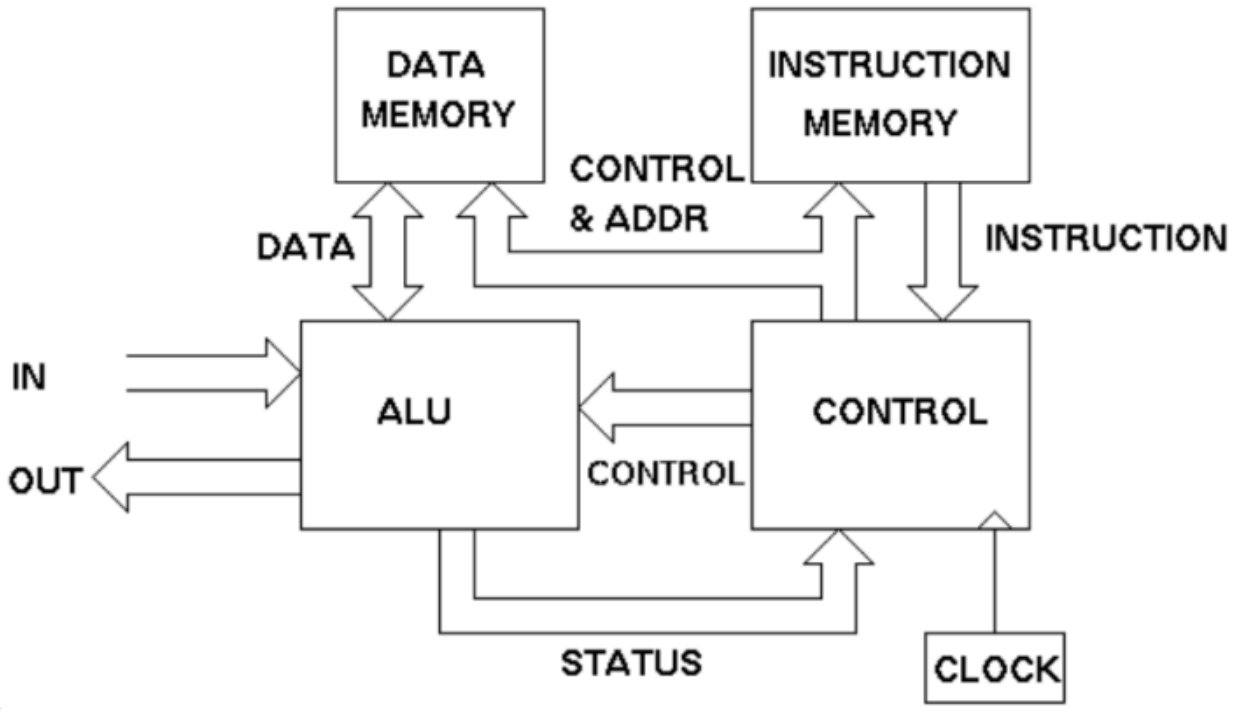
Použití Harvardské architektury

- **Vestavěné systémy** – např. mikrokontroléry **Atmel 8051, ARM Cortex**, digitální signálové procesory *(DSP)*.
- **Speciální výpočetní systémy** – například **grafické** procesory *(GPU)*, **audio** procesory, **síťové** čipy.
- **Vysoce výkonné počítače** – některé **moderní superpočítače** využívají prvky Harvardské architektury.

Kombinace obou architektur:

Dnešní procesory často využívají **modifikovanou Harvardskou architekturu**, kde:

- Paměť programu a dat jsou odděleny v **cache**, ale **propojeny v hlavní paměti**.
- **CPU** může **přistupovat k oběma pamětem současně**, ale zpracování je flexibilnější než u čisté Harvardské architektury.



2. Mikroprocesor

Faktor	Von Neumannova architektura	Harvardská architektura
Uložení programu a dat	Společná paměť	Oddělené paměti
Paměťová sběrnice	Jedna sběrnice pro instrukce i data	Dvě oddělené sběrnice
Přístup k datům a instrukcím	Sekvenční přístup	Paralelní přístup
Riziko přepsání programu	Ano	Ne
Efektivita využití paměti	Lepší (paměť může být využita univerzálně)	Horší (instrukční a datová paměť jsou fixní)
Náročnost na návrh CPU	Nižší	Vyšší
Využití	Obecné počítače, operační systémy	Vestavěné systémy, DSP, GPU

Architektura CISC vs. RISC

Architektura CISC (Complex Instruction Set Computer)

CISC procesory používají **složitou instrukční sadu**, která obsahuje **velké množství komplexních instrukcí**. Tyto instrukce mohou **provádět složité operace**, které by v jednodušší architektuře vyžadovaly více kroků.

- **Velká sada instrukcí** – podporuje široké spektrum operací, což usnadňuje programování.
- **Instrukce mají proměnlivou délku** – každá instrukce může zabírat jiný počet bajtů v paměti.
- **Různá doba vykonávání instrukcí** – některé instrukce trvají několik cyklů, zatímco jiné jsou rychlejší.
- **Mikroprogramové řízení** – instrukce jsou dekódovány a vykonávány mikroprogramem uloženým v řadiči procesoru.
- **Vyšší složitost CPU** – složité dekódování instrukcí vyžaduje větší množství tranzistorů a složitější řadič.
- **Pomalejší oproti RISC** – složité instrukce mohou trvat více cyklů hodin.

Typické procesory s CISC architekturou:

- Intel x86 *(Pentium, Core i7, AMD Ryzen)*
- Motorola 68000 *(používaný v historických Apple Macintosh, Amiga)*

Příklad: Jedna **složitá instrukce** může například **nahradit celou posloupnost operací**, jako je načtení hodnoty z paměti, její úprava a uložení zpět.

2. Mikroprocesor

Architektura RISC (Reduced Instruction Set Computer)

RISC procesory využívají redukovanou instrukční sadu, která obsahuje jednodušší a kratší instrukce. Každá instrukce obvykle odpovídá jedné operaci, což umožňuje efektivnější a rychlejší vykonávání.

- **Menší a optimalizovaná sada instrukcí** – každá instrukce provádí jednoduchou operaci (např. přesun, sčítání).
- **Všechny instrukce mají stejnou délku** – obvykle 32 bitů, což zjednodušuje dekódování a provádění instrukcí.
- **Rychlejší než CISC** – zpracování instrukcí trvá jediný hodinový cyklus.
- Obvodově řízené CPU – místo mikroprogramu se instrukce vykonávají přímo pomocí logických obvodů.
- **Možnost zvyšování frekvence CPU** – jednodušší struktura procesoru umožňuje vyšší taktovací frekvence.
- **Zřetězené zpracování instrukcí** *(pipelining)* – zatímco se jedna instrukce provádí, další se už načítá a dekóduje.

Typické procesory s RISC architekturou:

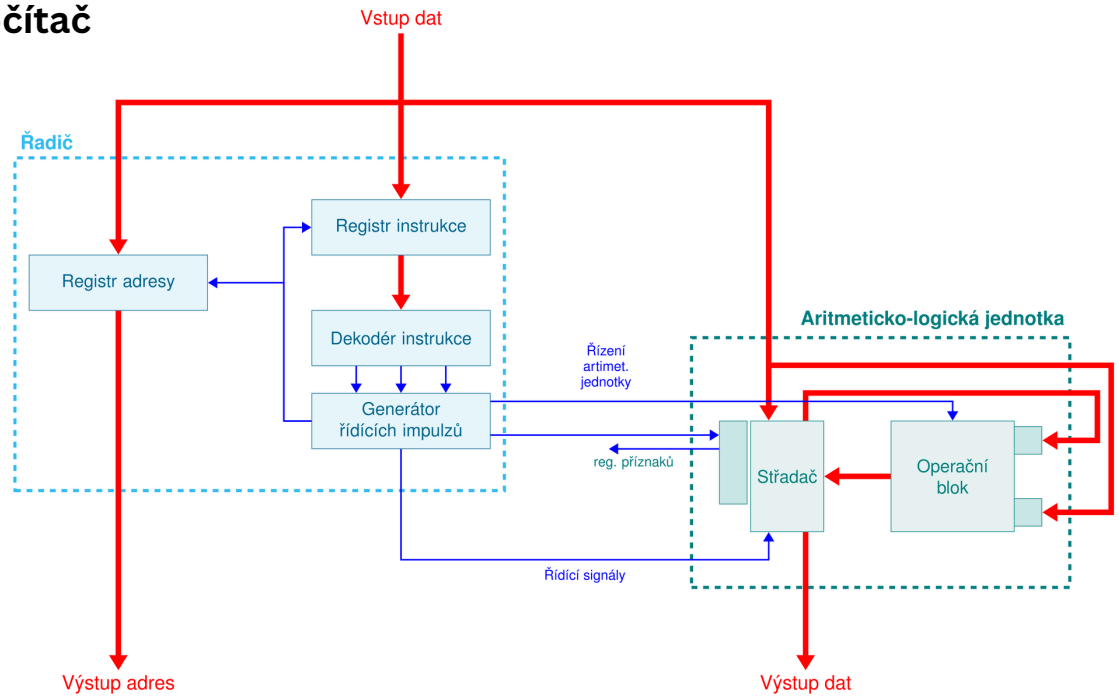
- **ARM** *(používané v mobilních telefonech, např. Apple A15, Qualcomm Snapdragon)*
- **Atmel AVR** *(např. mikrokontroléry Atmel 8051)*
- Moderní **Intel procesory** využívají **RISC-like prvky uvnitř jádra**, přestože zachovávají CISC kompatibilitu.

Příklad: Namísto jedné složité instrukce jako v CISC musí **RISC procesor provést několik jednoduchých instrukcí**, ale každá instrukce **se vykonává extrémně rychle**.

Faktor	CISC	RISC
Instrukční sada	Velká, složitá	Malá, jednoduchá
Délka instrukcí	Proměnlivá	Stejná délka (např. 32 bitů)
Doba vykonávání instrukcí	Některé trvají více cyklů	Každá instrukce obvykle 1 cyklus
Řízení CPU	Mikroprogramově řízeno	Obvodově řízeno
Pipelining (zřetězení instrukcí)	Náročné na implementaci	Snadná implementace
Efektivita spotřeby energie	Vyšší spotřeba	Nižší spotřeba
Použití	PC, servery, kompatibilní s historickými programy	Mobilní zařízení, superpočítače

2. Mikroprocesor

Mikropočítač



1. Řadič (Control Unit) – levá část obrázku

a) Registr adresy

- Uchovává adresu instrukce, která se má právě vykonat.
- Posílá ji na adresní sběrnici, kde podle této adresy procesor načítá instrukci z paměti.

b) Registr instrukce

- Načtená instrukce se uloží právě sem – tedy obsah paměti z určené adresy jde "přes vstup dat" sem.
- Obsahuje kód instrukce, např. „sečti“, „ulož“, „skoč“, apod.

c) Dekodér instrukcí

- Převádí načtenou instrukci z registru instrukce na strojové mikroinstrukce.
- Např. zjistí: „Aha, tohle je sčítání, tak musím spustit ALU, připravit registry…“

d) Generátor řídicích impulzů

- Generuje řídicí signály na základě výstupu z dekodéru (např. „povol čtení z paměti“, „aktivuj ALU“, „ulož do registru“).
- Tyto signály synchronizují celý procesor, aby dělal věci ve správný čas.

2. Aritmeticko-logická jednotka (ALU) – pravá část obrázku

e) Operační blok

- Zde se vykonává konkrétní instrukce (např. sčítání, bitový posun, logické AND).
- Vstupní data sem přijdou ze střadače a také z vstupu dat.

f) Střadač (akumulátor)

- Uchovává mezi-výsledky výpočtů – tzn. jeden operand může být načtený, druhý se zpracuje, výsledek se zase uloží sem.
- Výsledek z ALU se ukládá zpět do střadače a odtud jde ven přes „výstup dat“ nebo zpět do dalšího cyklu.

g) Registr příznaků (PSV)

- Sleduje stav po výpočtu:
  - Přetečení (overflow), Záporný / kladný výsledek, Parita (sudý / lichý počet jedniček), Nulový výsledek
- Tyto příznaky používá např. řadič pro skokové instrukce (např. "skoč, pokud výsledek byl 0").

3. Toky a barvy

- 🔴 Červené šipky = tok dat (vstupy, výstupy, instrukce, výsledky)
- 🔵 Modré šipky = řídicí signály a vnitřní tok řízení
- 🟢 Zeleně ohraničená část = vnitřní logika řadiče

Shrnutí logiky:

1. Registr adresy vyšle adresu instrukce.
2. Z paměti přijde instrukce přes vstup dat do registru instrukce.
3. Dekodér ji převede na řídicí kód, generátor impulsů vyšle příkazy dál.
4. Střadač přivede operandy do operačního bloku, ten vykoná výpočet.
5. Výsledek se uloží zpět do střadače a případně odešle ven.
6. PSV sleduje, jestli např. nebyl přetečen výpočet.

2. Mikroprocesor

Hybridní přístup – kombinace CISC a RISC

Moderní procesory **kombinují prvky obou architektur:**

- **Intel x86** procesory mají **CISC instrukční sadu**, ale interně je **dekódují na jednodušší RISC instrukce**, které se rychleji vykonávají.
- **ARM procesory** jsou čistě **RISC**, což jim umožňuje dosahovat vysokého výkonu s nízkou spotřebou energie.

Kde se co používá?

- **CISC** *(Intel x86, AMD Ryzen)* – klasické **osobní počítače a servery**, kde je klíčová kompatibilita s historickým softwarem.
- **RISC** *(ARM, Apple M1/M2, Qualcomm Snapdragon)* – **mobilní telefony, tablety, superpočítače, mikrokontroléry**, kde je důležitá rychlost a energetická efektivita.

Charakteristika procesoru ARM STM32F4

STM32F4 je **vysoce výkonný 32bitový mikrořadič** založený na architektuře **ARM Cortex-M4**, která je určena pro **embedded systémy**. Tento procesor je **oblíbený díky vysokému výkonu**, **nízké spotřebě energie** a **široké podpoře** vývojových nástrojů.

Procesorové jádro

- **Architektura:** ARM Cortex-M4
- **Typ jádra:** 32bitový RISC procesor
- **Taktovací frekvence:** až 168 MHz, což umožňuje rychlé zpracování instrukcí a efektivní řízení real-time aplikací.
- **Podpora DSP instrukcí** – zabudovaná digitální signálová procesorová jednotka (DSP) umožňuje efektivní práci s matematickými operacemi, jako jsou FFT, filtrace signálů a složité výpočty.
- **Jednotka ochrany paměti** *(MPU – Memory Protection Unit)* – zajišťuje bezpečné oddělení paměťových sekcí, což zvyšuje bezpečnost a stabilitu běhu programu.

Paměť

- **Flash paměť:** 1 MB – slouží k uchovávání programu i některých dat.
- **RAM paměť:** 192 kB – rychlá operační paměť pro běh programu.
- Teoretická možnost rozšíření **RAM až na 4 GB** *(s externí pamětí a správným návrhem)*.
- **Paměťová architektura:** Podporuje Load/Store architekturu *(viz níže)*.

Napájení a spotřeba

- **Napájecí napětí: 2V – 3,6V**, což umožňuje provoz v **nízkonapěťových systémech**, např. v bateriově napájených zařízeních.
- **Nízká spotřeba energie** – Cortex-M4 jádro obsahuje **optimalizované řízení napájení**, což umožňuje úsporné režimy *(low-power mode)*.

Load/Store architektura

- STM32F4 procesory používají **Load/Store architekturu**, což znamená, že:
  - S pamětí pracují **pouze dvě instrukce**:
    - **LOAD** – načte data z paměti do registru.
    - **STORE** – uloží data z registru zpět do paměti.
- Všechny ostatní instrukce pracují **výhradně s interními registry**, což zvyšuje rychlost výpočtů, protože **přístup do registrů je mnohem rychlejší než přístup do RAM**.
  - **Rozdělení instrukcí na dvě kategorie:** Práce s pamětí – Load/Store operace.
  - **Práce s registry** – matematické a logické operace probíhají pouze v vnitřních registrech.

2. Mikroprocesor

Vývoj a programování

- **Programovací jazyk:** ARM STM32 podporuje jazyk **C a C++,** což umožňuje jednoduché a efektivní programování.
- **Vývojové prostředí:**
  - STM32CubeIDE *(STMicroelectronics)*.
  - **Keil uVision *(ARM). <- použití na SPŠE***
  - IAR Embedded Workbench.
  - Open-source podpora *(PlatformIO, GCC ARM toolchain)*.
- **Podpora ladění a emulace** – procesory obsahují hardwarovou podporu pro debugging *(SWD – Serial Wire Debug)*.

Použití STM32F4

Díky výkonnému jádru, nízké spotřebě a široké podpoře periferií se **STM32F4 využívá v různých oblastech:**

- **Vestavěné systémy** *(embedded systémy)* – řízení průmyslových strojů, chytré senzory.
- **Robotika** – řízení motorů, snímání senzorů.
- **Automatizace** – IoT zařízení, inteligentní domácnosti.
- **Digitální zpracování signálů** *(DSP)* – zvuková analýza, filtrování signálů.
- **Lékařská zařízení** – snímání biologických dat, přenosové systémy.

Sběrnice a registry v STM32F4

Procesory STM32F4 využívají moderní sběrnicovou architekturu, která umožňuje efektivní **komunikaci mezi CPU, pamětí a periferiemi**. Hlavními sběrnicemi v této architektuře jsou **AHB** *(Advanced High-Performance Bus)* a **APB** *(Advanced Peripheral Bus)*.

1. **Sběrnice:**

AHB – Advanced High-Performance Bus

AHB je **vysokorychlostní sběrnice**, která **propojuje hlavní komponenty procesoru a umožňuje rychlou výměnu dat** mezi klíčovými částmi systému.

- Propojuje **CPU, RAM a DMA** *(Direct Memory Access)*.
- **Podpora dávkového přenosu dat** *(Burst Mode)* – umožňuje rychlé přesuny dat mezi pamětí a periferiemi.
- Slouží jako **most pro komunikaci** s **APB sběrnicí**.
- **Vysoká propustnost** – využívá širokou adresovou sběrnici a velkou šířku dat.

Použití AHB:

- **Přímý přístup** CPU k RAM a Flash paměti.
- Přenosy **řízené DMA** *(např. pro zpracování signálů bez zatížení CPU)*.
- Propojení s dalšími vysokorychlostními sběrnicemi.

2. Mikroprocesor

APB – Advanced Peripheral Bus

APB je sběrnice pro **pomalejší periferní zařízení,** která nevyžadují vysokou propustnost, ale **musí být energeticky úsporná**.

- Jednoduchý interface pro připojení periferií.
- Nižší spotřeba energie než AHB.
- Komunikace přes most mezi AHB a APB sběrnicí.
- Podporuje připojení pomalejších periferií jako:
  - **Časovače** *(TIM)*
  - **UART** *(Universal Asynchronous Receiver-Transmitter)*
  - **I2C** *(Inter-Integrated Circuit)*
  - **SPI** *(Serial Peripheral Interface)*

Použití APB:

- Připojení **nízkorychlostních periferií** k **procesoru**.
- **Ovládání senzorů, displejů a komunikačních rozhraní.**
- **Energeticky efektivní** komunikace.

Most mezi AHB a APB umožňuje propojení těchto dvou sběrnic a zajišťuje, že data **mezi rychlou a pomalejší sběrnicí mohou plynule proudit.**

2.RCC – Reset and Clock Control:

RCC (Reset and Clock Control) se stará o:

- **Správu hodinového signálu** pro různé části procesoru.
- **Aktivaci nebo deaktivaci periferií** podle potřeby *(šetření energie)*.

AHB1ENR – Aktivace periferií

- **AHB1ENR** *(AHB1 Enable Register)* je **konfigurační registr**, který umožňuje aktivaci různých **periferií na AHB1 sběrnici.**
- Každý **bit v registru odpovídá jedné periferii** – pokud je bit nastaven na **1**, daná periferie je **aktivní**.

Použití RCC -> AHB1ENR:

- **Aktivace portů GPIO** pro práci s tlačítky nebo LED diodami.
- Zapnutí **paměťových řadičů** nebo komunikačních rozhraní.
- Snížení spotřeby energie vypnutím nevyužitých periferií.

3. **Registry GPIO**

V STM32F4 jsou **GPIO** *(General-Purpose Input/Output)* piny **ovládány pomocí specifických registrů.** Tyto registry umožňují konfiguraci vstupů a výstupů, čtení tlačítek nebo ovládání LED diod.

MODER – Konfigurace režimu pinu

- **GPIOx_MODER** *(Mode Register)* definuje, zda je pin:
  - Vstupní *(Input)*
  - Výstupní *(Output)*
  - Analogový *(Analog Mode)*
  - Alternativní funkce *(AF – Alternate Function)*
    - **GPIOA->MODER |= (1 << 10);** *// Nastaví PA5 jako výstup*

2. Mikroprocesor

IDR – Čtení stavu vstupu (tlačítka)

- **GPIOx_IDR** *(Input Data Register)* slouží ke **čtení hodnoty vstupních pinů.**
  - Pokud je hodnota **1**, znamená to, že na pinu **je logická 1** *(napětí)*.
  - Pokud je hodnota **0**, je na pinu **logická 0** *(uzemnění)*.
  - *Příklad čtení tlačítka na pinu PA0:*
    - **if (GPIOA->IDR & (1 << 0))** *{*
      // Tlačítko je stisknuté
      }

ODR – Nastavení výstupu (LED)

- **GPIOx_ODR** *(Output Data Register)* slouží k **nastavení výstupní hodnoty pinu**.
  - Pokud zapíšeme **1** do registru, výstupní pin se **nastaví na logickou 1** *(LED svítí)*.
  - Pokud zapíšeme **0**, výstupní pin se **nastaví na logickou 0** *(LED zhasne)*.
  - Příklad rozsvícení **LED na PA5:**
    - **GPIOA->ODR |= (1 << 5);** // Zapnutí LED
  - Příklad zhasnutí LED na PA5:
    - **GPIOA->ODR &= ~(1 << 5);** // Vypnutí LED

Komponenta	Funkce
AHB sběrnice	Rychlá komunikace mezi CPU, RAM a DMA
APB sběrnice	Připojení pomalejších periferií (UART, SPI, I2C, TIM)
RCC – AHB1ENR	Aktivace periferií přes hodinový registr
GPIOx_MODER	Nastavení vstupního/výstupního režimu pinu
GPIOx_IDR	Čtení vstupních dat (např. tlačítka)
GPIOx_ODR	Nastavení výstupu (např. LED diody)

2. Mikroprocesor

Časovač (Timer) v procesorech STM32F4

Časovače *(Timers)* jsou **hardwarové moduly**, které umožňují **přesné měření času**, **generování časových signálů a řízení různých časově závislých operací**. Jsou klíčovou součástí **multitaskingu**, řízení událostí a **real-time** aplikací.

Funkce a princip časovače

- Časovače měří čas **nezávisle na hlavním programovém toku**, což umožňuje efektivní multitasking.
- Použití v **operačním systému** – **OP** využívá časovače k:
  - **Řízení** plánování úloh *(scheduler)*.
  - Časování **přerušení**.
  - **Synchronizaci** procesů.
- Hardwarové časovače jsou přítomny ve většině počítačů a mohou být:
  - Přímo v **CPU**.
  - V **čipsetu** základní desky.
  - Ve specializovaném **integrovaném obvodu** *(např. RTC – Real Time Clock)*.

Princip činnosti čítače (Timer)

1. Procesor nastaví **počáteční hodnotu časovače.**
2. Hodnota se automaticky **snižuje** *(dekrementuje)* s každým **hodinovým cyklem**.
3. Jakmile dosáhne **nuly**, vyvolá se **přerušení** *(IRQ)*, které informuje procesor.
4. **Procesor zpracuje přerušení** a může znovu načíst hodnotu časovače.

Výhoda:

- Procesor **nemusí aktivně čekat**, čímž se **šetří výpočetní výkon** – odstraňuje se tzv. **aktivní čekání.**

System Tick Timer (SysTick)

SysTick je **jednoduchý systémový časovač**, který je součástí jádra **Cortex-M4** a je optimalizován pro **real-time operační systémy** *(RTOS – Real-Time Operating Systems)*.

- **Periodicky odečítá 1** od uživatelem definované hodnoty.
- Při přetečení vyvolá **přerušení**.
- **Využití:** Kontrola sériové komunikace *(UART)*.
- Ovládání **LCD** displeje.
- Blikání **LED** diody.
- **Časování tlačítek** a **klávesnice**.
- **Konfigurace SysTick:**
  - Vše se automaticky nastavuje pomocí funkce **SysTick_Config();.**
  - **Nelze měnit prioritu přerušení** – SysTick má **pevnou** prioritu.
    - **SysTick_Config(SystemCoreClock / 1000);** *// Konfigurace SysTick každých 1 ms*

2. Mikroprocesor

16bitové časovače TIM6 a TIM7

TIM6 a TIM7 jsou **jednoduché 16bitové časovače**, které lze **využít pro základní časování**.

Vlastnosti TIM6 a TIM7:

- **16bitový čítač** s možností **auto-reload**.
- **Prescaler** *(dělička frekvence)* – umožňuje **snížit vstupní hodinový signál** pro přesnější časování.
- Podpora **přerušení při přetečení čítače**.
- Možnost **generování signálu pro DMA** – umožňuje přímý přenos dat bez zásahu CPU.
- Lze využít pro řízení **DAC** *(Digital-to-Analog Converter)*.

Možné režimy činnosti:

- **Periodický** časovač *(pro generování pravidelných přerušení)*.
- Časovač **událostí** *(pro synchronizaci se senzory)*.
- **Řízení PWM** signálu *(např. regulace jasu LED, řízení motorů)*.

Časovač	Vlastnosti	Použití
SysTick	Interní systémový časovač Cortex-M4, pevná priorita	RTOS plánování, měření času, blikání LED, klávesnice
TIM6 & TIM7	16bitové časovače s prescalerem, podpora auto-reload, možnost přerušení	Periodická přerušení, generování PWM, řízení DAC

Přerušovací systém

Přerušovací systém **umožňuje reakci CPU na události generované periferiemi** nebo jinými součástmi systému **bez neustálé kontroly procesorem**. Základní metodou správy přerušení je **Polling**, který je sice **jednoduchý**, ale **méně efektivní** než moderní přerušovací systémy.

Metoda Polling

Polling je **základní způsob detekce událostí v systému**, kdy **CPU pravidelně kontroluje stav** periferií.

Princip fungování Pollingu:

1. CPU se **cyklicky dotazuje** jednotlivých periferií.
2. Zjistí, zda některá periferie **potřebuje obsluhu**.
3. Pokud ano, **provede** odpovídající úlohu.
4. Pokud ne, **přejde k další** periferii a proces opakuje.

Vlastnosti Pollingu:

- Obsahuje **multiplexer řízený CPU**, který umožňuje komunikaci mezi CPU a vstupně-výstupními zařízeními.
- Zatěžuje CPU, protože procesor **neustále testuje stav periferií** i tehdy, když nejsou aktivní.
- **Nevhodné pro real-time systémy**, protože doba reakce závisí na počtu periferií a rychlosti dotazování.

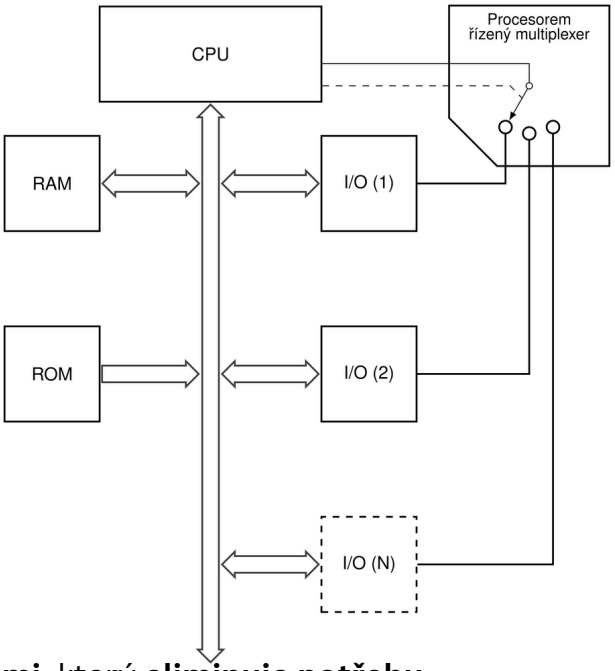
2. Mikroprocesor

Výhody Pollingu:

- **Jednoduchá implementace** – není potřeba složitá správa přerušení.
- **Predikovatelnost** – programátor ví, kdy se bude periferie testovat.

Nevýhody Pollingu:

- Zbytečně **vytěžuje CPU**, i když není žádná událost.
- **Zpoždění při reakci** – čím více periferií, tím delší cyklus dotazování.
- **Nevhodné pro systémy s nízkou spotřebou energie**, protože CPU běží neustále.



Metoda Interrupt (Přerušení)

Je **efektivnější způsob komunikace mezi CPU a periferiemi**, který **eliminuje potřebu neustálého dotazování** *(Polling)*. Přerušení umožňuje **asynchronní reakci procesoru na události**, čímž zvyšuje výkon a efektivitu systému.

Základní fungování přerušení:

1. Periferie vyšle signál přerušení do řadiče přerušení *(PIC – Programmable Interrupt Controller)*.
2. PIC **analyzuje požadavek** a **určí prioritu** přerušení.
3. CPU **přeruší běžící úlohu** a **vykoná přerušovací rutinu** *(ISR – Interrupt Service Routine)*.
4. Po dokončení ISR se CPU vrátí zpět k původní činnosti.

Hlavní rozdíly oproti Pollingu:

- **Není nutné neustálé dotazování periferií** – CPU se aktivuje pouze při události.
- **Zlepšuje výkon** – CPU se může věnovat jiným úlohám, dokud nepřijde přerušení.
- **Rychlejší reakce na události**, vhodné pro real-time aplikace.

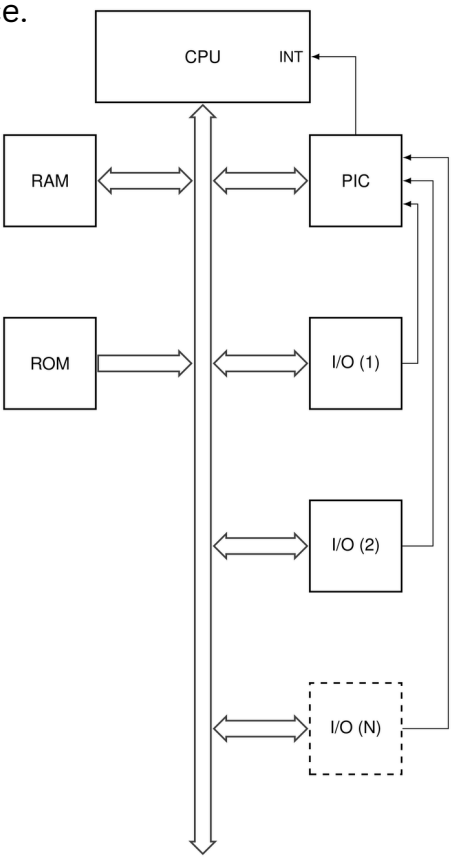
Role PIC *(Programmable Interrupt Controller)*
PIC *(řadič přerušení)* je **hardwarová** součást, která **spravuje přerušení a zjednodušuje** komunikaci mezi CPU a periferiemi.

Funkce PIC:

- **Příjem požadavků** na přerušení od periferií.
- **Prioritizace přerušení** – zpracování důležitějších událostí před méně důležitými.
- Odeslání signálu CPU pro zpracování přerušení.
- Ukončení přerušení a návrat k běžné operaci.

Výhody PIC:

- Zlepšuje organizaci přerušení.
- Snižuje zátěž CPU.
- Podporuje více úrovní priorit přerušení.



2. Mikroprocesor

Činnost přerušení řadiče

Přerušovací systém **umožňuje efektivní řízení událostí** v počítačovém systému, aniž by CPU muselo neustále kontrolovat stav periferií. **Řadič přerušení** *(PIC – Programmable Interrupt Controller)* **zajišťuje správu přerušení a prioritizaci požadavků.**

Postup zpracování přerušení

Periferie vyšle požadavek na přerušení *(IRQ – Interrupt Request)*

- Externí zařízení *(např. klávesnice, časovač)* **vygeneruje signál IRQ**, který signalizuje potřebu obsluhy CPU.
- Žádost může přijít z více zdrojů současně.

Řadič přerušení přijme požadavek

- Pokud je přerušovací systém povolen, řadič přerušení požadavek zaznamená a vyhodnotí prioritu.
- V případě více požadavků vybere řadič přerušení nejdůležitější požadavek.

Řadič přerušení vyšle signál INT procesoru

- Informuje CPU o nutnosti zpracování přerušení.

Procesor pozastaví aktuální činnost a vyšle potvrzení INTA (Interrupt Acknowledge)

- CPU dočasně zastaví vykonávání programu a uvolní sběrnice pro zpracování přerušení.
- Pošle signál INTA zpět řadiči přerušení.

Řadič přerušení určí typ přerušení a odešle ho CPU

- Každé přerušení má přiřazený identifikační kód, který procesor využije k nalezení správného obslužného programu.

Procesor vypočítá adresu obslužné rutiny *(Interrupt Service Routine – ISR)*

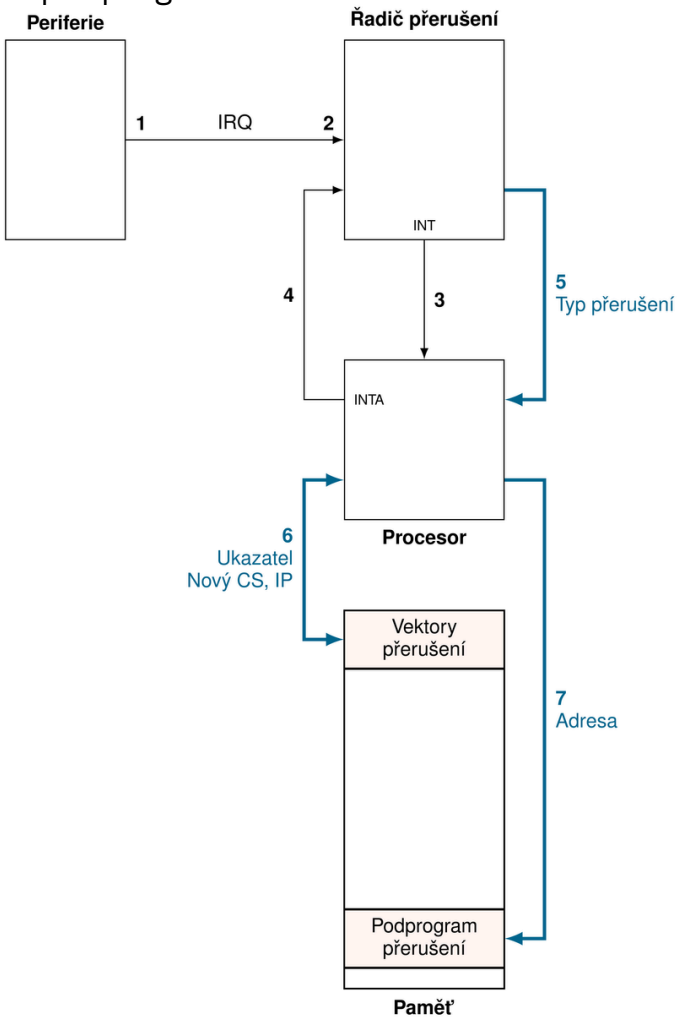
- CPU použije vektor přerušení k nalezení správného podprogramu.
- Zálohuje stav procesoru *(registry, instrukční ukazatel, atd.)*, aby se po obsluze mohl vrátit k původnímu úkolu.

Procesor provede obslužný podprogram přerušení

- Podprogram přerušení *(ISR)* vykoná odpovídající operaci, např. načtení stisknuté klávesy nebo aktualizaci časovače.
- Po dokončení se CPU vrátí k předchozí činnosti.

Vektory přerušení

- Vektor přerušení je tabulka adres obslužných rutin.
- Každé přerušení má v této tabulce svou adresu, kterou CPU použije k vykonání ISR.
- Příklad vektorového přerušení pro klávesnici (x86):
  - IRQ1 → Adresa 0x09 → Klávesnicová rutina



2. Mikroprocesor

DMA – Direct Memory Access (Přímý přístup do paměti)

DMA *(Direct Memory Access)* je **hardwarová technologie**, která umožňuje **přímý přenos dat mezi pamětí a periferiemi bez zásahu CPU**. Díky tomu se výrazně snižuje zátěž procesoru a zrychluje přenos velkých objemů dat.

Princip fungování DMA

Běžný přenos dat bez DMA:

1. Procesor načte data z paměti.
2. Procesor přepošle data do periferie *(např. na grafickou kartu)*.
3. Procesor opět načítá nová data a cyklus se opakuje.

Tento proces blokuje CPU, protože se **musí věnovat přesunu dat místo výpočtům**.

Přenos dat s DMA:

- **CPU pouze inicializuje přenos** – nastaví zdrojovou a cílovou adresu, délku přenosu a režim.
- DMA řadič **převezme kontrolu nad sběrnicí** a provede přenos bez zapojení CPU.
- Po dokončení přenosu **DMA vyšle přerušení informující** CPU o dokončení operace.

Typy přenosů s DMA

DMA umožňuje tři typy přenosů:

1. **Periferie ↔ RAM** *(např. přenos dat z ADC do paměti)*.
2. **Periferie ↔ Periferie** *(např. kopírování dat mezi dvěma I/O zařízeními)*.
3. **Paměť ↔ Paměť** *(např. kopírování bloků dat v RAM)*.

Architektura DMA

Dnes je **DMA součástí základní desky** *(integrovaný v čipsetu nebo procesoru)*.
Dříve existoval jako **samostatný řadič DMA**.
DMA obsahuje **vlastní registry pro správu:**

- **Zdrojové adresy** *(odkud se data kopírují)*.
- **Cílové adresy** *(kam se data ukládají)*.
- **Délky přenosu** *(počet bajtů k přenosu)*.
- **Režimu přenosu** *(blokový, burst mód, kontinuální)*.
- **Synchronizace přenosu** *(např. na základě časovače)*.

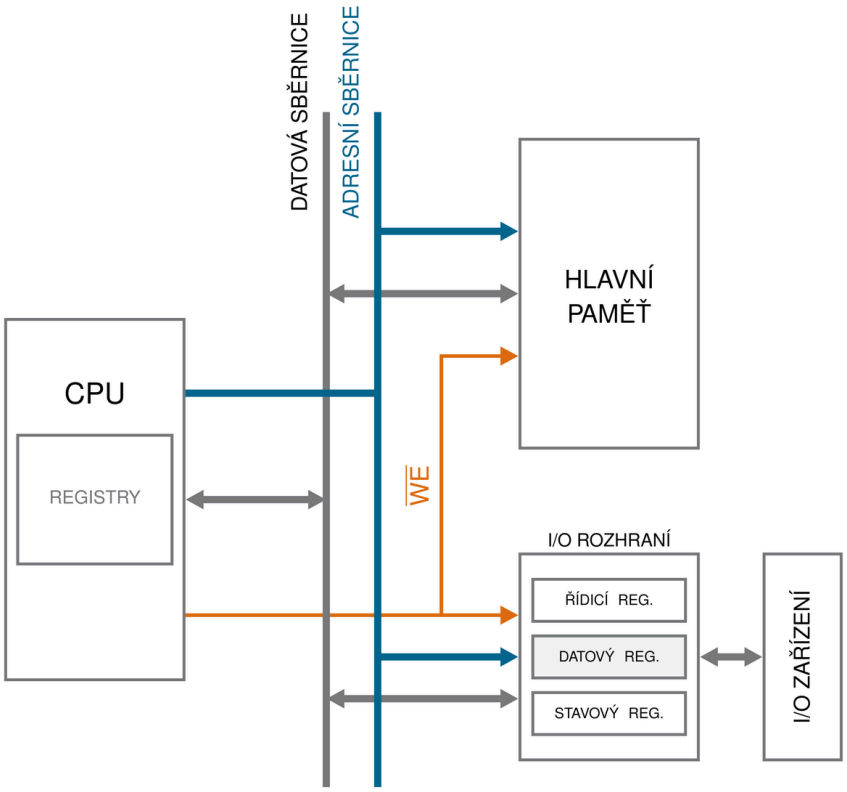
Inicializace a nastavení DMA

Před spuštěním přenosu **je nutné softwarově nastavit DMA řadič**. **To zahrnuje:**

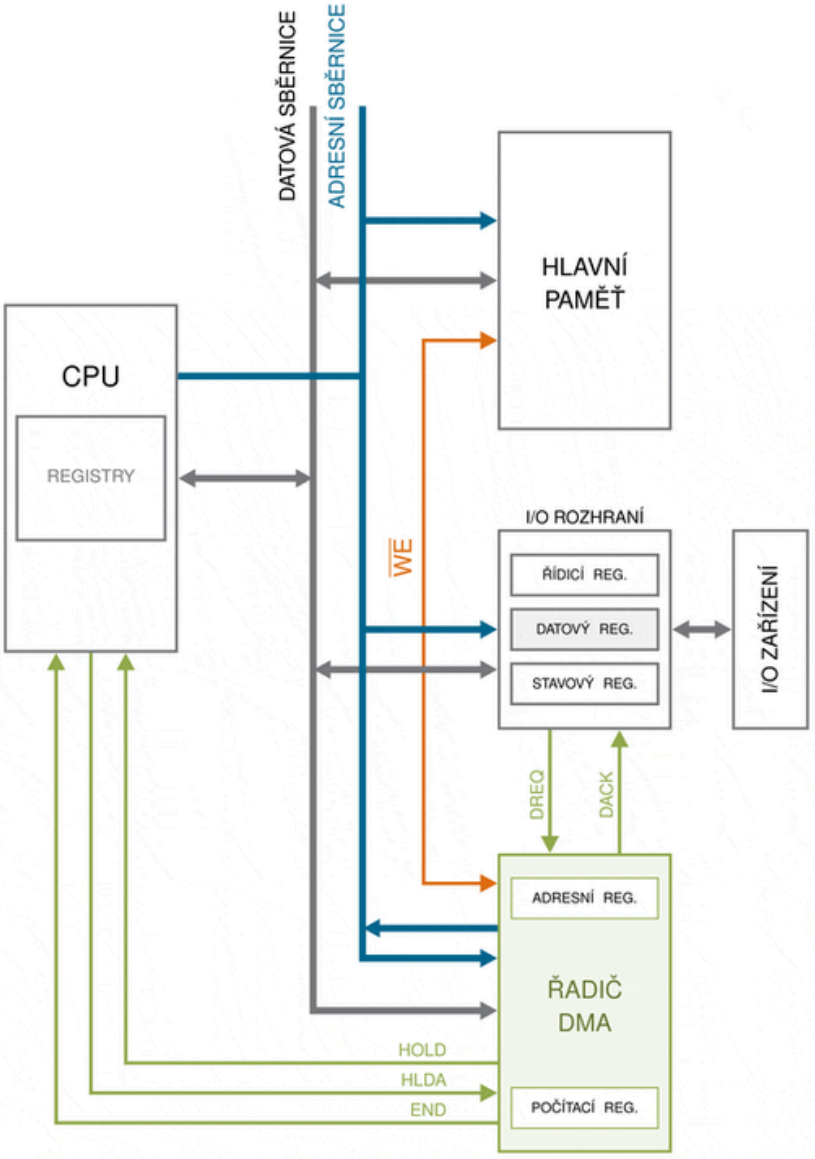
1. **Určení režimu přenosu** – zda se jedná o blokový přenos nebo burst mód.
2. **Typ přenosu** – zda bude data přenášet z periferií do RAM, z RAM do periferií, nebo mezi periferiemi.
3. **Zdrojová adresa** – odkud se data načítají.
4. **Cílová adresa** – kam se data zapisují.
5. **Délka přenosu** – počet bajtů nebo slov k přenosu.
6. **Způsob synchronizace** – kdy má přenos proběhnout *(např. po příchodu určitého signálu)*.
7. **Indikace konce přenosu** – zda se má vyvolat přerušení po dokončení.

2. Mikroprocesor

PIO:



DMA:



2. Mikroprocesor

Princip fungování DMA (Direct Memory Access) podrobněji:

DMA využívá **speciální signály a sběrnice k řízení přenosu**. Proces se skládá z následujících kroků:

Požadavek na přenos dat *(DREQ – DMA Request)*

- Periferie aktivuje signál **DREQ** a tím požádá řadič DMA o **přenos dat mezi pamětí a sebou**.
- DREQ může být generován **časovačem, sériovým portem, AD převodníkem** nebo jinou periferií.

Kontrola nastavení a aktivace přenosu

- Řadič DMA **ověří nastavení přenosového kanálu** – kontroluje:
  - Zdrojovou a cílovou adresu.
  - Velikost přenosu *(počet bajtů/slov)*.
  - Režim přenosu *(jednotlivé bloky, burst mód, kontinuální režim)*.

Převzetí sběrnice *(HOLD & HLDA)*

- Řadič DMA **aktivuje signál HOLD** *(HRQ – Hold Request)* a požádá CPU o **uvolnění sběrnice**.
- Pokud CPU aktuálně nepotřebuje sběrnici, odpoví signálem HLDA *(Hold Acknowledge)* a uvolní ji.

Zahájení přenosu

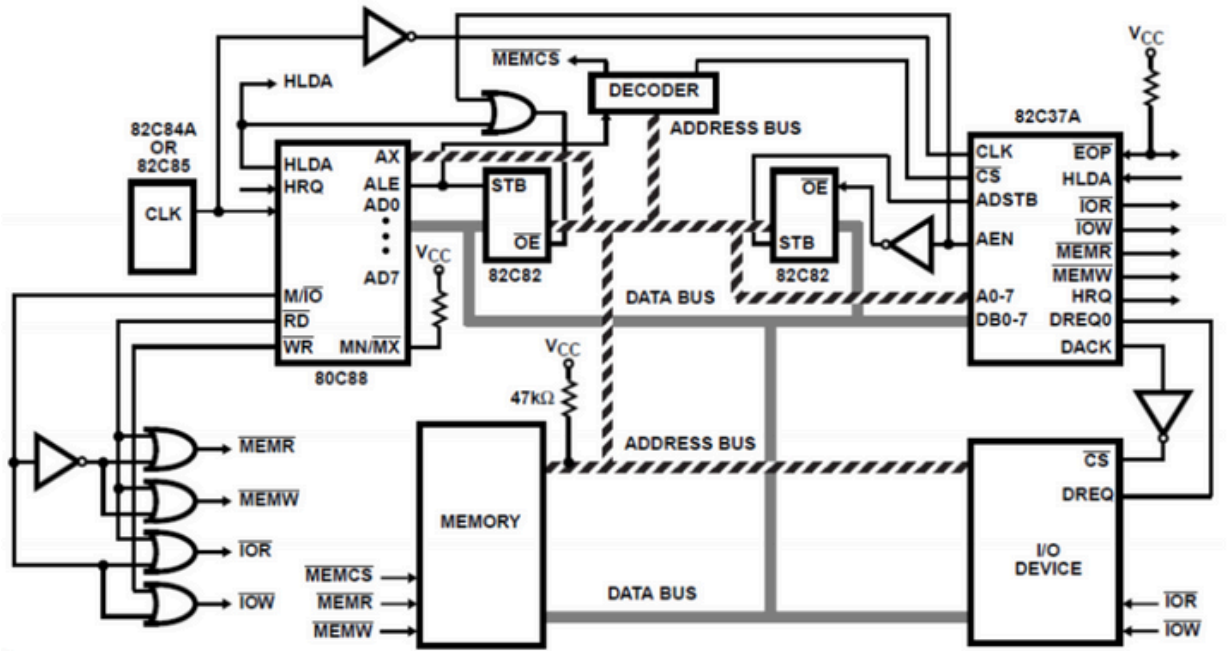
- DMA řadič vystaví adresu v paměti a **nastaví řídicí signály pro čtení/zápis.**
- Používá se **adresní sběrnice A0-A7**, která se posílá ve dvou cyklech *(nižších 8 bitů a vyšších 8 bitů)*.
- Vyšší adresy se posílají přes datovou sběrnici pomocí STB (strobe register).

Přenos dat *(DACK – DMA Acknowledge)*

- DMA aktivuje signál DACK *(DMA Acknowledge)*, což informuje periferii, že může odeslat nebo přijmout data.
- Přenos probíhá nezávisle na CPU a pokračuje, dokud je **DREQ aktivní**.

Ukončení přenosu *(EOP – End of Process)*

- Při posledním slově **DMA aktivuje signál** EOP *(End of Process)*, čímž informuje CPU, že přenos byl dokončen.
- DMA **uvolní signál HOLD**, což znamená navrácení sběrnice CPU.
- CPU **deaktivuje HLDA** a opět přebírá kontrolu nad sběrnicí.



2. Mikroprocesor

Charakteristika STM32F4 + školního kitu

STM32F407G Discovery je **výkonný 32bitový mikrokontrolér** založený na ARM Cortex-M4, který je optimalizován pro **real-time aplikace s vysokou rychlostí a nízkou spotřebou energie**.

Základní vlastnosti STM32F407G Discovery:

- **Procesor:** 32bit ARM Cortex-M4 *(RISC architektura, 5stupňový pipeline)*.
- **Paměť:** 1 MB Flash, 192 + 4 kB RAM.
- **Taktovací frekvence:** Až 168 MHz *(interní 36 MHz oscilátor)*.
- **Debugging:** ST-LINK nástroj pro ladění.

Periferie a vstupně-výstupní možnosti:

- 3× 12bitový AD převodník *(analog → digitál)*.
- 2× 12bitový DA převodník *(digitál → analog)*.
- 14 časovačů.
- 140 GPIO pinů s podporou přerušení.
- 3× I²C a SPI sběrnice.
- USB 2.0 podpora.
- Ethernet 10/100 Mb/s.
- Vestavěný generátor náhodných čísel.
- **Napájení:** 5V přes USB, vstup 3V nebo 5V.

Školní kit obsahuje navíc:

- LCD displej 8×2.
- Maticová klávesnice 4×3.
- 4× LED dioda.
- Sériový port COM.
- Fotorezistor.
- Reproduktor

Konfigurace projektu v keil 5

- Abychom k portu mohli něco připojit, musíme ho **nakonfigurovat**
- K tomu slouží registry *(moder)* - Všechny registry jsou **32 bitové**