Ústav radioelektroniky
Vysoké učení technické v Brně

Bloková struktura mikrokontrolérů

Mikroprocesorová technika a embedded systémy

Přednáška 1

doc. Ing. Tomáš Frýza, Ph.D.

Obsah přednášky

Popis a použití mikrokontroléru, mikroprocesoru a mikropočítače

Realizace řídicí aplikace

Základní typy architektur v mikroprocesorové technice
Von Neumann, Harvardská, CISC, RISC, VLIW, ...

Obecná bloková struktura mikrokontrolérů
Aritmeticko/logická jednotka, centrální řídicí jednotka, paměti, vstupně/výstupní obvody

Ukázka programu v JSA pro AVR
Práce s registry, aritmetické operace
Ovládání vstupně/výstupního portu

Obsah přednášky

Popis a použití mikrokontroléru, mikroprocesoru a mikropočítače

Realizace řídicí aplikace

Základní typy architektur v mikroprocesorové technice
- Von Neumann, Harvardská, CISC, RISC, VLIW, ...

Obecná bloková struktura mikrokontrolérů
- Aritmeticko/logická jednotka, centrální řídicí jednotka, paměti, vstupně/výstupní obvody

Ukázka programu v JSA pro AVR
- Práce s registry, aritmetické operace
- Ovládání vstupně/výstupního portu

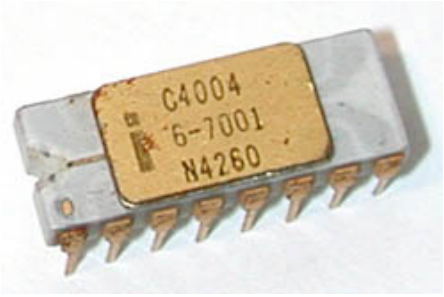
Mikroprocesor

Definice (Mikroprocesor)

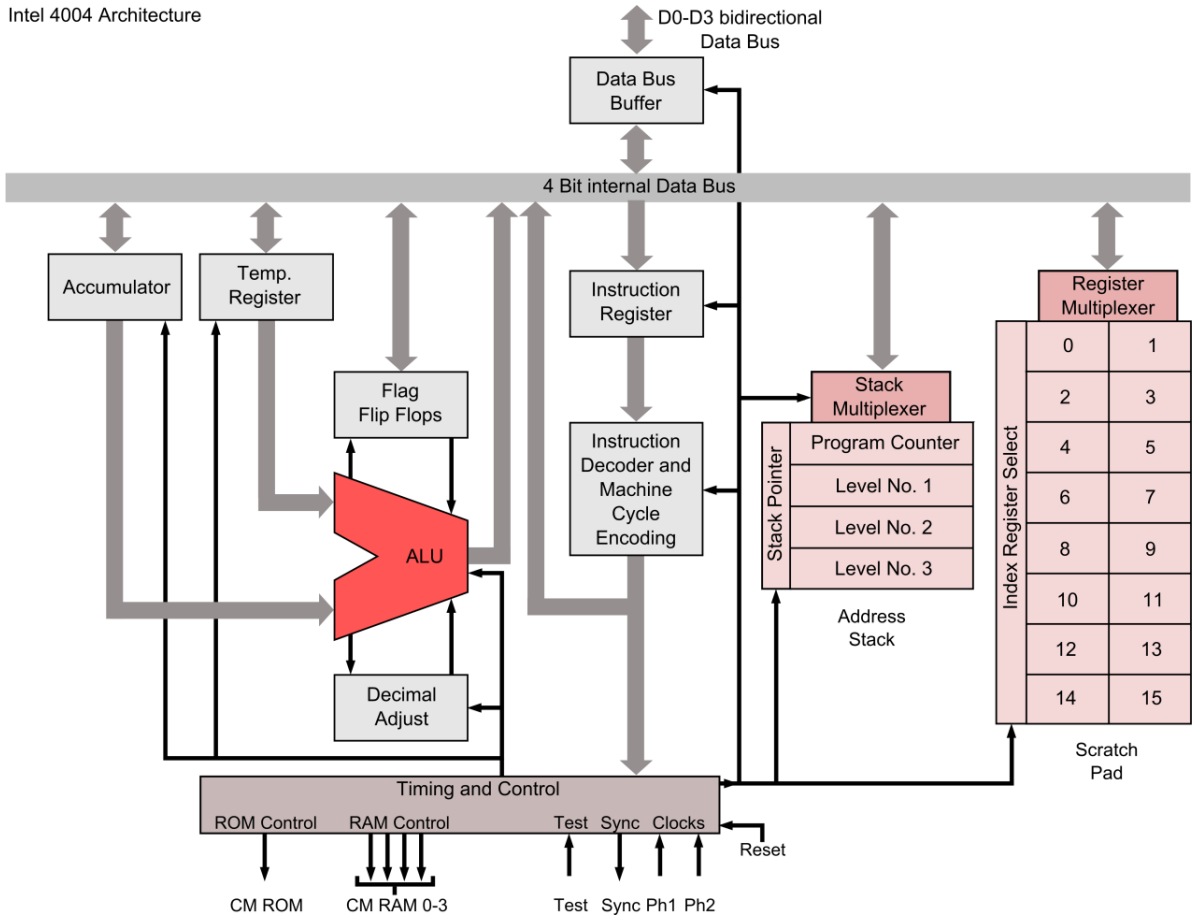
Mikroprocesor (MPU – Microprocessor Unit) je centrální řídicí jednotka (CPU – Central Processing Unit) na samostatném čipu.

- První mikroprocesor vyvinula firma Intel v roce 1971 pod označením 4004:
  - 4bitová CPU, 16pinové pouzdro, hodinový signál o frekvenci 740 kHz, Harvardská architektura (tj. oddělená paměť pro program a data),
  - jediná multiplexovaná 4bitová sběrnice přenášela 12bitovou adresu ve třech krocích ($\Leftrightarrow$ maximální adresní prostor 4 kB), 8 bitů instrukce, 4 bity data,
  - instrukční sada: 46 instrukcí (41 8bitových, 5 16bitových instrukcí), 16 4bitových registrů,
  - původně určen pro kalkulačky.

Mikroprocesor



Obrázek: Pouzdro 4004.



Obrázek: Bloková struktura mikroprocesoru 4004.

Mikropočítač

Definice (Mikropočítač)

Doplněním mikroprocesoru o podpůrné obvody, tj. vstupně/výstupní periferie a paměť (pro program i data) vznikne mikropočítač.

- První mikropočítače vznikaly v polovině 70tých let (logicky bezprostředně po vzniku mikroprocesorů), zpravidla bez klávesnice a displeje. Velikost paměti typicky 4 až 16 kB.
- Mikropočítač je obecně určen pro zpracování dat a řízení procesů.
- Jedním z prvních mikropočítačů byl Altair 8800 z roku 1975, obsahující 8bitový mikroprocesor Intel 8080A, hodinový signál 2 MHz, velikost paměti RAM 256 B až 64 kB, 78 instrukcí.

Počátky mikropočítačů

- Obsah paměti se zapisoval a četl pomocí přepínačů a LED diod. Přepínači se nastavila požadovaná adresa; následně se navolila osmice bitů (8bitový systém); LED signalizovaly obsah vybrané paměťové buňky.

Obrázek: Mikropočítač Altair 8800.

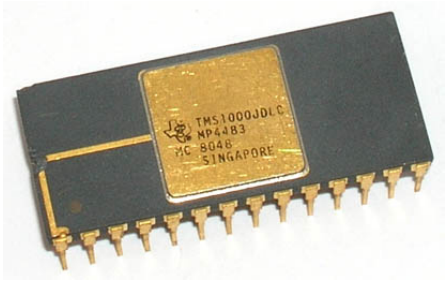
Mikrokontrolér

Definice (Mikrokontrolér)

Mikrokontrolér (MCU – Microcomputer Unit) vznikne sdružením všech částí mikropočítače (řídící jednotka, paměti RAM, ROM, vstup/výstup, časovač/čítač, a jiné periférie) na jediný čip (součástku).

- Vyznačují se nízkými náklady, nízkou spotřebou a dostatečnou hardwarovou výbavou pro řízení jednoduchých aplikací.
- Struktura a funkce mikrokontrolérů se do současnosti příliš nezměnila. Základní dělení mikrokontrolérů je podle šířky registrů a sběrnice (nejčastěji 8bitové a 16bitové, příp. 32bitové).

Mikrokontrolér



Obrázek: Mikrokontrolér TMS1000 firmy Texas Instruments.

- První mikrokontroléry vytvořila firma Texas Instruments pod označením TMS1000 v roce 1974:
  - 28pinové pouzdro, hodinový signál o frekvenci 400 kHz, 4bitová sběrnice, velikost paměti RAM 32 B, ROM 1 kB, obsahovaly periférie: oscilátor, 4 vstupní piny, 11 výstupních, 8bitový výstupní paralelní port.
- Nyní existuje velké množství výrobců i dodávaných řad mikrokontrolérů. Některé řady 8bitových mikrokontrolérů: 8051 (Intel), 68HCS08, (Freescale), Z8 (Zilog), PIC (Microchip), H8 (Hitachi), AVR (Atmel), ...
- Moderní mikrokontroléry mohou obsahovat velké množství periférií (GPIO, časovač/čítač, A/D převodník, analogový komparátor, sériové sběrnice I2C, USB, CAN, ...).

Obsah přednášky

Popis a použití mikrokontroléru, mikroprocesoru a mikropočítače

Realizace řídicí aplikace

Základní typy architektur v mikroprocesorové technice
- Von Neumann, Harvardská, CISC, RISC, VLIW, ...

Obecná bloková struktura mikrokontrolérů
- Aritmeticko/logická jednotka, centrální řídicí jednotka, paměti, vstupně/výstupní obvody

Ukázka programu v JSA pro AVR
- Práce s registry, aritmetické operace
- Ovládání vstupně/výstupního portu

Jednoduché aplikace řízené mikrokontroléry

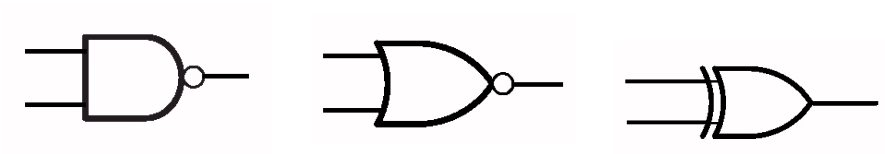
- Typická řídicí aplikace s mikrokontroléry: neustálý sběr vstupních dat, jejich zpracování/vyhodnocování, poskytnutí výstupních informací (dat).
  - Realizace digitálního osciloskopu,
  - aplikace kapacitních snímačů v technické praxi,
  - reklamní LED RGB trubice s ovládací jednotkou,
  - PC osciloskop – hardwarová část,
  - jednoduchý digitální fotoaparát,
  - ...
  - Více na `http://www.urel.feec.vutbr.cz/~fryza/?Samostatn%26eacute%3B_projekty`
  - Přenos obrazových dat z RC modelu (`http://www.wiredhouse.fr/R10SD/index.html`).

Realizace řídicí aplikace

- Libovolnou řídicí aplikaci lze zpravidla vyřešit několika způsoby. Vždy záleží na složitosti, na dosažitelných parametrech (rychlost, přesnost, ...) nákladech (finance, čas), rozměrech, požadované variabilitě, ...
- Obecně existují tyto základní možnosti:
  - logická součástka s požadovanou funkcí,
  - použití jednotlivých logických členů (AND, OR, XOR),
  - obvody PROM, multiplexor,
  - programovatelné logické obvody (PAL, PLD, CPLD, FPGA),
  - specifické/zákaznické obvody ASIC (Application-Specific Integrated Circuit),
  - mikrokontroléry.

Realizace funkce pomocí jednotlivých logických členů

- NAND – logický součin
- NOR – logický součet
- XOR – exkluzivní součet
- viz předmět B/K/ICT



Obrázek: Členy logického součinu NAND, součtu NOR a exkluzivního součtu XOR.

Tabulka: Pravdivostní tabulka logických funkcí

A	B	$\overline{A}$	$A \cdot B$	$A + B$	$A \oplus B$
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Realizace funkce pomocí logických členů

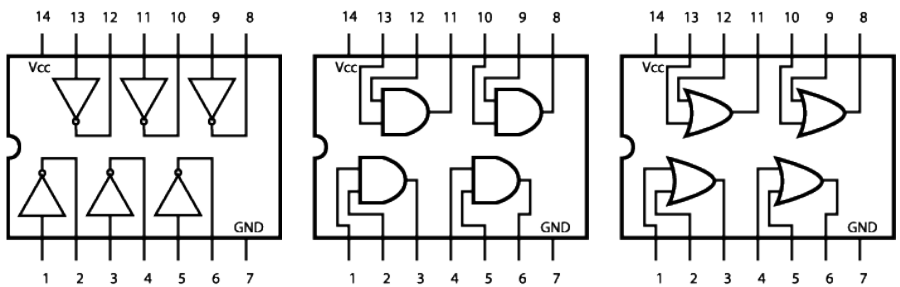
Příklad

Realizujte kombinační logickou funkci $f = [(\bar{a}\, b) + (a\, \overline{b})] \cdot (c\, \overline{d})$ *pomocí jednotlivých hradel.*

Řešení

Využití 3 integrovaných obvodů TTL:

- *invertor 7404 obsahuje 6 hradel – využity jen 3 (50 %),*
- *AND 7408 (logický součin): 4 členy – 4 využity (100 %),*
- *OR 7432 (logický součet): 4 členy – 1 využit (25 %),*
- *z dostupných 14 hradel je využito jen 8, tj. 57,14 %.*



Realizace funkce pomocí PAL

PAL (Programmable Array Logic)

Příklad

Realizujte kombinační logickou funkci $f = [(\overline{a}\, b) + (a\, \overline{b})] \cdot (c\, \overline{d})$ *pomocí programovatelného logického obvodu.*

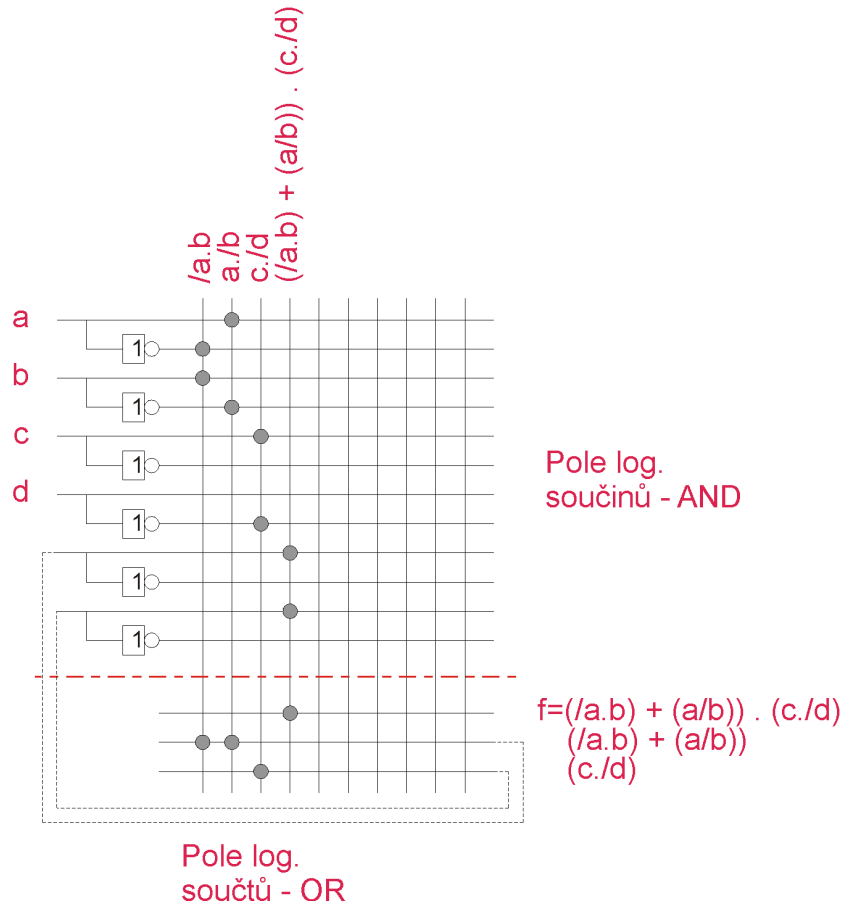
Řešení

Využití 1 obvodu GAL 16L8:

- *obvod obsahuje přibližně 150 hradel, z nichž bylo použito 12 (8 hradel logického součinu a 4 hradla logického součtu), tj. 8 % celkového počtu.*

- Složitější struktura má za následek větší spotřebu v porovnání s obvody ASIC (Application-Specific Integrated Circuit), které jsou určeny pro konkrétní aplikace.

Realizace funkce pomocí PLD



$$f = [(\overline{a}\, b) + (a\, \overline{b})] \cdot (c\, \overline{d})$$

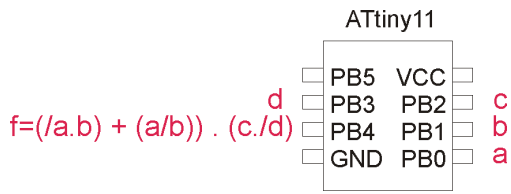
Obrázek: Vnitřní propojení obvodu GAL 16L8.

Příklad

Realizujte kombinační logickou funkci $f = [(\overline{a}\, b) + (a\, \overline{b})] \cdot (c\, \overline{d})$ *pomocí mikrokontroléru.*

Řešení

Použití mikrokontroléru, obsahující min. 5 vstupně/výstupních pinů (4 vstupy, 1 výstup – např. ATtiny11) a korektně napsaný obslužný program.



- Srovnatelná fyzická velikost s PLD, ale nižší spotřeba a univerzálnost aplikace.
- Nižší rychlost zpracování oproti PLD.

Výhody/nevýhody použití mikrokontrolérů

- Méně součástek v systému způsobí nižší náklady na výrobu plošných spojů.
- Větší spolehlivost v důsledku menšího počtu propojení.
- Nižší napěťové nároky na použité obvody, tj. snadnější návrh napěťové části zařízení.
- Jednodušší vývoj a testování. Změnit funkci lze pouhým přeprogramováním bez nutnosti zásahu do hardware.
- Rozšíření, doplnění stávající funkce celé aplikace snadným přeprogramováním.
- Nižší rychlost zpracování než logické obvody. Nižší rychlost je způsobena sekvenční podstatou vykonávaného programu.
- V některých aplikacích je výhodnější použít PLD v kombinaci s mikrokontrolérem, příp. samostatné PLD.

Obsah přednášky

Popis a použití mikrokontroléru, mikroprocesoru a mikropočítače

Realizace řídicí aplikace

Základní typy architektur v mikroprocesorové technice
Von Neumann, Harvardská, CISC, RISC, VLIW, ...

Obecná bloková struktura mikrokontrolérů
Aritmeticko/logická jednotka, centrální řídicí jednotka, paměti, vstupně/výstupní obvody

Ukázka programu v JSA pro AVR
Práce s registry, aritmetické operace
Ovládání vstupně/výstupního portu

Základní dělení mikropočítačů podle architektury

- První dělení mikropočítačů iniciovala americká vláda v 70. letech, když požádala Princetonskou a Harvardskou univerzitu, aby navrhly architekturu vhodnou pro potřeby dělostřelectva.
- Vznikly dvě základní koncepce:
  - Von Neumannova,
  - Harvardská.
- Von Neumannova architektura:
  - popisuje jak má číslicový systém pracovat a z jakých hlavních částí by se měl skládat: řídicí jednotka, paměti, I/O obvody,
  - zásadní myšlenka von Neumannovy architektury je použití pouze jedné paměti a to pro kontrolní program (instrukce) i pro data (proměnné, . . .) – obojí je "jedno a totéž"!
  - nekoresponduje s vyššími programovacími jazyky; např. neumožňuje pracovat s vícerozměrnými poli. Von Neumannovu architekturu využívají dneska počítače typu PC.
  - program je vykonáván sekvenčně, tj. instrukce se provádějí tak jak jdou za sebou – "až na ně dojde řada",
  - změnu pořadí vykonávaní instrukcí lze provést jen podmíněným skokem, nepodmíněným skokem, či voláním podprogramu a přerušením,
  - vnitřní architektura je nezávislá na řešené úloze. Veškeré změny mají být řešeny softwarově, tzn. počítač je řízen obsahem paměti,
  - původní přednost v univerzálnosti architektury je ve svém důsledku nevýhodná – systém dokáže zpracovat libovolný problém, ale neefektivně.

von Neumannova architektura

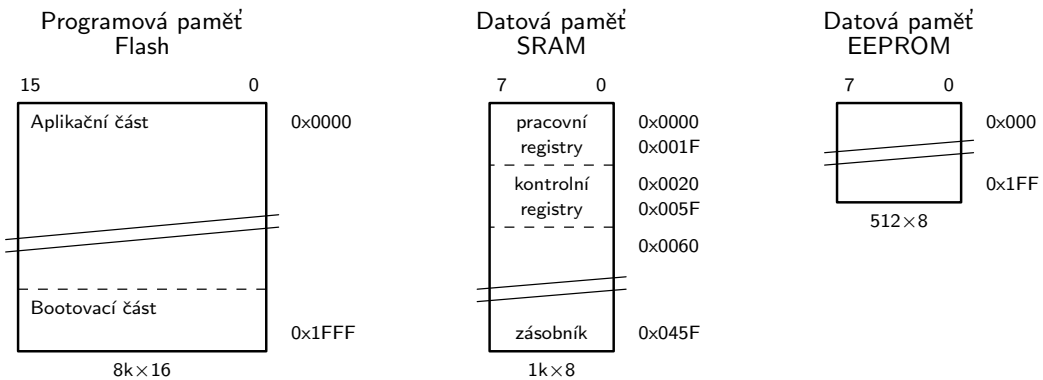
Von Neumanovy postuláty, pravidla, pokračování:

- neexistuje princip paralelismu.
  - Výhodnější a přehlednější pro tvorbu aplikací; paralelní programování je složité a špatně čitelné (viz např. Linux),
  - nevýhodné pro optimalizaci výkonu jednotlivých částí systému (vždy je zatížena pouze jedna část).
- Paměť je rozdělena na stejně velké buňky, jejichž pořadové čísla se využívají jako identifikační adresy.

Obrázek: Paměťové buňky.

Harvardská architektura

- Harvardská architektura chronologicky navazuje na architekturu von Neumannovu a mění některé její vlastnosti.
- Zásadní rozdíl je oddělená část paměti pro program a data:
  - program tak nemůže přepsat sám sebe,
  - možnost použití pamětí odlišných technologií (EEPROM, Flash, . . .),
  - dvě sběrnice (pro instrukce, pro data) umožňují současný přístup k instrukcím i k datům,
  - nevyužitou část paměti pro data ovšem nelze využít pro uložení programu a naopak.
- Sekvenční vykonávání instrukcí zachováno, tzn. že paralelní zpracování lze provádět pouze na úrovni operačního systému.



Obrázek: Koncepce oddělené paměti Harvardské architektury u 8bitového mikrokontroléru ATmega16.

- Paměť pro program v paměti typu Flash, šířka slova 16 b.
- Paměť pro data v paměti SRAM: 32 obecných pracovních registrů, 64 vstupně/výstupních (I/O) registrů, interní/externí paměť RAM.
- Datová paměť EEPROM: např. pro tabulky hodnot.

Procesory CISC/RISC

- Dosavadní dělení procesorů výlučně podle hardwaru. Dále základní dělení procesorů z pohledu instrukční sady:
  - CISC (Complex Instruction Set Computer – Počítač s komplexním souborem instrukcí),
  - RISC (Reduced Instruction Set Computer – Počítač s redukovaným souborem instrukcí).
- Prakticky neexistují "ryzí" procesory CISC nebo RICS, vždy se jedná o kompromis.
- CICS procesory obsahují velké množství instrukcí, které s malými obměnami vykonávají ty samé operace (např. pomocí přímého adresování, indexového adresování, apod.) – lze je snadno nahradit posloupnosti jiných instrukcí.
- Výskyt některých instrukcí je velmi nízký $\Rightarrow$ proč mít tyto instrukce v instrukčním souboru?
- Každá instrukce rozšiřuje řadič procesoru – procesor musí "rozpoznat" všechny instrukce $\Rightarrow$ zvyšuje se hardwarová složitost.

Tabulka: Statistická četnost typů instrukcí v programech

Operace	**Četnost**	
Načítání z paměti	27,3 %	
Podmíněný skok	13,7 %	
Zápis do paměti	9,8 %	$\sum$50,8 %
Porovnávání hodnot	6,2 %	
Načtení adresy	6,1 %	
Odečítání	4,5 %	
Vložení znaku	4,1 %	
Sečítání	3,7 %	$\sum$75,4 %

Procesory CISC/RISC

- Neudržitelný nárůst složitosti CISC procesorů vedl na konci 70tých let k vývoji zjednodušené struktury RISC.
- Statistický výzkum měl za úkol nalézt optimální instrukční soubor pro procesory typu RISC.
- Procesory RISC se kromě malého počtu instrukcí vyznačují také:
  - malým počtem způsobů adresování,
  - používá zřetězené zpracování instrukcí,
  - instrukce mají pevnou délku (u AVR 16 bitů) a jednotný formát, což urychluje jejich dekódování,
  - používají větší počet rovnocenných registrů (u AVR 32 reg. R0, R1, . . ., R31) na rozdíl od tzv. Akumulátoru (příp. střadače) u CISC.
- Ukázka formátu instrukce `ADD Rd, Rr` (součet u AVR: Rd=Rd+Rr)
  - `0000 11rd dddd rrrr`
  - `d` $\in \{0; 31\}$ – identifikátor registru (1. operand, výsledek)
  - `r` $\in \{0; 31\}$ – identifikátor 2. operandu
- Ukázka formátu instrukce `SUB Rd, Rr` (rozdíl u AVR: Rd=Rd-Rr)
  - `0001 10rd dddd rrrr`
  - `d` $\in \{0; 31\}$ – identifikátor registru (1. operand, výsledek)
  - `r` $\in \{0; 31\}$ – identifikátor 2. operandu
- Výsledný program pro procesory RISC:
  - je zpravidla delší z důvodu většího počtu instrukcí s konstantním počtem bitů,
  - doba vykonání programu může být kratší, protože většina instrukcí se vykoná v jednom hodinové cyklu.

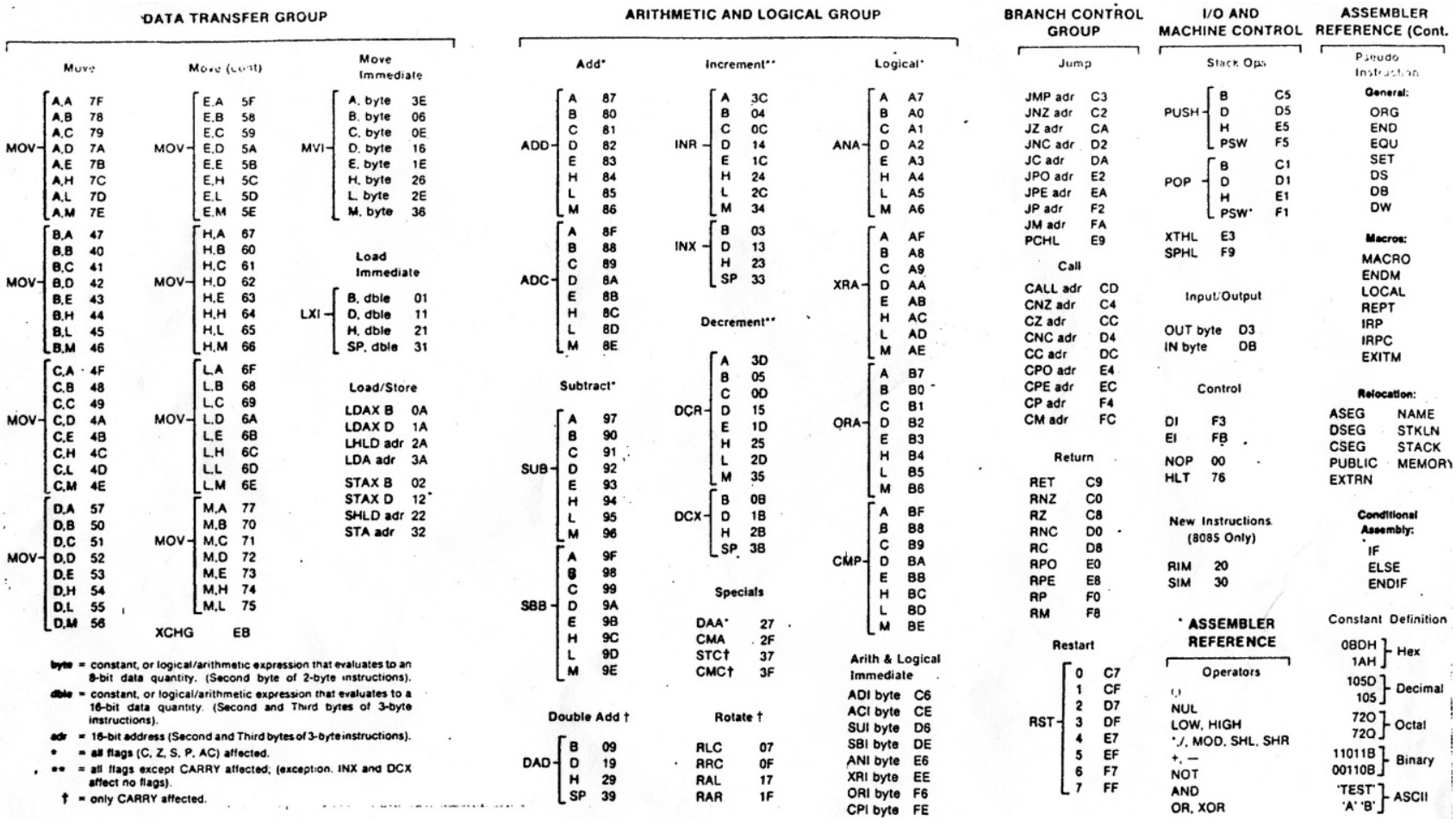
Historické srovnání CISC/RISC

Tabulka: Srovnání testovacích aplikací mikrokontrolérů Intel 8085 a ATmega16

Funkce/aplikace	**Intel 8085**	**ATmega16**
Zpoždění (delay) – velikost	6 B	8 B
Zpoždění (delay) – rychlost*	10,5 ms	2 ms
Stopky – velikost	60 B	92 B (JSA) 3,5 kB (jazyk C)

* Počet opakování funkce: 1 000×, $f_{CPU} = 2$ MHz.

Instrukční sada Intel 8085



DATA TRANSFER GROUP

Move

MOV	A,A	7F
	A,B	78
	A,C	79
	A,D	7A
	A,E	7B
	A,H	7C
	A,L	7D
	A,M	7E
MOV	B,A	47
	B,B	40
	B,C	41
	B,D	42
	B,E	43
	B,H	44
	B,L	45
	B,M	46
MOV	C,A	4F
	C,B	48
	C,C	49
	C,D	4A
	C,E	4B
	C,H	4C
	C,L	4D
	C,M	4E
MOV	D,A	57
	D,B	50
	D,C	51
	D,D	52
	D,E	53
	D,H	54
	D,L	55
	D,M	56

Move (cont)

MOV	E,A	5F
	E,B	58
	E,C	59
	E,D	5A
	E,E	5B
	E,H	5C
	E,L	5D
	E,M	5E
MOV	H,A	67
	H,B	60
	H,C	61
	H,D	62
	H,E	63
	H,H	64
	H,L	65
	H,M	66
MOV	L,A	6F
	L,B	68
	L,C	69
	L,D	6A
	L,E	6B
	L,H	6C
	L,L	6D
	L,M	6E
MOV	M,A	77
	M,B	70
	M,C	71
	M,D	72
	M,E	73
	M,H	74
	M,L	75

XCHG EB

Move Immediate

MVI	A, byte	3E
	B, byte	06
	C, byte	0E
	D, byte	16
	E, byte	1E
	H, byte	26
	L, byte	2E
	M, byte	36

Load Immediate

LXI	B, dble	01
	D, dble	11
	H, dble	21
	SP, dble	31

Load/Store

LDAX B	0A
LDAX D	1A
LHLD adr	2A
LDA adr	3A
STAX B	02
STAX D	12
SHLD adr	22
STA adr	32

byte = constant, or logical/arithmetic expression that evaluates to an 8-bit data quantity. (Second byte of 2-byte instructions).

dble = constant, or logical/arithmetic expression that evaluates to a 16-bit data quantity. (Second and Third bytes of 3-byte instructions).

adr = 16-bit address (Second and Third bytes of 3-byte instructions).

* = all flags (C, Z, S, P, AC) affected.

** = all flags except CARRY affected; (exception: INX and DCX affect no flags).

† = only CARRY affected.

ARITHMETIC AND LOGICAL GROUP

Add*

ADD	A	87
	B	80
	C	81
	D	82
	E	83
	H	84
	L	85
	M	86
ADC	A	8F
	B	88
	C	89
	D	8A
	E	8B
	H	8C
	L	8D
	M	8E

Subtract*

SUB	A	97
	B	90
	C	91
	D	92
	E	93
	H	94
	L	95
	M	96
SBB	A	9F
	B	98
	C	99
	D	9A
	E	9B
	H	9C
	L	9D
	M	9E

Double Add †

DAD	B	09
	D	19
	H	29
	SP	39

Increment**

INR	A	3C
	B	04
	C	0C
	D	14
	E	1C
	H	24
	L	2C
	M	34
INX	B	03
	D	13
	H	23
	SP	33

Decrement**

DCR	A	3D
	B	05
	C	0D
	D	15
	E	1D
	H	25
	L	2D
	M	35
DCX	B	0B
	D	1B
	H	2B
	SP	3B

Specials

DAA*	27
CMA	2F
STC†	37
CMC†	3F

Rotate †

RLC	07
RRC	0F
RAL	17
RAR	1F

Logical*

ANA	A	A7
	B	A0
	C	A1
	D	A2
	E	A3
	H	A4
	L	A5
	M	A6
XRA	A	AF
	B	A8
	C	A9
	D	AA
	E	AB
	H	AC
	L	AD
	M	AE
ORA	A	B7
	B	B0
	C	B1
	D	B2
	E	B3
	H	B4
	L	B5
	M	B6
CMP	A	BF
	B	B8
	C	B9
	D	BA
	E	BB
	H	BC
	L	BD
	M	BE

Arith & Logical Immediate

ADI byte	C6
ACI byte	CE
SUI byte	D6
SBI byte	DE
ANI byte	E6
XRI byte	EE
ORI byte	F6
CPI byte	FE

BRANCH CONTROL GROUP

Jump

JMP adr	C3
JNZ adr	C2
JZ adr	CA
JNC adr	D2
JC adr	DA
JPO adr	E2
JPE adr	EA
JP adr	F2
JM adr	FA
PCHL	E9

Call

CALL adr	CD
CNZ adr	C4
CZ adr	CC
CNC adr	D4
CC adr	DC
CPO adr	E4
CPE adr	EC
CP adr	F4
CM adr	FC

Return

RET	C9
RNZ	C0
RZ	C8
RNC	D0
RC	D8
RPO	E0
RPE	E8
RP	F0
RM	F8

Restart

RST	0	C7
	1	CF
	2	D7
	3	DF
	4	E7
	5	EF
	6	F7
	7	FF

I/O AND MACHINE CONTROL

Stack Ops

PUSH	B	C5
	D	D5
	H	E5
	PSW	F5
POP	B	C1
	D	D1
	H	E1
	PSW*	F1
XTHL		E3
SPHL		F9

Input/Output

OUT byte	D3
IN byte	DB

Control

DI	F3
EI	FB
NOP	00
HLT	76

New Instructions (8085 Only)

RIM	20
SIM	30

ASSEMBLER REFERENCE

Operators

( )
NUL
LOW, HIGH
*/, MOD, SHL, SHR
+, —
NOT
AND
OR, XOR

ASSEMBLER REFERENCE (Cont.

Pseudo Instruction

General:

ORG
END
EQU
SET
DS
DB
DW

Macros:

MACRO
ENDM
LOCAL
REPT
IRP
IRPC
EXITM

Relocation:

ASEG	NAME
DSEG	STKLN
CSEG	STACK
PUBLIC	MEMORY
EXTRN	

Conditional Assembly:

IF
ELSE
ENDIF

Constant Definition

0BDH, 1AH	Hex
105D, 105	Decimal
72O, 72Q	Octal
11011B, 00110B	Binary
'TEST', 'A' 'B'	ASCII

Procesory VLIW

- Procesory VLIW (Very Long Instruction Word–velmi dlouhé instrukční slovo) umožňují efektivnější vykonání programu z důvodu paralelního zpracování instrukcí.
- Paralelismus je realizován větším počtem funkčních jednotek (v jádře) se specifickými funkcemi:
  - aritmetické operace, hardwarová násobička, komunikace s pamětí, bitové operace, . . .,
  - každá může pracovat nezávisle na ostatních a všechny mohou pracovat současně.
- Podstata paralelního zpracování: zatímco se provádí např. součet dvou čísel, je možné jiné operandy násobit a z paměti si načíst další hodnoty.
- Zástupce VLIW architektury je signálový procesor řady TMS320C6000 (fy Texas Instruments) – viz konec semestru.

Obsah přednášky

Popis a použití mikrokontroléru, mikroprocesoru a mikropočítače

Realizace řídicí aplikace

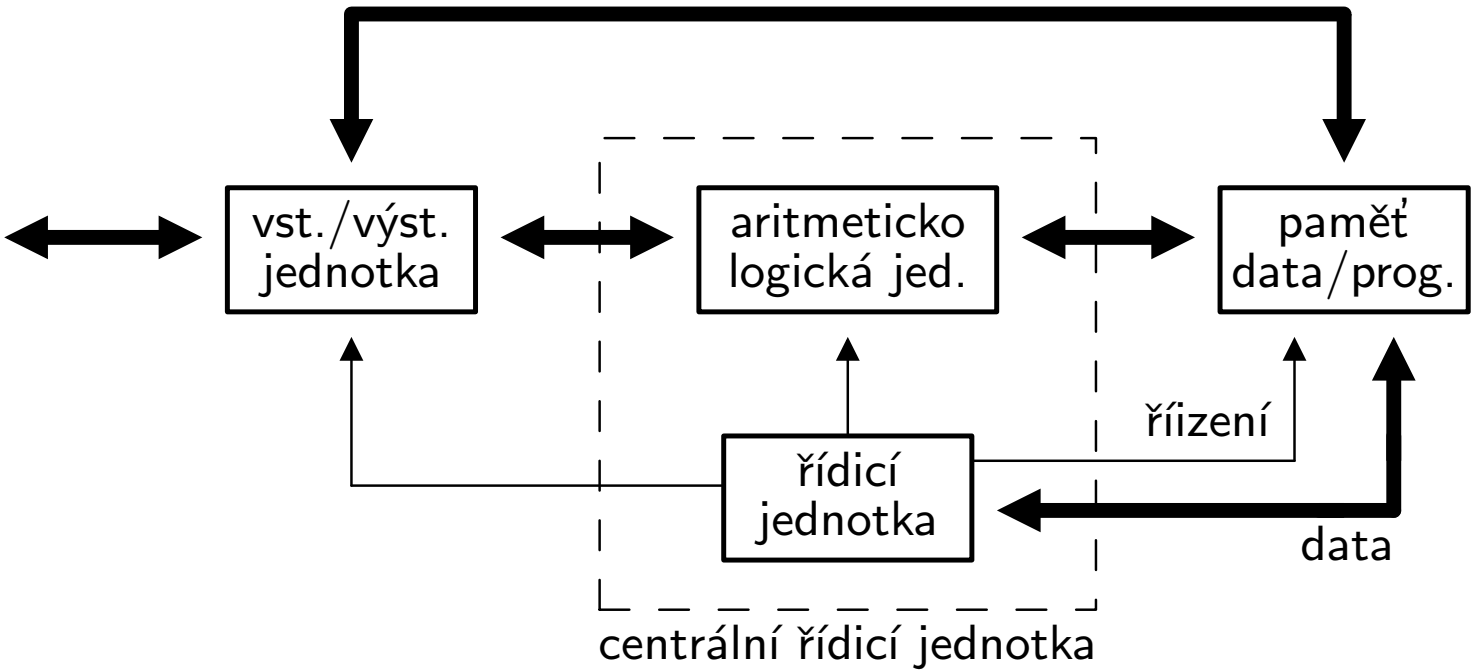
Základní typy architektur v mikroprocesorové technice
- Von Neumann, Harvardská, CISC, RISC, VLIW, ...

Obecná bloková struktura mikrokontrolérů
- Aritmeticko/logická jednotka, centrální řídicí jednotka, paměti, vstupně/výstupní obvody

Ukázka programu v JSA pro AVR
- Práce s registry, aritmetické operace
- Ovládání vstupně/výstupního portu

Obecná bloková struktura mikrokontrolérů

- Každý mikropočítač obsahuje podle von Neumanna 3 základních částí:
  - centrální řídicí jednotku CPU (včetně aritmeticko/logické jednotky a řídicí jednotky),
  - paměť(i) pro obslužný program, příp. data,
  - vstupně/výstupní jednotku pro komunikaci s externími zařízeními.

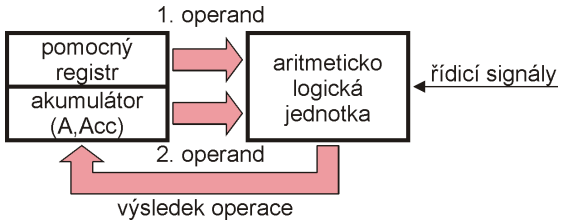


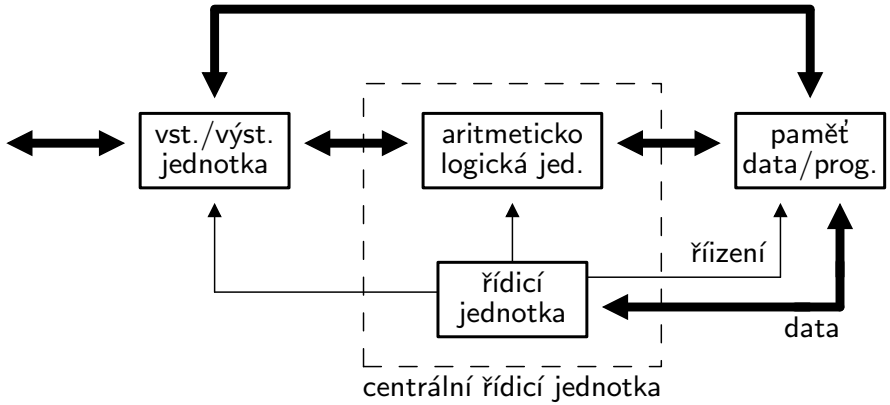
Obrázek: Principiální struktura mikropočítače.

Centrální řídicí jednotka

- CPU (Central Processing Unit) představuje řídicí mozek celého systému (nejdůležitější část):
  - obsahuje aritmetricko/logickou jednotku a řídicí jednotku,
  - kombinuje obvody generující všechny řídicí signály pro výkon instrukcí s obvody pro samotný výkon volaných instrukcí,
  - např. instrukce `ADD R16, R18` řídí volbu operandů (registry R16 a R18) a typ operace (součet).
- Aritmeticko/logická jednotka (ALU – Arithmetic/Logic Unit) provádí aritmetické a logické operace s daty, která jsou reprezentována dvěma binárními čísly:
  - sčítání, odčítání, násobení, dělení, odmocnina, exponenciála, bitový posun (shift), logické operace (AND, OR, . . .),
  - Pozn.: Uvedené operace nemusí být obsaženy ve všech ALU, záleží na jejich složitosti,
  - jednoduché mikrokontroléry umožňují jen některé z uvedených operací; ostatní lze "poskládat" ze stávajících instrukcí (např.: násobení bit po bitu).

Aritmeticko/logická jednotka





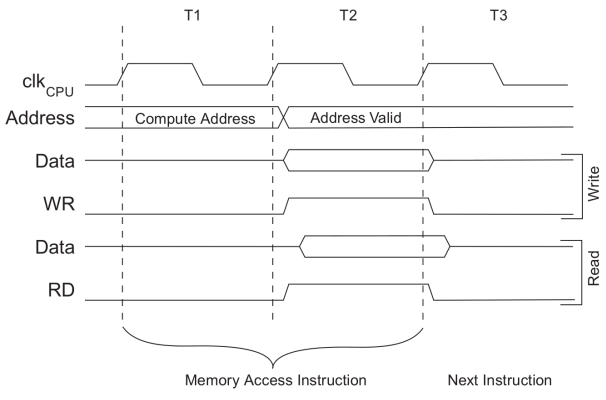
- Zdroj dat určuje řídicí jednotka; buď je jím obsah paměti – v podstatě také registr, nebo vstupní obvody.
- Pozn.: Akumulátor:
  - nejvýznamnější registr u CISC; podílí se na většině operací ALU a ukládá většinu výsledků (např. u MCU řady '51, 68HC11, . . .),
  - některé MCU jej nahradili souborem rovnocenných registrů (RISC).
- Přesný typ operace oznamuje řídicí jednotka pomocí řídicích signálů; výsledek operace je uložen buď do paměti/registru nebo do výstupní jednotky.
- ALU vrací dva typy výsledků:
  - aritmetickou nebo logickou hodnotu – ukládá do paměti/registru, výstupní jednotky,
  - příznakové bity uložené ve speciálním registru (označován např. PSW – Processor status word, SREG – Status Register, . . .) charakterizující výsledky nejpoužívanějších operací – vhodné např. pro větvení programu.

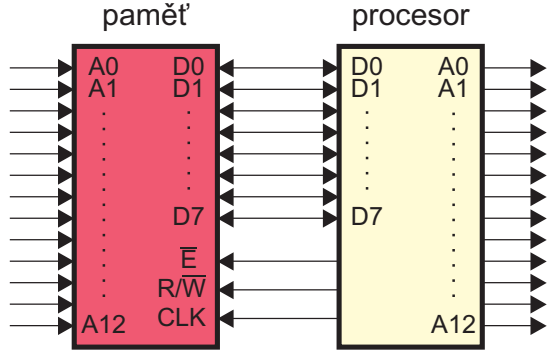
Řídicí jednotka

- Řídicí jednotka obsahuje logické a časovací obvody, generující signály potřebné pro výkon každé instrukce v programu.
- Řídicí signály mohou být z pohledu mikrokontroléru chápany jako vstupní ($\overline{IRQ}$), jiné jako výstupní ($R/\overline{W}$, E, $IO/\overline{M}$, ...), příp. obousměrné.
  - Output enable $\overline{OE}$, Input enable $\overline{IE}$.
  - Signál $R/\overline{W}$ je generován mikrokontrolérem:
    - informuje ostatní zařízení o směru čtení/zápisu dat z/na datovou sběrnici ($R/\overline{W} = 1 \Leftrightarrow$ READ; $R/\overline{W} = 0 \Leftrightarrow$ WRITE),
    - zajišťuje korektní komunikaci mezi pamětí, příp. I/O obvody a mikrokontrolérem.
  - Signál $\overline{IRQ}$ (Interrupt Request) informuje mikrokontrolér o požadavku na přerušení:
    - jedno nebo více I/O zařízení vyžaduje pozornost mikrokontroléru,
    - musí dojít k přerušení výkonu běžícího programu,
    - příklady přerušení: externí – např. od tlačítka; interní – přetečení časovače, dokončení A/D převodu, ... (podrobněji o přerušeních později).

Způsoby určování čtení/zápisu

- Proces čtení nebo zápisu je řízen vždy dvěma signály. Existují dvě typické koncepce:
  - Intel zavedl použití signálů *RD* – read, *WR* – write (využívají také např. mikrokontroléry AVR) – vysoká úroveň definuje proces,
  - přístup do paměti trvá minimálně dva cykly.
  - druhý způsob používá např. Freescale, nebo řadiče LCD displeje: $R/\overline{W}$ – úroveň definuje směr komunikace, $\overline{E}$ – nízká úroveň aktivuje start procesu.





Paměť

- Paměť uchovává skupiny bitů (slova), která mohou reprezentovat:
  - instrukce, které mají být vykonány (=program) a také data, která mají být programem použita,
  - dočasné úložiště výsledků aritmeticko/logických operací.
- Zápis a čtení z paměti je řízeno signály $\overline{RD}$ a $\overline{WR}$ (příp. ekvivalenty), které generuje řídicí jednotka a konkrétní data jsou vybírána podle adresy, tj. indexu paměť. buňky.
- Zdrojem zapisovaných dat může být ALU nebo vstupní jednotka; vše je opět řízeno příslušnými signály, příp. adresou z řídicí jednotky.
- Čtená data mohou být přenesena do ALU nebo do výstupních obvodů.
- Číslicový systém může obsahovat interní i externí paměť.
- Paměti interní (z hlediska procesory):
  - vždy polovodičové,
  - slouží k uložení programu i dat aktuálně používané CPU; proto musí být nejrychlejší v mikropočítači; v opačném případě by proces čtení/zápisu omezoval výkon programu,
  - RAM, ROM, cache, ...
- Paměti externí:
  - slouží k uložení značného množství dat bez nutnosti stálého napájení; nemusí být extrémně rychlé,
  - ukládají se program i data, která nejsou aktuálně vyžadována CPU; pokud jsou tyto data vyžadována, provede se jejich přesun do interní paměti,
  - Flash, eeprom, DVD, ...

Vstupně/výstupní jednotka

- Vstupně/výstupní jednotka zajišťuje komunikaci s "okolním světem".
- Vstupní jednotka obsahuje obvody, které umožňují přenos externích dat (klávesnice, tlakový senzor, sériová linka, A/D převodník, ...) do vnitřní paměti procesoru, nebo do ALU.
- Výstupní jednotka obsahuje obvody, které zajišťují přenos dat a informací z interní paměti nebo ALU vně systém (LED, LCD, modem, ...).
- (Podrobná struktura I/O obvodů později.)

Obsah přednášky

Popis a použití mikrokontroléru, mikroprocesoru a mikropočítače

Realizace řídicí aplikace

Základní typy architektur v mikroprocesorové technice
- Von Neumann, Harvardská, CISC, RISC, VLIW, ...

Obecná bloková struktura mikrokontrolérů
- Aritmeticko/logická jednotka, centrální řídicí jednotka, paměti, vstupně/výstupní obvody

Ukázka programu v JSA pro AVR
- Práce s registry, aritmetické operace
- Ovládání vstupně/výstupního portu

Ukázka programu v JSA pro AVR: "Hello"

- Ukázka jednoduchého programu v jazyce symbolických adres (JSA, "=assembleru") pro mikrokontrolér AVR (např. ATmega16).
- Obecné informace:
  - prefix "0x" identifikuje hodnotu v hexadecimální soustavě, "0b" v binární,
  - označení R16, R17, ... reprezentuje 8bitové pracovní registry; AVR jich má celkem 32.
- Význam použitých instrukcí:
  - `ldi R16, 0x48`
    do registru R16 ulož osmibitovou konstantu 0x48 (v 10tkové soustavě hodnota 72),
  - `mov R18, R20`
    do registru R18 zkopíruj obsah registru R20,
  - `ADD R18, R17`
    k obsahu registru R18 přičti obsah registru R17,
  - `LSR R20`
    bitový posun doprava,
  - `RJMP loop`
    skoč na návěští/pozici v programu "loop".

```
1  .include <m16def_mod.inc>      ; vložit upravený popisný soubor pro ATmega16
2  ; "nahrání" dat
3      ldi    R16, 0x48           ; registr R16 = 48 (hex.), tj. 72 (dek.)
4      ldi    R17, 101            ; R17 = 101 (dek.)
5      ldi    R20, 0b00000111     ; R20 = 0b00000111 (bin.), tj. 7 (dek.)
6  ; početní a binární operace s daty
7      mov    R18, R20            ; R18 <- R20, tj. R18 = 7
8      add    R18, R17            ; R18 = R18 + R17 = 108
9      mov    R19, R18            ; R19 = R18 = 108
10     lsr    R20                 ; bitový posun doprava, tj. R20 = 3
11     add    R20, R18            ; R20 = R20 + R18 = 111
12 ; nekonečná smyčka
13 loop:
14     rjmp   loop                ; skok na návěští/místo v programu loop
```

Ukázka programu v JSA pro AVR: I/O port

- Použité registry:
  - R16 – jeden z 32 8bitových registrů pro obecné použití,
  - DDRB (PortB Data Direction Register) – řídicí registr; určuje směr komunikace jednotlivých pinů na portu B; 0⇔ vstupní pin, 1⇔ výstupní pin,
  - PORTB – řídicí registr, který obsahuje data pro výstupní piny portu B.
- Význam použitých instrukcí:
  - `ser temp`
    do pracovního registru temp ulož osmibitovou konstantu 0xFF (255 dek.),
  - `out DDRB, temp`
    do řídicího registru DDRB zkopíruj obsah pracovního registru temp; tj. v tomto případě bude celý port B výstupní,
  - `out PORTB, temp`
    do řídicího registru PORTB zkopíruj obsah pracovního registru temp,
  - `dec temp`
    od obsahu registru temp odečti jedničku (dekrementace),
  - `rjmp loop`
    skoč na návěstí loop.
- Velikost výsledného kódu je 12 B, tj. 0,1% paměti Flash mikrokontroléru ATmega16.

Ukázka programu v JSA a v jazyce C pro AVR: I/O port

```
; inicializace aplikace
.include <m16def_mod.inc>  ; vložit upravený popisný soubor pro ATmega16
.def temp = R16            ; definovat symbolický název pracovního reg. R16

; nastavení portu
    ser   temp             ; temp = 255
    out   DDRB, temp       ; směrový reg. DDRB = 255, tj. celý port B: výstup

; nekonečný binární čítač
loop:
    out   PORTB, temp      ; zápis hodnoty na výstupní port B
    dec   temp             ; dekrementace temp
    rjmp  loop             ; skok na návěští/místo v programu loop
```

```
/* inicializace aplikace */
#include "avr\io.h"          // vložení popisného/hlavičkového souboru AVR

/* hlavní funkce aplikace */
int main( void )
{
    DDRB  = 0xFF ;           // směrový reg. DDRB = 255, tj. celý port B: výstup
    PORTB = 0xFF ;           // zápis hodnoty na výstupní port B

    /* nekonečný binární čítač */
    while( 1 ){
        PORTB-- ;            // PORTB = PORTB - 1
    }

    /* formální ukončení funkce main */
    return( 1 ) ;            // návratová hodnota funkce main = 1
}
```

Ukázka programu v jazyce C pro AVR: I/O port

- Identická aplikace v jazyce C. Konkrétní typ mikrokontroléru AVR (např. ATmega16) je předán překladači jako jeden z parametrů.
- Hlavičkový soubor `io.h` obsahuje názvy a adresy všech registrů a periférií mikrokontrolérů AVR. Jména registrů jsou zde definovány VELKÝMI písmeny. Jazyk C je case-sensitive!
- Některé formy zápisu nekonečných (tj. vždy pravdivých) smyček v jazyce C:
  - `while( 1 ){ ... }`
  - `for( ;; ){ ... }`
- Zkrácený zápis aritmetické operace `PORTB--` je totožný se zápisem `PORTB = PORTB - 1`.
- Program je napsán pro překladač (=kompilátor) AVR-GCC! Pro jiné překladače může být syntakticky nesprávný.
- Velikost výsledného kódu je 184 B, tj. 1,1% paměti Flash mikrokontroléru ATmega16. Pozn.: Srovnejte s ASM.

Zdroje informací


VÁŇA, V.
Mikrokontroléry ATMEL AVR; popis procesorů a instrukční soubor.
Ben – technická literatura, Praha, 2003, ISBN 80-7300-083-0.


SPASOV, P.
Microcontroller Technology.
Pearson Prentice Hall, New Jersey (USA), 2004, ISBN 0-13-112984-8.


TOCCI, R.J.
Microprocessors and Microcomputers.
Prentice Hall, New Jersey (USA), 2003, ISBN 0-13-060904-8.


LEVENTHAL, L.; WALSH, C.
Microcomputer Experimentation with the Intel SDK-85.
Prentice-Hall, Inc., New Jersey (USA), 1980, ISBN 0-13-580860-X.


Atmel Corporation.
Atmel Microcontrollers, (září 2013).
http://www.atmel.com/products/microcontrollers/default.aspx