Cesta k prvnímu mikroprocesoru

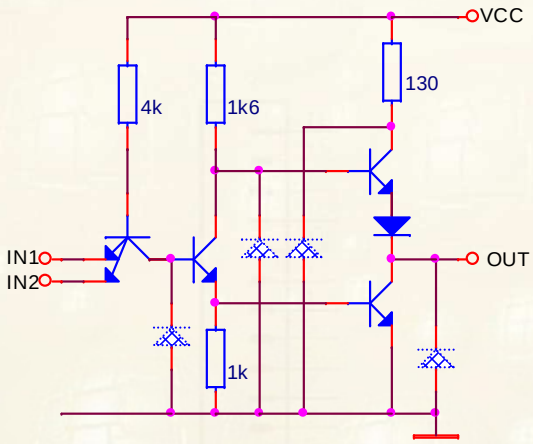
OD LOGICKÝCH OBVODŮ K μP – VLASTNOSTI VÝSTUPŮ LOG.OBVODŮ

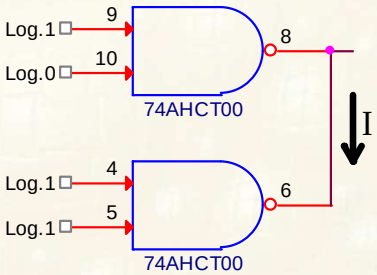
Logické obvody se vyrábí se třemi různými typy výstupů.

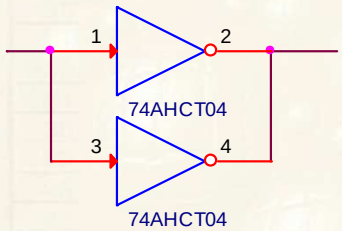
- Obvody s dvoustavovým výstupem
- Obvody s otevřeným kolektorem
- Obvody s třístavovým výstupem

Výstup obvodu s dvoustavovým výstupem

- Stav log.0 (Uvýst= typ. 0,1V až 0,2V, závisí na proudu)
- Stav log.1 (Uvýst= typ. 3,4V bipolární technologii, Ucc=5V, Uvýst= typ. Ucc-0,1V CMOS). Počet připojitelných vstupů stanoven větvením nebo kapacitou vstupů.

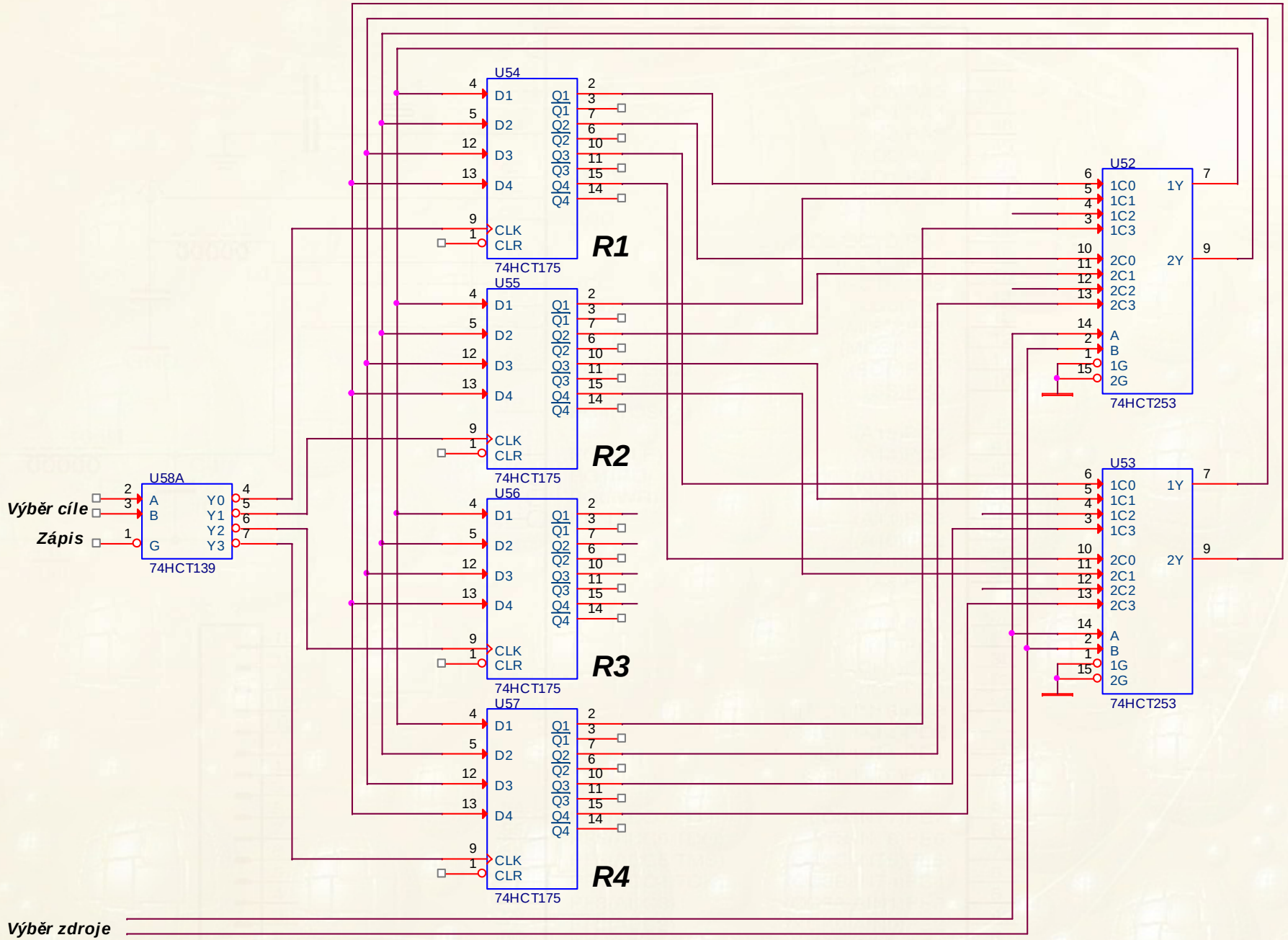




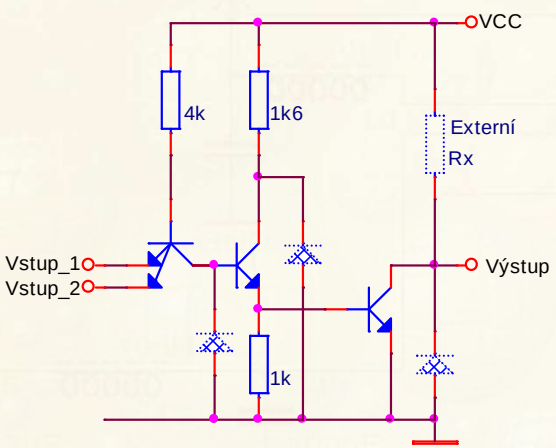


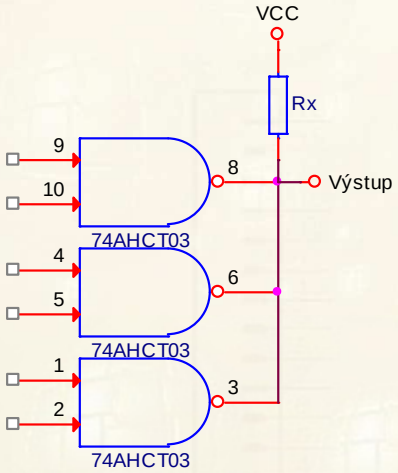
Okolo 1965 se realizovaly složité řadiče, u kterých bylo potřeba zajistit propojení řady registrů. ⇒ realizovatelné **multiplexory**

PROPOJENÍ LOGICKÝCH OBVODŮ (REGISTRŮ) S DVOUSTAVOVÝM VÝSTUPEM



- Propojení registrů multiplexory komplikuje zapojení zařízení.
- **Jiná cesta** – soustava propojovacích vodičů – **sběrnice**





- Ke sběrnici jsou připojeny vstupy registrů
- A jejich **výstupy s otevřeným kolektorem**
- Každý vodič sběrnice připojen přes odpor k napájení
  - Velikost odporu výrazně ovlivňuje rychlost přechodu z log.0 do log.1.
  - Problém připojení vstupu rychlého obvodu

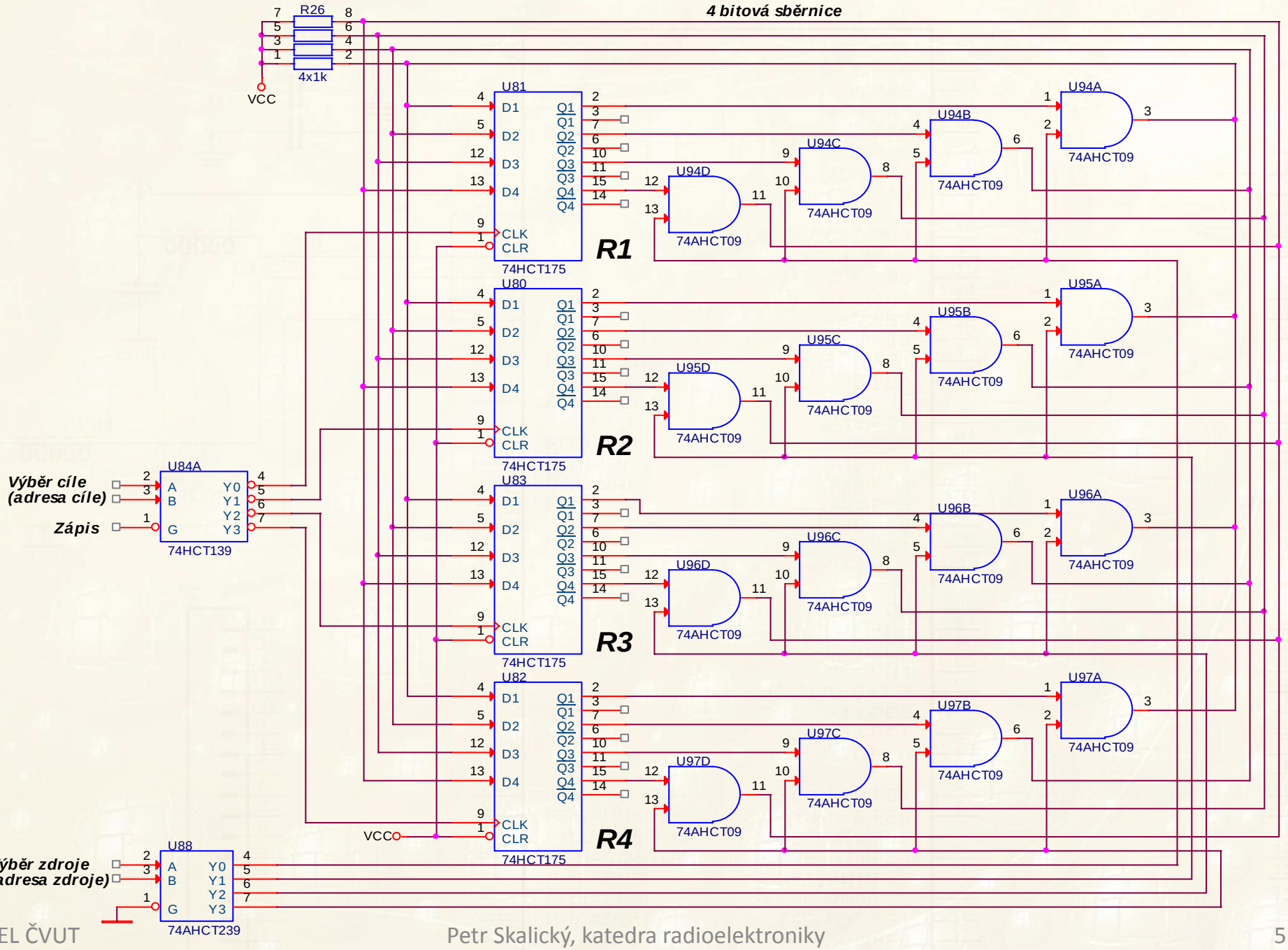
Obvody s otevřeným kolektorem s hradly NAND realizují funkci AND-OR-INVERT

$$Y = \overline{A.B}.\overline{C.D}.\overline{E.F} = \overline{\overline{A.B}.\overline{C.D}.\overline{E.F}} = \overline{A.B + C.D + E.F}$$

Označováno jako **montážní součin** nebo **součet** viz. logická funkce.

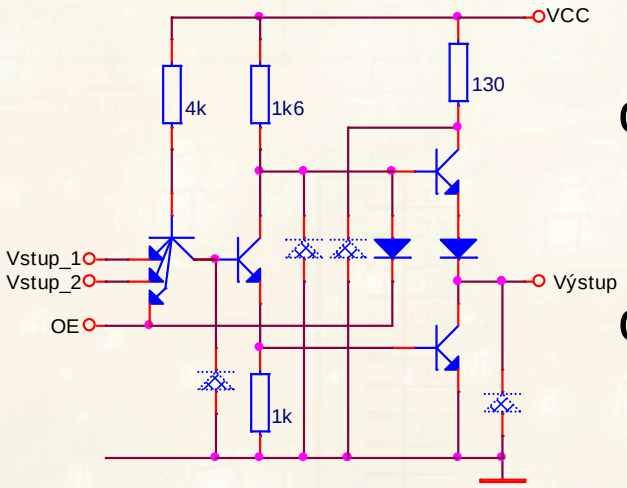
Současné využití - synchronizace různě rychlých obvodů s možností stanovení priority (I^2C), sloučení několika žádostí o přerušení.

PROPOJENÍ LOGICKÝCH OBVODŮ (REGISTRŮ) S OTEVŘENÝM KOLEKTOREM



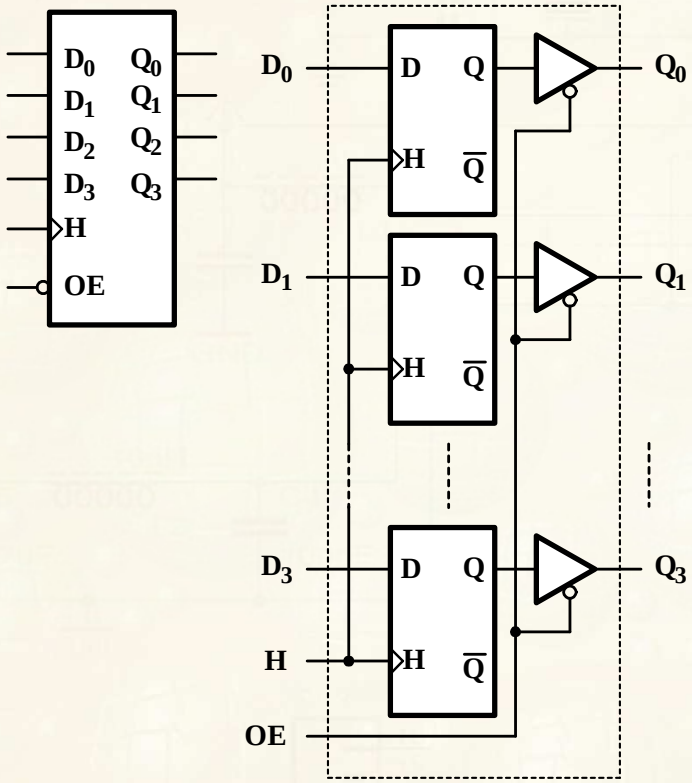
OD LOGICKÝCH OBVODŮ K μP – VLASTNOSTI VÝSTUPŮ LOG.OBVODŮ

- Otevřený kolektor - problémy s časovým zpožděním a výkonovým zatížením výstupů ⇒ nahrazen obvody s třístavovým výstupem.
- Dva stavy log.0, log.1 a ve třetím stavu jsou oba koncové tranzistory jsou uzavřené – **stav vysoké impedance**.
- Tento stav ovládá vstup označovaný OE (obvykle log.1 = výstupní vývod je ve stavu vysoké impedance - svodový odpor stovky k**Ω** až jednotky M**Ω**, kapacita jednotky pF).

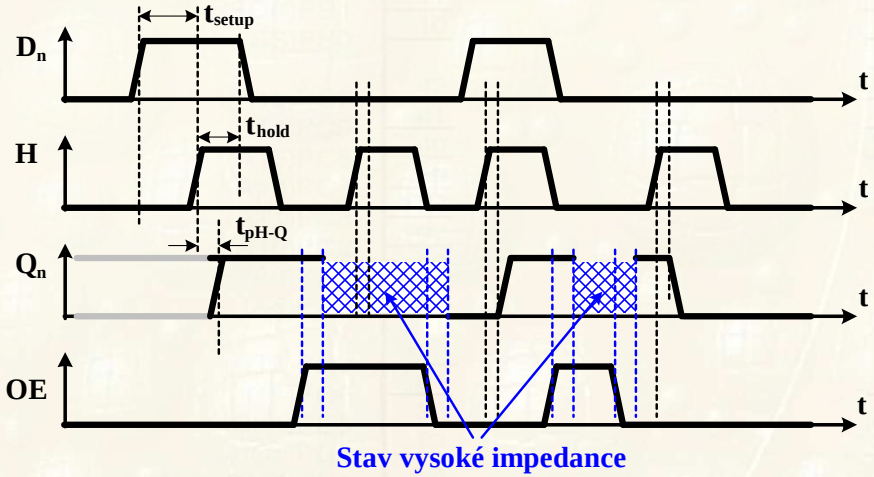


- **Výstupy** těchto obvodů mohou být spojeny
- **Pouze jeden obvod** smí být aktivní (výstup v log.1 nebo log.0)
- Ostatní připojené obvody budou mít **výstup v daném okamžiku** ve stavu vysoké impedance.
- Jsou-li k vodiči připojeny vstupy i výstupy je umožněna obousměrná komunikace označujeme - **datová sběrnice**.
- Zdroj nebo cíl přenosu - **adresová sběrnice**.

Registr, Latch - s třístavovým výstupem (paměť většího počtu bitů (2, 4, 6 nebo 8 bitů).

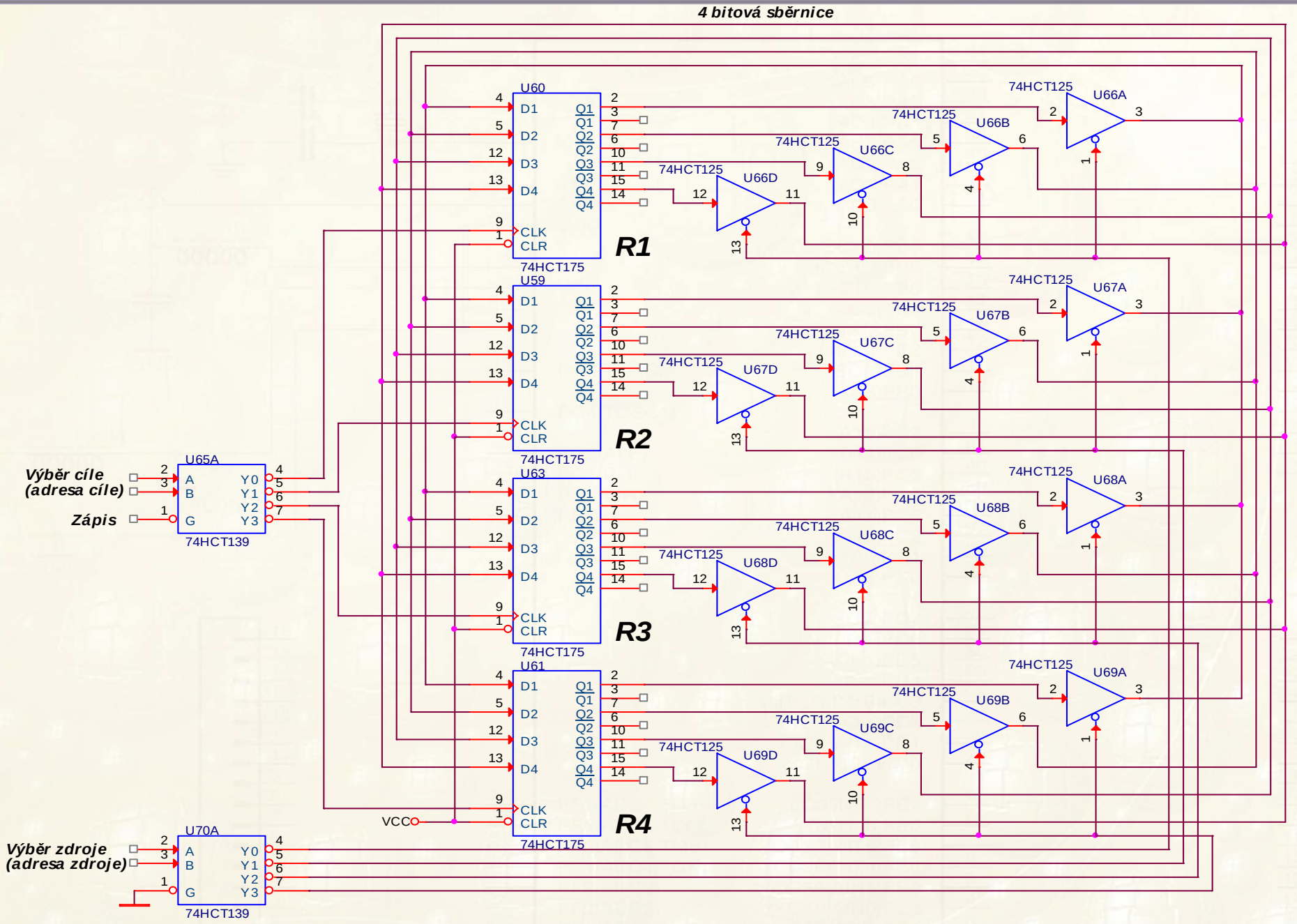


Často vybaven třístavovým výstupem. Hodinový vstupu umožňuje uložit vstupní informaci, která je na výstupech k dispozici pouze, je-li budič aktivován (log.0).

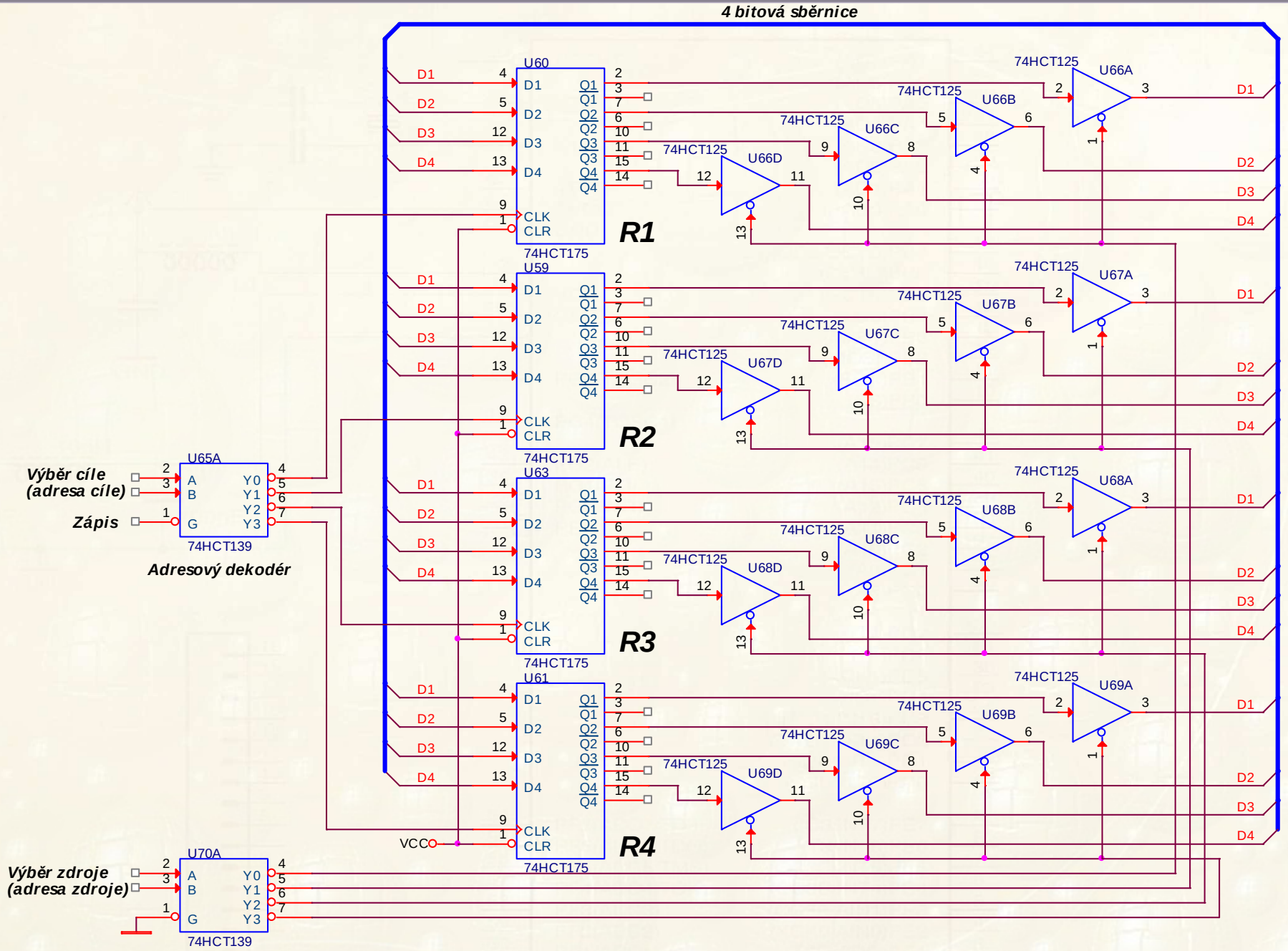


Použití: Vyrovnávací paměť – uchování krátce platné hodnoty, registr „pipeline“. **Výstupní brána** procesorového systému (OE=0). **Paměť** vnitřních proměnných LSO (OE=0). **Vstupní brána uP** systému (OE=funkce (adresy a řídícího signálu_čtení).

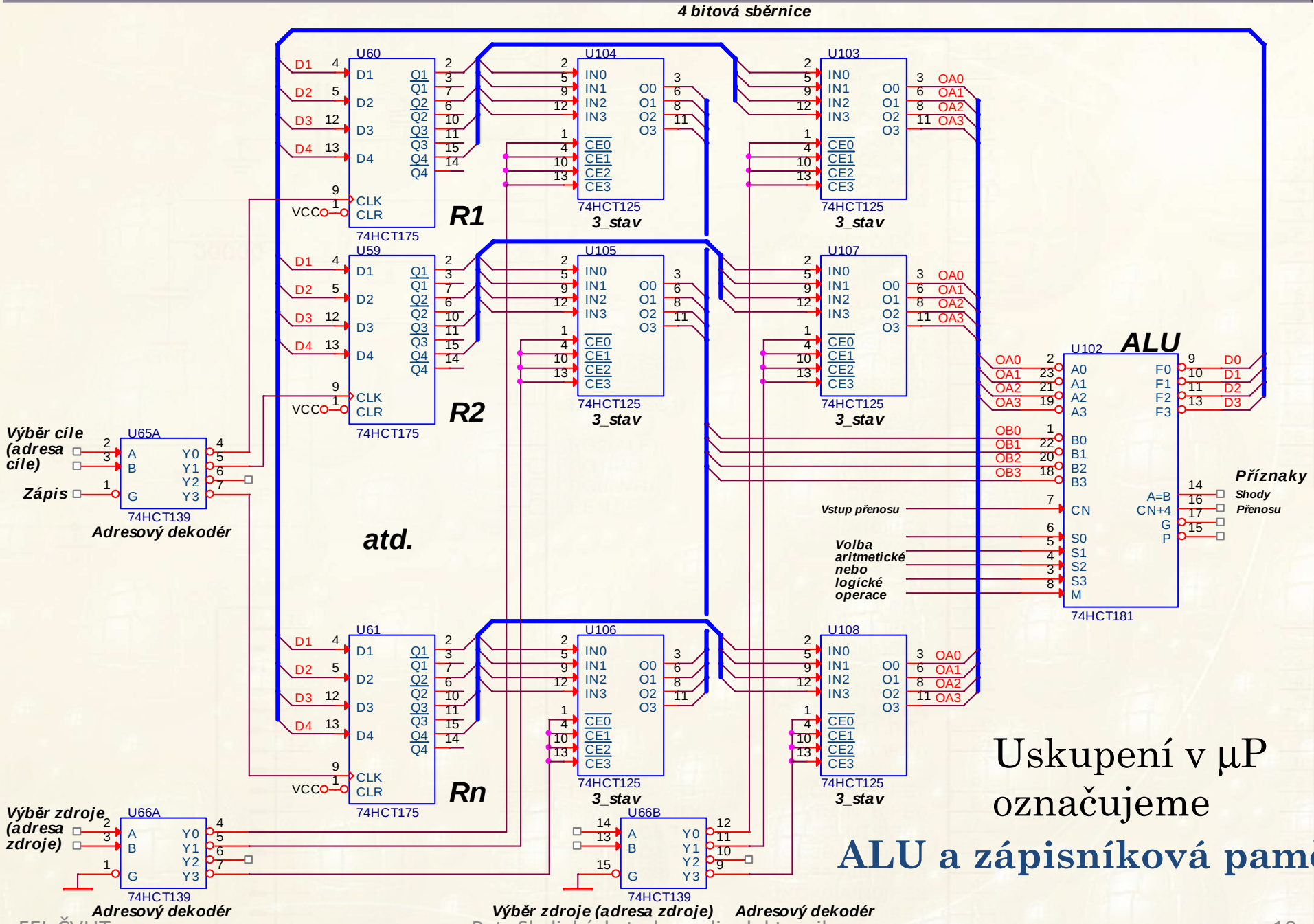
PROPOJENÍ LOGICKÝCH OBVODŮ (REGISTRŮ) S TŘÍSTAVOVÝM VÝSTUPEM



PROPOJENÍ LOGICKÝCH OBVODŮ (REGISTRŮ) S TŘÍSTAVOVÝM VÝSTUPEM



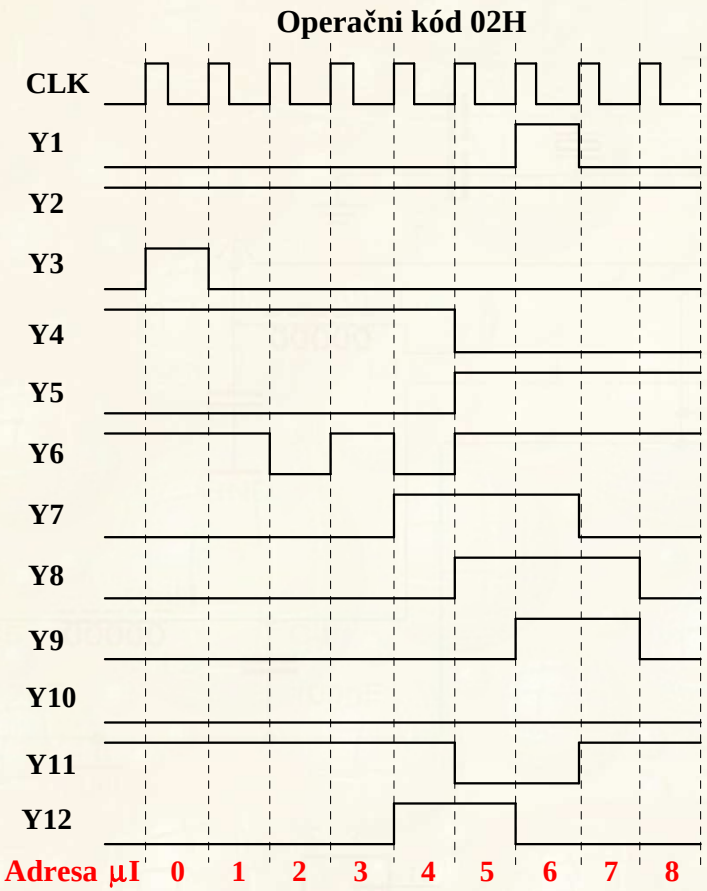
PROPOJENÍ REGISTRŮ S ARITMETICKO LOGICKOU JEDNOTKOU



Uskupení v µP označujeme

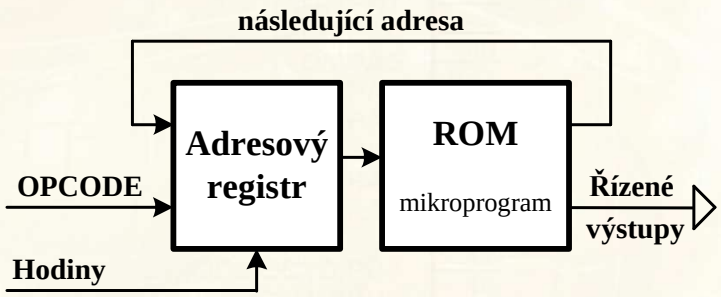
ALU a zápisníková paměť

MIKROPROGRAMOVATELNÝ ŘADIČ



Adresa	Obsah ROM	Adresa	Obsah ROM
020h	42E1h	025h	8F26h
021h	42A2h	026h	1F37h
022h	40A3h	027h	5B28h
023h	42A4h	028h	4329h
024h	C4A5h		

Adresa mikroinstrukce
Adresa následující mikroinstrukce



Adresa ROM = Operační kód + následující adresa.

Výstup ROM = **Generovaná sekvence** (mikroinstrukce) + následující adresa mikroprogramu.

Pro obrázek vlevo:

- Výstupy ROM = Q15÷Q4 signály Y12÷Y1, Q3÷Q0 následující adresa,.
- Adresa ROM = An÷A4 operační kód (zde 02h), A3÷A0 následující adresa.

OD OBVODOVÉHO ŘADIČE K MIKROPROGRAMOVÉMU

Modifikovaná konfigurace umožňuje:

❖ **Sekvenční čtení ⇒ řídící signály** pro realizaci operačního kódu a **sekvenci pro načtení dalšího operačního kódu (Fetch)**. *V dále uvedeném příkladu miniprocesoru zjednodušeno na impulz pro čítač instrukcí.*

❖ **Realizovat skok**

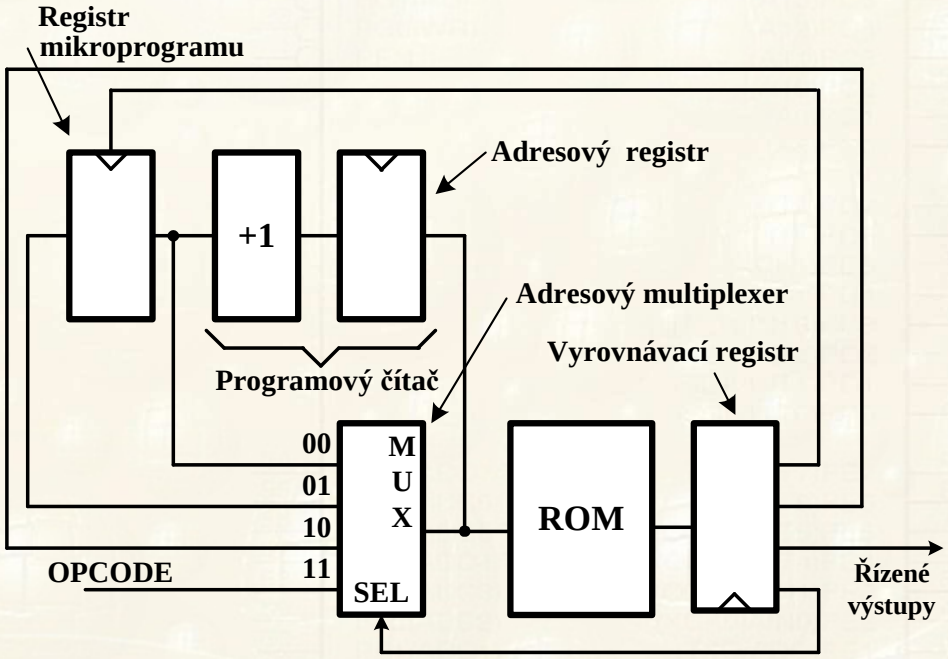
- Neumožňuje – **návrat z podprogramu**
- Neumožňuje – **podmíněné větvení**

U procesorů s velkým počtem instrukcí → zbytečně zvětšována velikost paměti ROM řadiče stejnou opakující se sekvencí (**Fetch**) ⇒ **degradace řadiče**.

Efektivnější řešení = skok do mikropodprogramu (například fetch). Návrat z mikroprogramu na adresu za mikroinstrukcí, která jej vyvolala.

Modifikace struktury

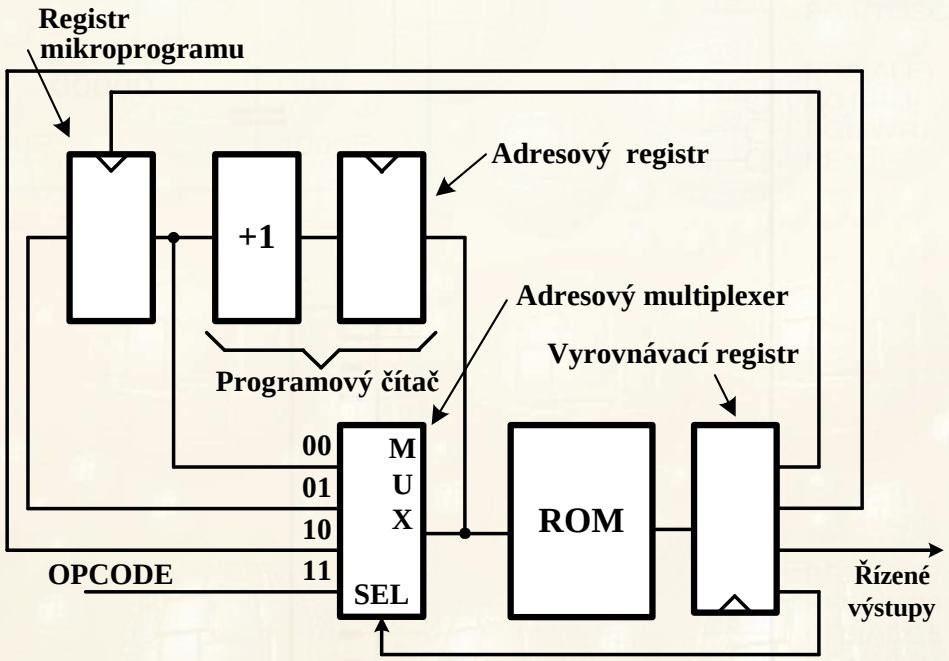
- **Programový čítač** (Program counter) → adresový registr doplněn o inkrementující sčítačku.
- **Registr** mikroprogramu → **jednoúrovňový zásobník** pro uchování **návratové adresy** (návrat z mikropodprogramu).
- **Multiplexer** – výběr následující adresy.
- **Vyrovnávací registr** – synchronizace všech signálů a eliminace zpoždění způsobeného **dobou vybavení** z ROM.



Adresa na výstupu multiplexoru

- **SEL=00 - Programový čítač** (sekvenční zpracování μprogramu)
- **SEL=01 - Registru mikroprogramu** (návrat z podprogramu)
- **SEL=10 - Vyrovnávací registr** (skok, volání podprogramu)
- **SEL=11 - Nový operační kód**

Jednotka umožňuje



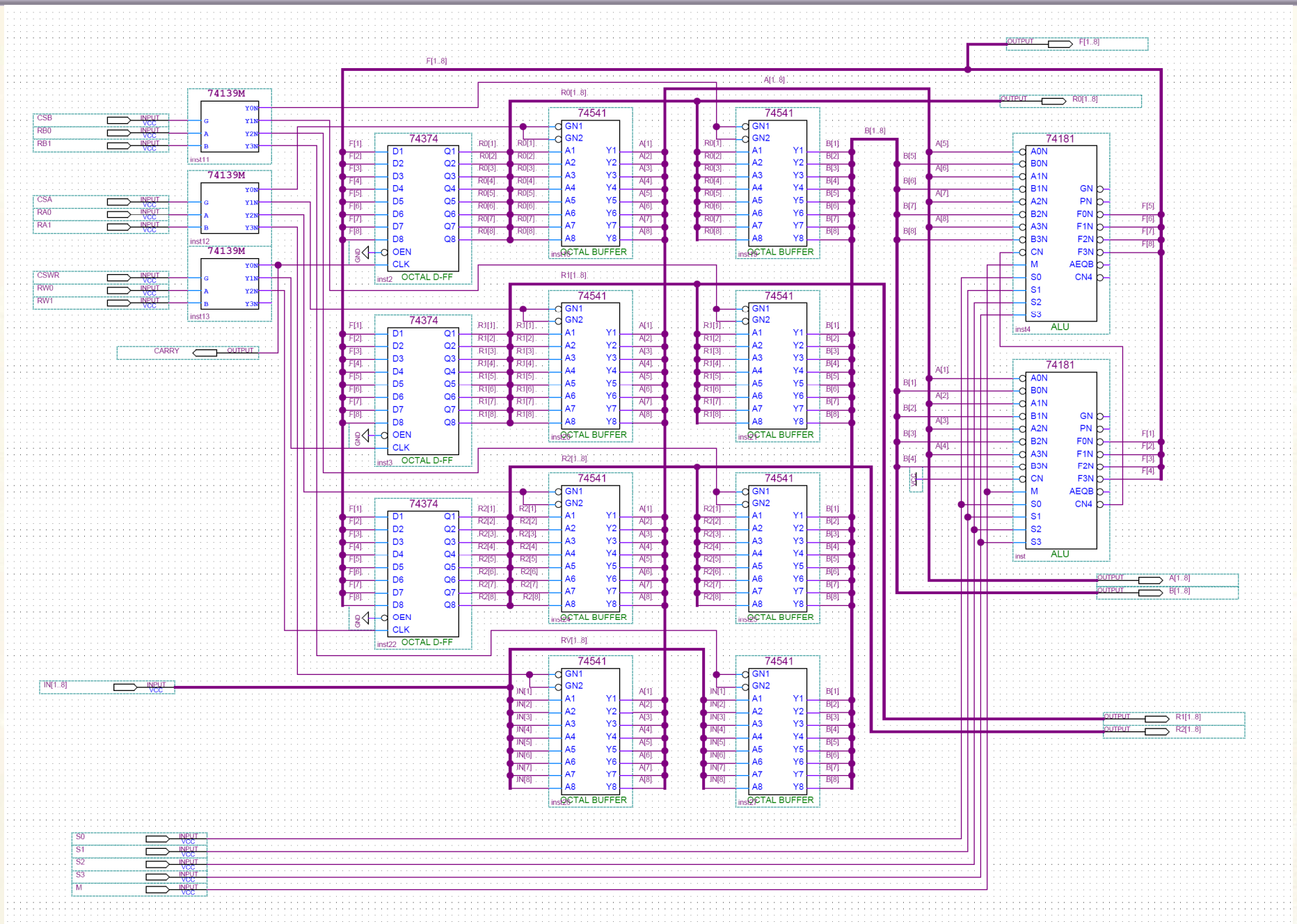
- **Sekvenční čtení**
- **Volání podprogramu** (vždy jen jednoho)
- **Skok na libovolnou adresu**
- **Zavolání podprogramu**
- **Návrat z podprogramu.**

Změnou ovládání MUX lze zavést **podmíněné větvení**. Pro řadič nutno přidat **stavové příznaky** jako vstupní signály.

Jednoduchý

miniprocesor

8-BITOVÉ PROPOJENÍ REGISTRŮ S ALU 74181 SE ZÁPISNÍKOVOU PAMĚTÍ

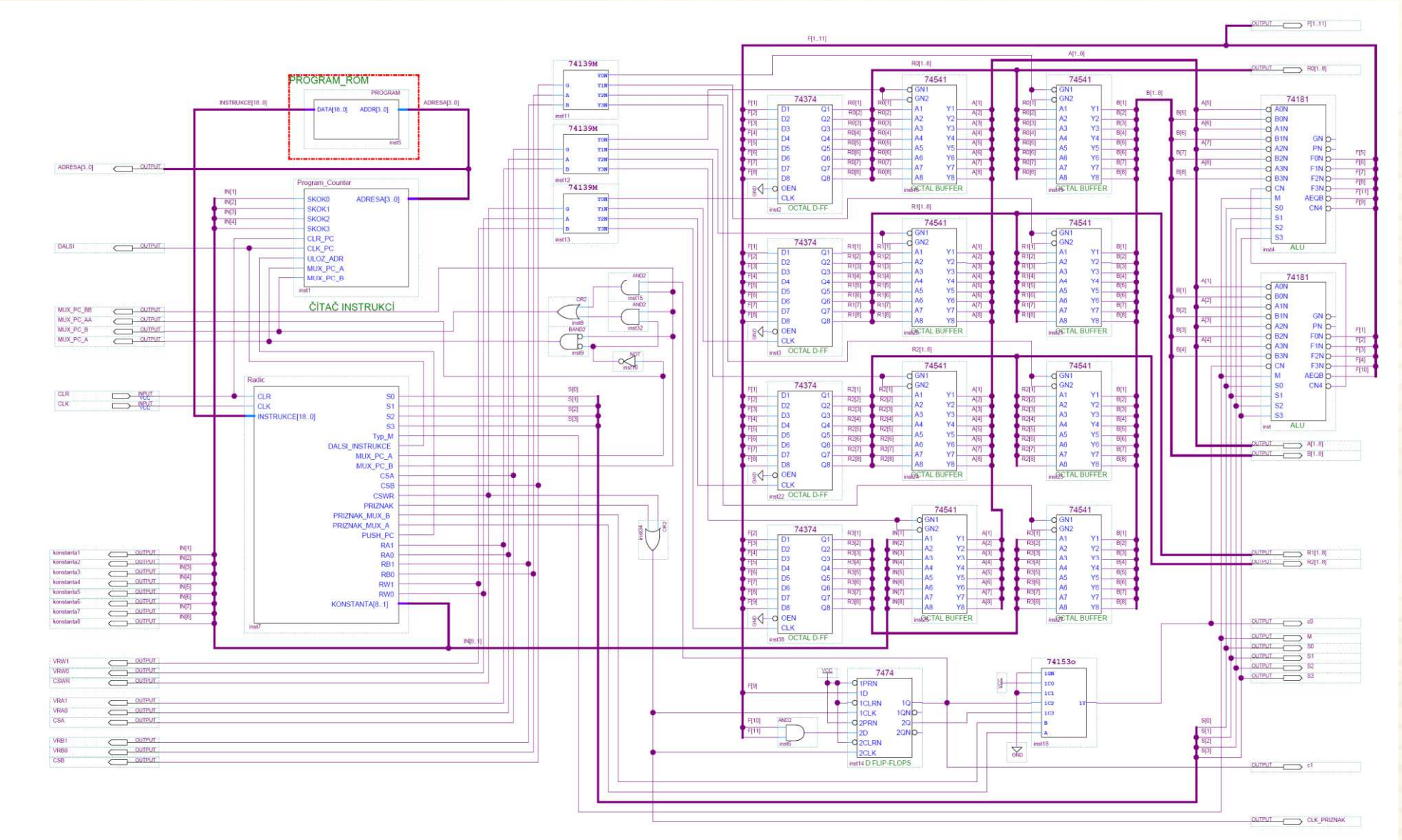


LOGICKÉ A ARITMETICKÉ OPERACE ALU 74181

TABLE 2

SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	$\overline{C_n}$ = H (no carry)	$\overline{C_n}$ = L (with carry)
L	L	L	L	$F = \overline{A}$	F = A	F = A PLUS 1
L	L	L	H	$F = \overline{A + B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	$F = \overline{A}B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	L	L	$F = \overline{AB}$	$F = A$ PLUS $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS 1
L	H	L	H	$F = \overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$ PLUS 1
L	H	H	L	$F = A \oplus B$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
H	L	L	L	$F = \overline{A} + B$	F = A PLUS AB	F = A PLUS AB PLUS 1
H	L	L	H	$F = \overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS 1
H	L	H	H	F = AB	F = AB MINUS 1	F = AB
H	H	L	L	F = 1	F = A PLUS A†	F = A PLUS A PLUS 1
H	H	L	H	$F = A + \overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

JEDNODUCHÝ MINIPROCESOR VARIANTA 3

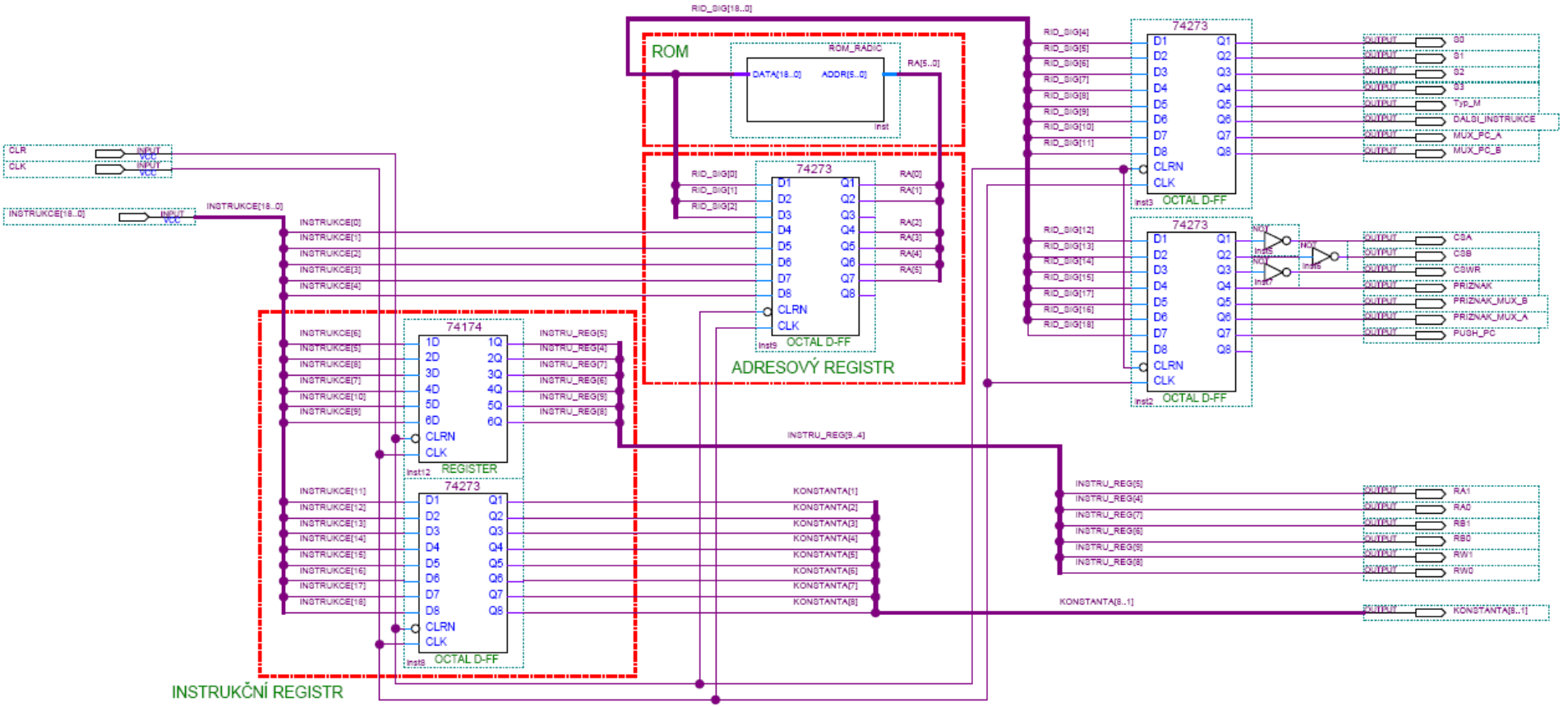


REGISTRY S ALU

- **Registry (zápisníková paměť, banky registrů)** - soubor obecných a speciálních registrů s rychlým přístupem k ALU. Slouží k uložení operandů, mezivýsledků, mohou představovat ukazatel nebo mít funkci určenou výrobcem. V jazyce C slouží k vykonání operací a trvale neuchovávají proměnné. Registry mohou být přístupné :
  - **Implicitně** - instrukce obsahuje informaci o použitém registru
  - **Paměťově** – v instrukci je obsažena adresa registru nebo odkaz na jejího ukazatele.
- **Instrukce** – obsahuje operačním kód instrukce doplněný dalšími potřebnými informacemi k jejímu vykonání tj.
  - Registry
  - Konstanty
  - Adresy
  - Ukazatele na adresu

ŘADIČ PRO 8-BITOVÉ PROPOJENÍ REGISTRŮ S ALU 74181

- **Instrukce** – v tomto ukázkovém příkladu má pevný formát skládající se z 5 bitů pro operační kód, 6 bitů určujících RegA, RegB a cílový RegW. Zbývajících 8 bitů určuje konstantu, adresu skoku nebo volaného podprogramu (5 bitů). Informace o registru a konstantě nemusí být využita.



MIKROPROGRAMOVATELNÝ ŘADIČ PRO REGISTRY A ALU 74181

	Adresa ROM	Význam obsahu paměti ROM
Operační kód 00	00h až 03h	Mikroprogram pro instrukci NOP
Operační kód 01	04h až 07h	Mikroprogram ----- pro zatím nevyužitou instrukci
	atd.	atd.
Operační kód 09	24h až 27Fh	Mikroprogram pro instrukci LOAD Rn, RAn, RBn
Operační kód 10	28h až 2Bh	Mikroprogram pro instrukci XOR Rn, RAn, RBn
Operační kód 11	2Ch až 2Fh	Mikroprogram pro instrukci ADD Rn, RAn, RBn
	atd.	atd.

-- ADRESA ŘADIČE - 6 bitů v pořadí OP3 OP2 OP1 OP0 A1 A0

-- DATA - 19 bitů v pořadí PUSH MUX_PRIZNAK_BA,PRIZNAK CSWR,CSB,CSA,
MUX_PC_BA dalšíINS, typ_M, funkce(S3,S2,S1,S0), A3,A2,A1,A0

-- NAČTENÍ KONSTANTY URČENÉ IN8-IN1

ROM_WORD'('0'&"001"&"000"&"00"&'0'&'0'&"0000"&"0001"), -- OPkod=1001 ADR=01001_00 36

ROM_WORD'('0'&"001"&"001"&"00"&'0'&'0'&"0000"&"0010"), -- OPkod=1001 ADR=01001_01 37

ROM_WORD'('0'&"001"&"101"&"00"&'0'&'0'&"0000"&"0011"), -- OPkod=1001 ADR=01001_10 38

ROM_WORD'('0'&"001"&"001"&"00"&'1'&'0'&"0000"&"0000"), -- OPkod=1001 ADR=01001_11 39

-- SOUČET RW=RA plus RB nebo RW=RB plus IN8-IN1

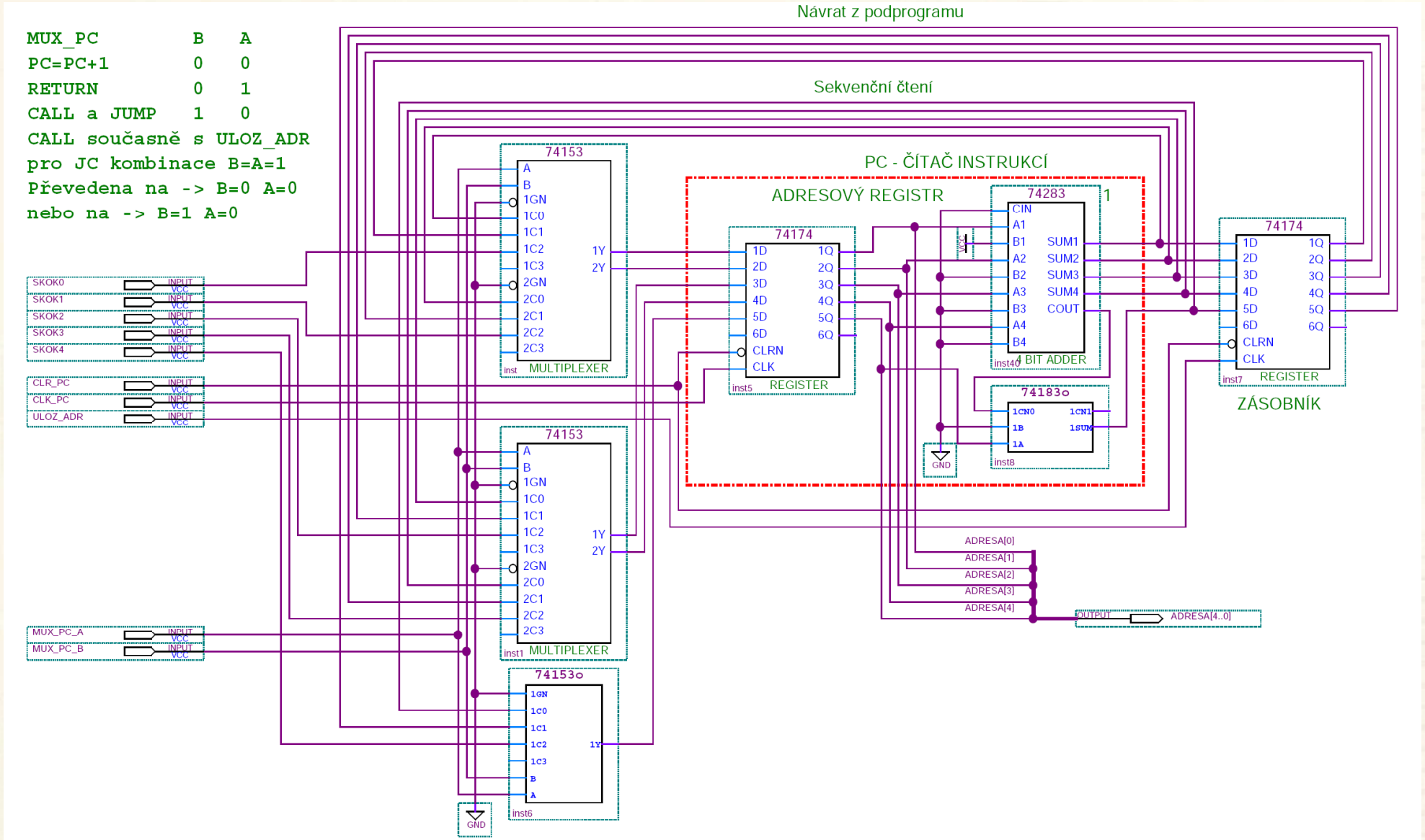
ROM_WORD'('0'&"000"&"000"&"00"&'0'&'0'&"1001"&"0001"), -- OPkod=1011 ADR=01011_00 44

ROM_WORD'('0'&"000"&"011"&"00"&'0'&'0'&"1001"&"0010"), -- OPkod=1011 ADR=01011_01 45

ROM_WORD'('0'&"000"&"111"&"00"&'0'&'0'&"1001"&"0011"), -- OPkod=1011 ADR=01011_10 46

ROM_WORD'('0'&"000"&"011"&"00"&'1'&'0'&"1001"&"0000"), -- OPkod=1011 ADR=01011_11 47

PROGRAMOVÝ ČÍTAČ varianta 3



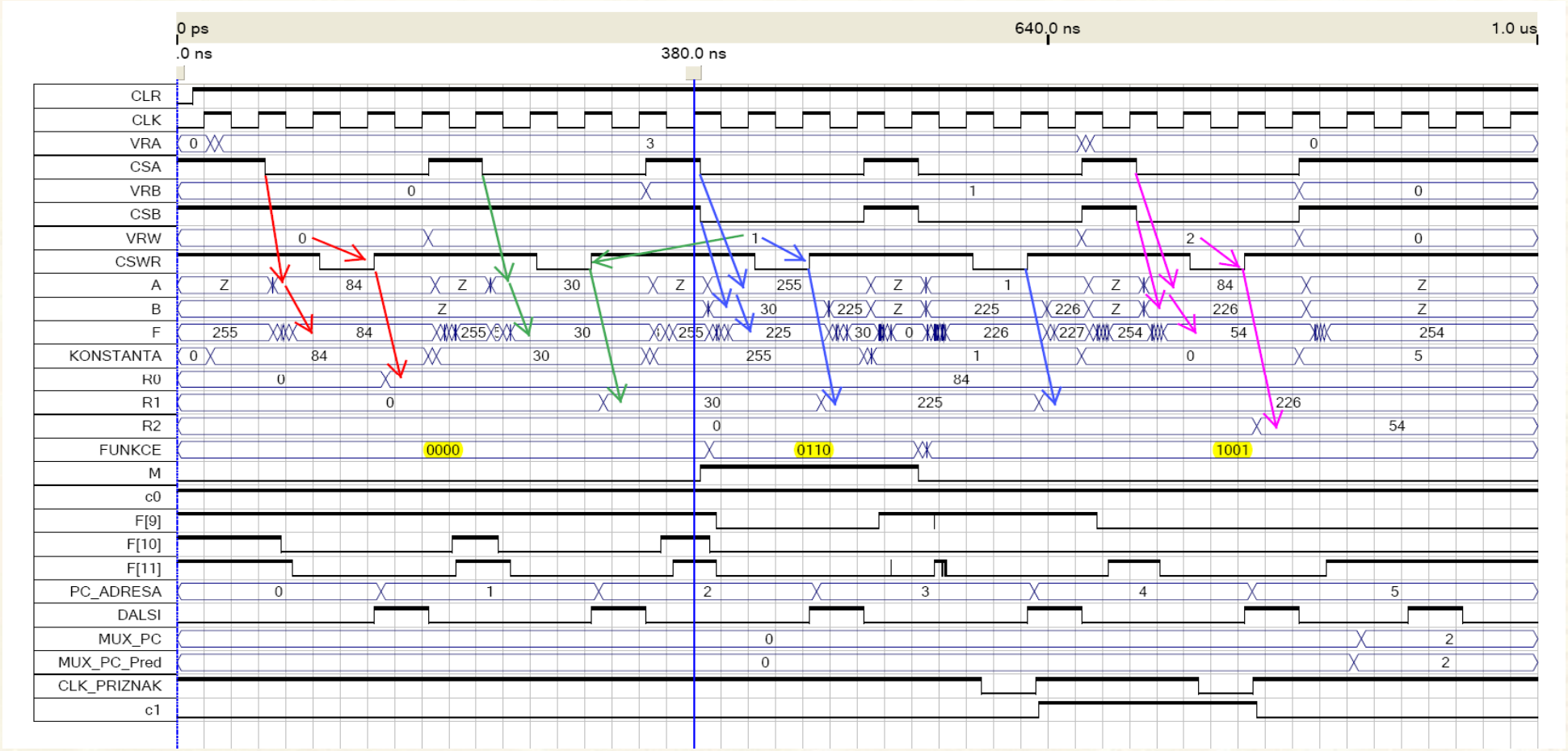
OBSAH PROGRAMOVÉ PAMĚTI A JEHO VÝZNAM PRO REGISTRY A ALU

```
-- ADRESA    - 4 bity v pořadí A3 A2 A1 A0 - MAXIMÁLNĚ 16 INSTRUKCÍ
-- DATA      - 19 bitů v pořadí   IN8,IN7,IN6,IN5,IN4,IN3,IN2,IN1,
                                  RW1,RW0, RA1,RA0, OP4,OP3,OP2,OP1,OP0
-- Kde  OPx-operační kód, RWx - cílový registr,  RAx - zdrojový registr pro sběrnici A
--      RBx - zdrojový registr pro sběrnici B, IN8-IN1 vstup konstanty, IN3-IN1 adresa skoku, CALL

constant PROGRAM: ROM_TABLE := ROM_TABLE'(
-- ADRESY ŘADIČE = OP+rr (kroky mikroprogramovatelného řadiče= počet hod.cyklů na instrukci)
--                    IN       RW   RB   RA      OP                                  ADR. ŘADIČE
ROM_WORD'("01010100"&"00"&"00"&"11"&"01001"),   -- PC= 0   LOAD R0,# 0x54      01001rr
ROM_WORD'("00011110"&"01"&"00"&"11"&"01001"),   -- PC= 1   LOAD R1,# 0x1E      01001rr
ROM_WORD'("11111111"&"01"&"01"&"11"&"01010"),   -- PC= 2   XOR R1,R1,#0xFF     01010rr
ROM_WORD'("00000001"&"01"&"01"&"11"&"01011"),   -- PC= 3   ADD R1,R1,#0x01     01011rr
ROM_WORD'("00000000"&"10"&"01"&"00"&"01011"),   -- PC= 4   ADD R2,R0,R1        01011rr
ROM_WORD'("00000101"&"00"&"00"&"00"&"01110"),   -- PC= 5   JMP 101             01110rr
ROM_WORD'("00000000"&"00"&"00"&"00"&"00000"));  -- PC= 6    -- (Skok na adresu 5)
begin      DATA <= PROGRAM(ADDR);  -- Read from the ROM
end PROGRAM_a_PC;
```

REALIZACE ROZDÍLU ČÍSEL 81 A 30 V PROPOJENÍ REGISTRŮ S ALU

Načtení prvého operandu do registru R0	LOAD R0,#0x54
Načtení druhého operandu do registru R1	LOAD R1,#0x1E
Vytvoření jednotkového doplňku z registru R1	XOR R1,R1,#0xFF
Přičtení jedničky k R1 = dvojkový doplněk	ADD R1,R1,#0x01
Sečtení registru R0 a R1, výsledek do R2	ADD R2,R0,R1
Ukončení, skok na adresu 101	JMP 5



VZNIK PRVNÍHO MIKROPROCESORU

Stolní kalkulačky - výpočetní jádro tvořilo propojení registrů s aritmetickou jednotkou spolu s řídící jednotkou. Operace realizovány

- S Nx4 bity kaskádně řazenými ALU (74181).
- Sériově s využitím jedné 4-bitové ALU

⇒ Obě řešení potřebovala řadu podpůrných obvodů a **jednoúčelový řadič zajišťující všechny** výpočty (**funkce**) kalkulačky.

Myšlenka – vytvořme instrukce (operace ALU), z kterých složíme **požadovaný výpočet. Instrukce uložené v programové paměti budou postupně čteny a vykonávány**. **Řadič bude jednoúčelový pouze s ohledem na realizaci jednotlivých instrukcí nikoliv k funkci celého zařízení**.

Řadič se zjednodušil na generování signálů pro čtení instrukce, dekódování, načtení operandů, vykonání instrukce a uložení výsledku. **Je nezávislý** na potřebném výpočtu a funkcích celého zařízení. Takový obvod byl vytvořen firmou **Intel** v rámci vývoje do nového stolního kalkulátoru Busicom 141-FP a dostal označení **I4004**.

VÝVOJ MIKROPROCESORŮ

- 1971 – (15.prosince) **I4004** první procesor (4-bitový) navržen pro snazší realizaci stolní kalkulačky. CPU se skládalo z 4 integrovaných obvodů a obsahovalo cca 8000 tranzistorů. Příliš neuspěl stejně jako I8008 a Intel zpětně odkoupil realizovaný vývoj .
- 1974 uveden legendární procesor **I8080**, který dosáhl masového využití nejen ve vojenských a průmyslových aplikacích, tak i prvních osobních počítačích. CPU se skládala ze 3 obvodů a celý procesorový systém bylo možné realizovat v 8 a více integrovaných obvodech.
- 1976 první jednočipový procesor 8048 předchůdce řady **8051** (1980).
- 1979 uveden první **signálový procesor I2920**. Naznačil nový možný směr vývoje procesorů. Jeho následovníci již mají výrazně odlišnou architekturu (modifikovaná Harvardská struktura, dvě sběrnice, jednotka MAC).
- 1982 - procesor **80286**, první specializovaný procesor pro osobní počítače.
- Koncem 70 let vznikají první vývojové systémy, **překladače assembleru**, **aritmetické knihovny** s pohyblivou čárkou a **operační systémy CP/M** (1982 předchůdce systému **DOS**)
- 2001 – **NIOS** (Altera) první komerční implementace procesoru v programovatelném poli **LCA** (FPGA)

Mikroprocesor

Mikroprocesor

- **sekvenční obvod s pevně nastaveným algoritmem**
- **postupně čte jednu nebo více instrukcí uživatelského programu**
- **přečtené instrukce zpracovává postupně nebo současně (paralelně)**
- Jako každý sekvenční obvod, musí být po připojení k napájení uveden do **počátečního stavu**, aby algoritmus začal z definovaného stavu.
- Počáteční stav je určen **čítačem instrukcí** (obvykle nulový) a stavem některých **důležitých registrů** (přerušovacího systému, interních periferií, vstupně-výstupních obvodů, atd.) před zahájením zpracování uloženého programu.
- Počáteční stav čítače instrukcí PC (není-li sčítán s kódovým segmentem) **představuje adresu**, z které se **bude číst první instrukce programu**.

ALGORITMUS PROCESORU

- **Řadič** zajistí přivedení hodnoty PC na **adresovou sběrnici**
- **Řadič** vygeneruje řídící signál pro čtení z operační paměti (Von Neuman) nebo programové paměti (harvardská struktura)
- Po vybavení hodnoty z paměti plus zpoždění na sběrnici se na datové sběrnici objeví **operační kód** první instrukce realizovaného programu.
- **Operační kód** je přenesen a uložen do **instrukčního registru** a analyzován v dekodéru instrukcí.
- Je-li přečtená instrukce je jednobytová nebo jednoslovná, jsou řadičem generovány řídící signály potřebné k jejímu vykonání.
- Jsou-li k operačnímu kódu potřeba **data** nebo **adresa** je z následující adresy (PC=PC+1) přečten další byte nebo slovo.
- **Nejsou-li všechny části instrukce přečteny, pokračuje procesor v jejich čtení.**

ALGORITMUS PROCESORU

- Je-li k dispozici celá instrukce, potom jsou **řadičem** generovány signály potřebné k **jejímu vykonání**.
- Čítač instrukcí je inkrementován a z **následující adresy je přečtena hodnota, kterou procesor bude chápat jako operační kód.** (*Vyjma případu, kdy procesor je schopen zjistit, že přečtená hodnota není operačním kódem a zareagovat na tuto situaci*).
- **Podle uvedeného algoritmu procesor postupuje od počáteční adresy, nezávisle na tom, zda čtená data jsou smysluplná nebo ne, zda jsou vůbec čtena z nějaké paměti.**

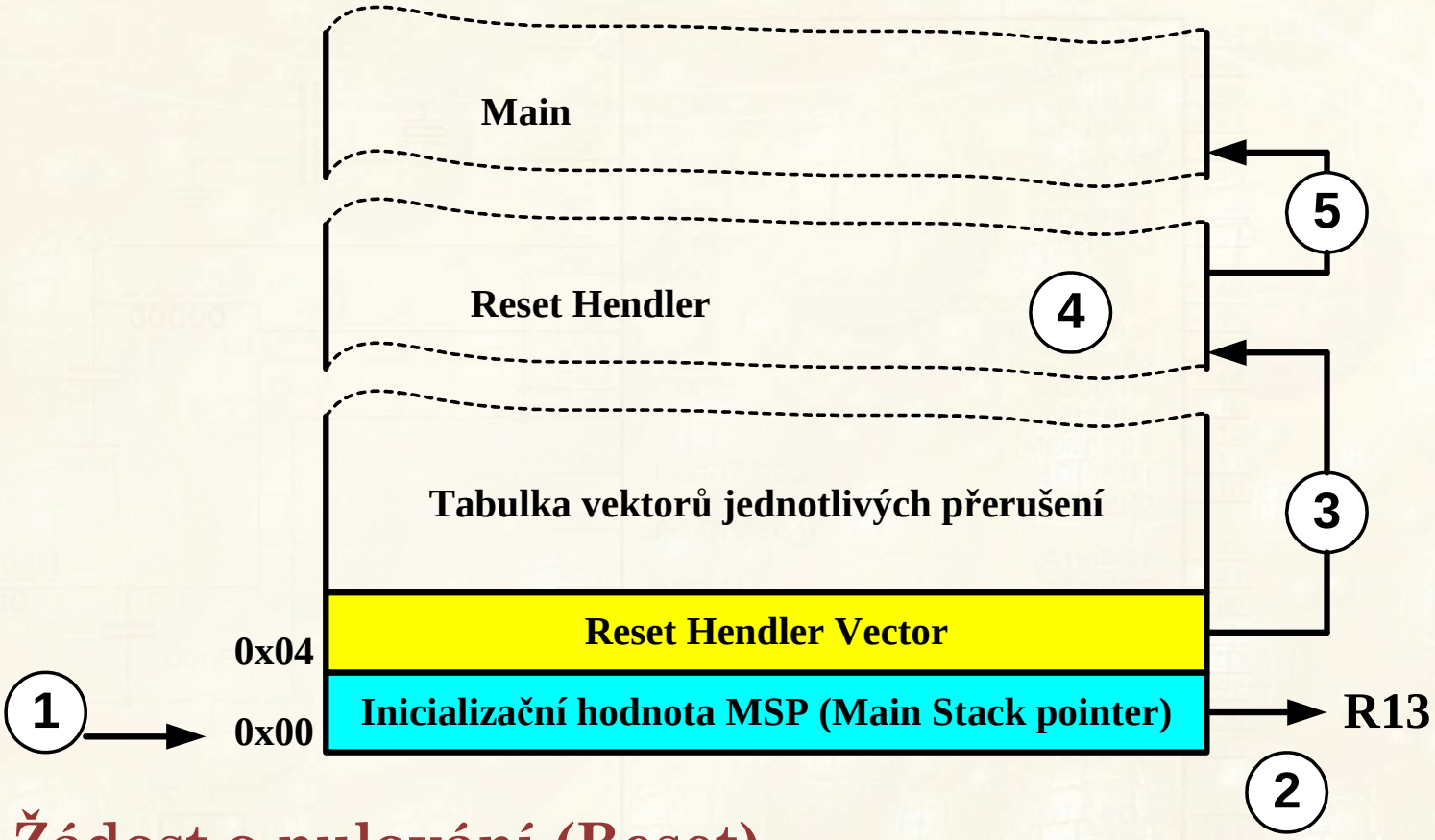
$\Rightarrow$

Uživatelský program (aplikace, operační systém, atd.) musí představovat nekonečnou smyčku.

Program **musí** být ukončen instrukcí skoku nebo instrukcí, která je součástí nekonečného cyklu.

- **Špatně ukončený program**
  - Může **ošetřit** operační nebo vývojový systém chybovým hlášením a návratem do systému.
  - V aplikaci bez operačního nebo vývojového systému **pokračuje** procesor **podle svého algoritmu**.
- **Chování procesoru pak závisí na mnoha okolnostech**.
  - Může proběhnout celou paměť a vrátit se na počáteční adresu.
  - Může přepsat část programové (je-li to možné) nebo datové paměti.
  - Může vykonávat **nekonečnou smyčku**, která vznikne skokem na byte (slovo), které v paměti programu není operačním kódem, ale **procesor ho tak bude chápat**. Tím dochází **k jiné interpretaci** obsahu paměti (obvykle zcela nesmyslné). Hovoříme pak o **desinterpretaci programu**, která může být též způsobena i dalšími vlivy (napájení, rušení, EMC, PCB, nevhodné obvody, atd.).

START PROCESORU ARM



1. Žádost o nulování (Reset)
2. Naplnění ukazatele MSP inicializační hodnotou
3. Přečtení adresy Reset Handler z adresy 0x04
4. Zpracování Reset Handler v Thread Módu
5. Přesměrování na Main program

♣ **Programový čítač** – obsahuje adresu, z které se čte instrukce nebo data/adresa vykonávané instrukce. Jeho hodnota obvykle není přímo přístupná. U ARM je PC přístupný (registr **R15**).

♣ **Aritmeticko-logická jednotka (ALU)**

- Realizuje aritmetické a logické operace
- Nastavuje příznakové bity a je jejich stavem ovlivňována
- Operace: **aritmetické sčítání, jednotkový doplněk (inverze bitů), logický součin (identifikace hodnoty bitu) a podmíněné větvení na základě hodnoty nebo bitu** tvoří nezbytné **minimum** pro realizaci celé aritmetiky s pevnou a pohyblivou čárkou.
- **Má-li ALU větší počet bitů**, než do ní vstupující operandy ⇒ **musí být známo, zda je operand se znaménkem nebo bez znaménka** (informace se nastavuje nějakým bitem).

♣ **Instrukční registr** – interní registr pro dočasné uchování přečtené instrukce pro zpracování v **dekodéru instrukcí** a **řadiči procesoru.**

♣ **Řadič - generuje posloupnosti signálů** nutné k vykonání dané instrukce. Činnost řadiče můžeme rozdělit do těchto fází:

- **Načtení instrukce -** (fetch) a její uložení do registru instrukcí
- **Dekódování instrukce -** a inkrementace PC. Na základě dekódování je rozhodnuto o dalším čtení z programové paměti nebo o zahájení vykonání instrukce.
- **Příprava operandů -** přenos operandů instrukce, které mohou být uloženy v registrech, datové paměti nebo jako konstanta.
- **Vykonání instrukce -** poslední fáze jejího zpracování a uložení výsledku do registru, datové paměti nebo V/V brány.

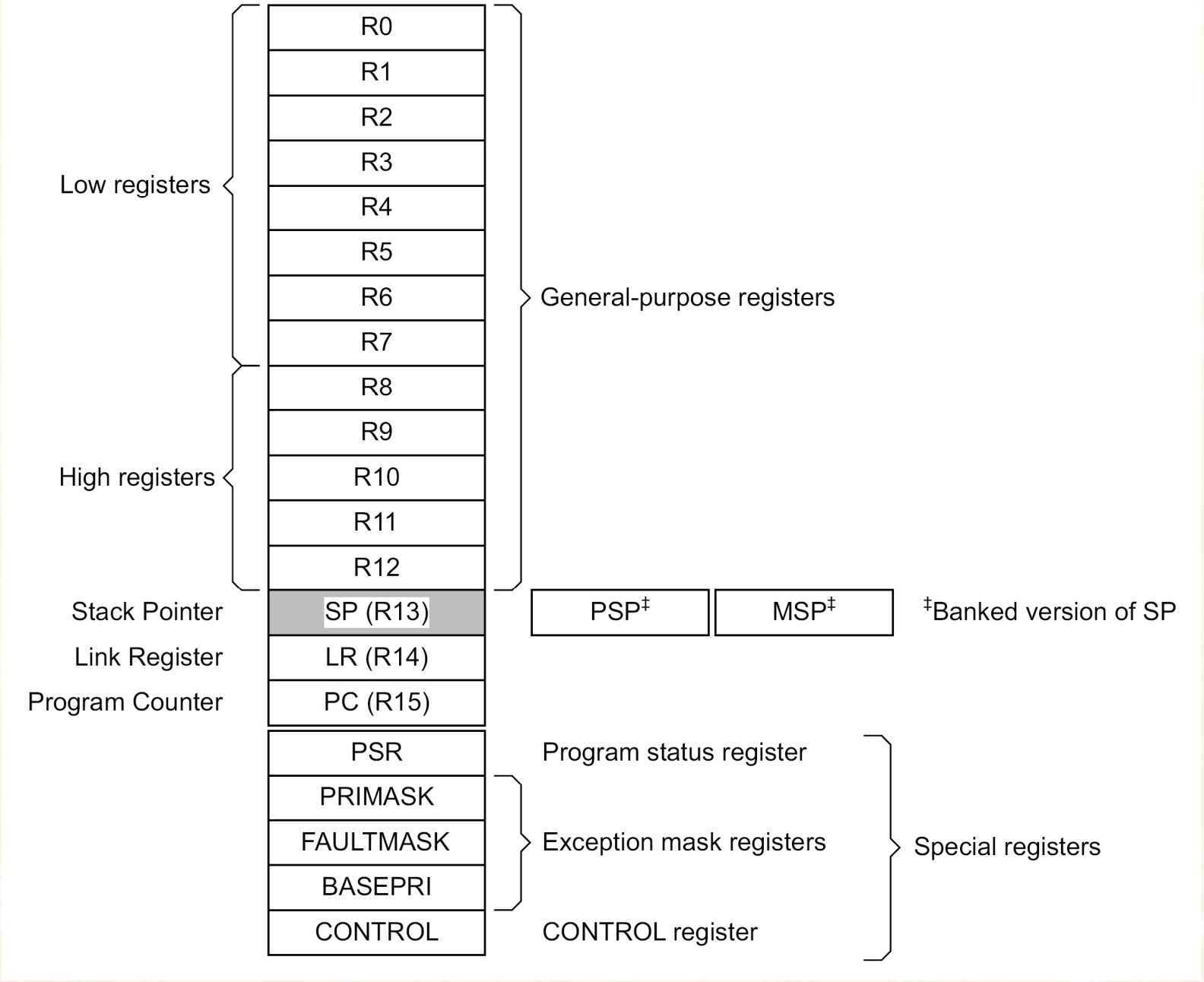
Řadič může být realizován:

- Sekvenčním obvodem s neměnnou logikou. Obvykle používaný u procesorů RISC (tzn. signálové a některé jednočipové procesory).
- Programový (mikroprogramovatelný) řadič s možností měnit logiku chování a efektivně realizovat obsluhu mnoha instrukcí za pomoci **mikrokódu.** Využívá se u procesorů CISC s různou dobou trvání instrukcí.

♣ Registry (zápisníková paměť, banky registrů)

- Staré procesory disponují malým počtem registrů obvykle 8 a střadačem (registr A). Mohou být dostupné pouze implicitně (MOV A,B) nebo jsou i paměťově mapované (MOV B,ACC tj. MOV B, adresa A). Některé mohou zastávat funkci ukazatele do paměti.
- Nové procesory mívají obvykle 32 registrů a střadačem může být každý. Některé mohou zastávat i další funkce např. ukazatele do paměti.
- Procesor ARM disponuje 13 všeobecně použitelnými registry (R0 až R12). Na rozdíl o běžných procesorů má **přímo dostupný čítač instrukcí PC (R15).**

REGISTRY ARM M3 A M4 A JEJICH FUNKCE



- **Stavový registr** - uchovává indikátory (příznaky) výsledku provedené operace. U ARM má označení **PSR**.

- **Standardní příznaky:** **u ARM**
  - Přenos (označen **C**)
  - Částečný přenos (není)
  - Přetečení (označen **V**)
  - Parita (není)
  - Znaménko (není)
  - Záporný nebo menší než (označen **N**)
  - Nula (označen **Z**)
  - Saturace (označen **Q**)

♣ **Ukazatel zásobníkové paměti SP** – je 8/16/32 bitový ukazatel do vnitřní nebo vnější datové paměti a slouží:

- Pro zápis/čtení **návratových adres** při volání/návratu z podprogramů nebo přerušení.
- Pro dočasné **bez adresné ukládání mezivýsledků** a **proměnných**
- Pro předávání mnoha proměnných do podprogramu.

Obecně ukazatel zásobníkové paměti ukazuje na:

- Neobsazenou nebo obsazenou adresu v datové paměti
- Při ukládání se SP dekrementuje nebo inkrementuje.

U ARM, je tvořen registrem **R13,** ukazuje na **obsazenou adresu** a při plnění se **dekrementuje** (adresa pro uložení hodnoty se zmenšuje).